

DOUBLE TROUBLE

PROJET C++
2020 – 2021

THOMAS CARDON • ALEXANDRE ARNIAUD • MOHAMED LABIDI
INES HAMIDI • ANGÈLE DERRIVES

TABLE DES MATIERES

01

BRAINSTORM

Idées, formation du groupe

02

PROGRAMMATION

Échecs et réussites, progrès

03

AUDIOVISUEL

Animation et Sons, rendre le jeu vivant

04

FINALITÉ

Critiques et retrospective

INTRODUCTION

Règles et modalités

Projet "Pac-Man"

Lors de notre premier semestre dans l'IUT Aix-Marseille en informatique, nous avons été chargé de créer de toutes pièces un petit jeu-vidéo inspiré du jeu d'arcade culte qu'on ne présente plus : Pac-man. Ainsi, nous avons quelques mois pour bâtir ce programme en C++ tout en respectant les normes imposées par nos professeurs:

- Les fichiers dans le dossier "Correc_Prof" **ne doivent pas être modifié.**
- Nos ajouts/modifications doivent apparaître dans le registre "nos_fichiers"
- Le jeu doit s'exécuter **dans un terminal** ou **dans une fenêtre** à l'aide du moteur graphique "minGL 2".
- Le projet doit être rendu avant le **22 Janvier.**
- Présence de documentation, générée par Doxygen.
 - La fonctionnalité "référence croisée entre le code source et la sortie" **doit être activée.**
 - La fonctionnalité "Graph des fonctions appelante et appelées" **doit être activée.**
 - L'archive **doit être nommée** selon une nomenclature **précise.**
 - Le projet doit respecter une certaine structure, arborescence.

Notre groupe



Alexandre Arniaud
(G3)



Thomas Cardon
(G3)



Mohamed Labidi
(G3)



Ines Hamidi
(G4)



Angele Derrives
(G4)

ETAPES DE CREATIONS



PROGRAMMATION

Le début de notre
projet



AUDIOVISUEL

Rendre le projet
vivant et attrayant



FINALITE

Retrospective et
critiques

A decorative grid of blue squares in various shades (light blue, medium blue, dark blue) arranged in a 6x6 pattern, with some squares missing or faded, creating a pixelated background effect.

Mais d'abord

..

Pour s'attaquer au projet, il est
question d'avoir des idées dans
un premier temps..

... Trouver des idées

... Pour ensuite les mettre à
exécution dans un second temps.



01

BRAINSTORM

Recherche d'idées, trier
le bon du mauvais

Formation du groupe, idées...



Le groupe part au départ du groupe 3 avec Mohamed, Thomas et Alexandre qui sont ensemble depuis la rentrée et qui ont eu l'occasion de faire connaissance. On a ensuite rencontré Ines et Angèle lors du premier partiel d'Algorithmique et Programmation et voyant que l'ambiance de travail avait été bonne et que le travail avait été efficace, il paraissait logique de se mettre ensemble pour ce projet.

Les idées sont venues assez naturellement. Le projet était de s'inspirer du célèbre jeu "Voleur / Contrebandidier" qui à lui-même inspiré Pac-man et d'ajouter les fonctionnalités voulues. Tout de suite ils nous a paru évident à tous de faire un jeu similaire à Pacman avec deux joueurs.

Pour le nom "Double Trouble", c'est simplement le fait que le jeu envoie au front deux joueurs pour se disputer la victoire!




LE BUT DU JEU ?

Le projet réalisé est une conception à la fois rétro avec une touche de modernité du jeu Pacman, sorti en mai 1980. Le principe est de faire le plus gros score en mangeant des “pac-gommes” et des bonus dans un labyrinthe tout en évitant les fantômes.

Le principe de **Double Trouble** est similaire, mais respecte tout de même la base du projet qui est un jeu avec 2 joueurs. Deux joueurs s'affrontent dans une carte où 4 monstres vont tout faire pour empêcher les joueurs de faire le plus grand score. Pour gagner ? il existe plusieurs stratégies possibles :

- Avoir un plus gros score que l'autre joueur
- Faire perdre toutes ses vies au joueur adverses avec plusieurs ruses
- Atteindre le premier le score de 9000 points

A decorative graphic on the left side of the slide consisting of a grid of blue squares of varying shades (dark blue, medium blue, and light blue) arranged in a pattern that tapers to the right.

02

PROGRAMMATION

Ajouts de fonctionnalités,
réussites, échecs...

OBJECTIFS PREMIERS

F

FACILITE

Permettre aux collaborateurs de s'intégrer facilement

D

DIDACTIQUE

Instruire facilement en ayant un code accessible

P

POLYVALENCE

Avoir un modèle de classes polyvalent afin d'intégrer les fonctionnalités de manière modulable

ETATS DE JEU SEPARES

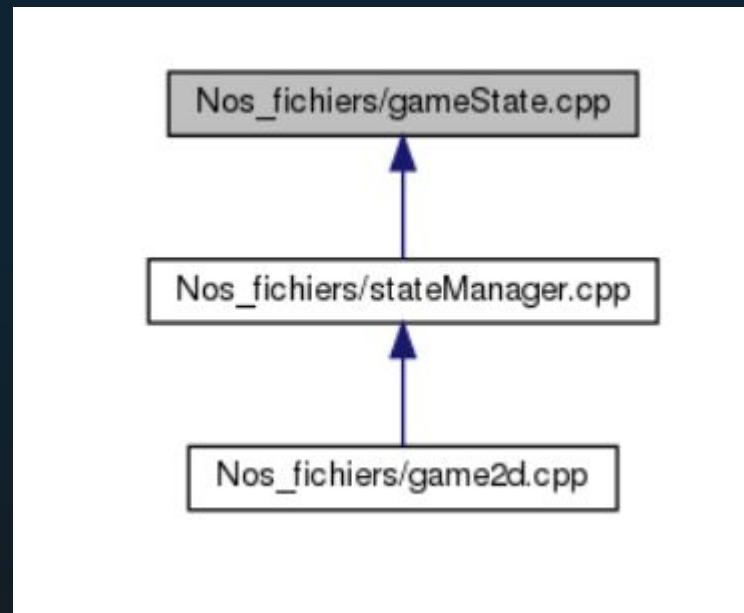
Afin d'avoir notre propre menu principal, nos crédits, et la logique du jeu de manière séparée, nous avons créé notre propre système

MODULABLE

- Il est très facile de rajouter des états
- Les états dépendent tous de **State.h**

SIMPLE

- Une fonction **State#render()**, **State#load()** et **State#update()** qui s'exécutent dès que StateManager reçoit l'ordre de changer d'état
- La logique du jeu est ainsi séparée des scènes qui dépendent d'images simples ou d'animations (crédits...)



DES ANIMATIONS CLAIRES

Le moteur graphique minGL 2 ne disposant pas de système permettant d'animer des textures facilement, nous avons dû nous résoudre à créer notre propre classe.

Ces animations se mettent à jour avec un nombre **delta**, désignant le nombre de millisecondes passées depuis le dernier frame; afin d'avoir un affichage plus ou moins précis des effets et personnages.

Public Member Functions

```
Animation ()  
Animation (nsGraphics::Vec2D pos)  
Animation (unsigned delay, bool alternate)  
Animation (unsigned delay, bool alternate, nsGraphics::Vec2D pos)  
void update (unsigned delta)  
    Updates animation, changes sprite according to delta and direction. More...  
void setCoordinates (int x, int y)  
    Sets coordinates on the window. More...  
void setPosition (int x, int y)  
    Sets position on the map. More...  
void setPosition (nsGraphics::Vec2D pos)  
    Sets position on the map. More...  
void render (MinGL &window)  
    Renders resources. More...
```

Public Attributes

```
unsigned delay = 500  
    Delay between frames. More...  
unsigned currentSprite = 0  
bool alternate = true  
    Allows animation to go from start to end, then end to start, and vice versa.  
std::vector< nsGui::Sprite > sprites  
    Sprites list. More...
```

UNE LOGIQUE DE JEU

La classe **GameState**, qui s'étend de notre classe *State*, gère la logique du jeu ainsi que les différentes données importantes

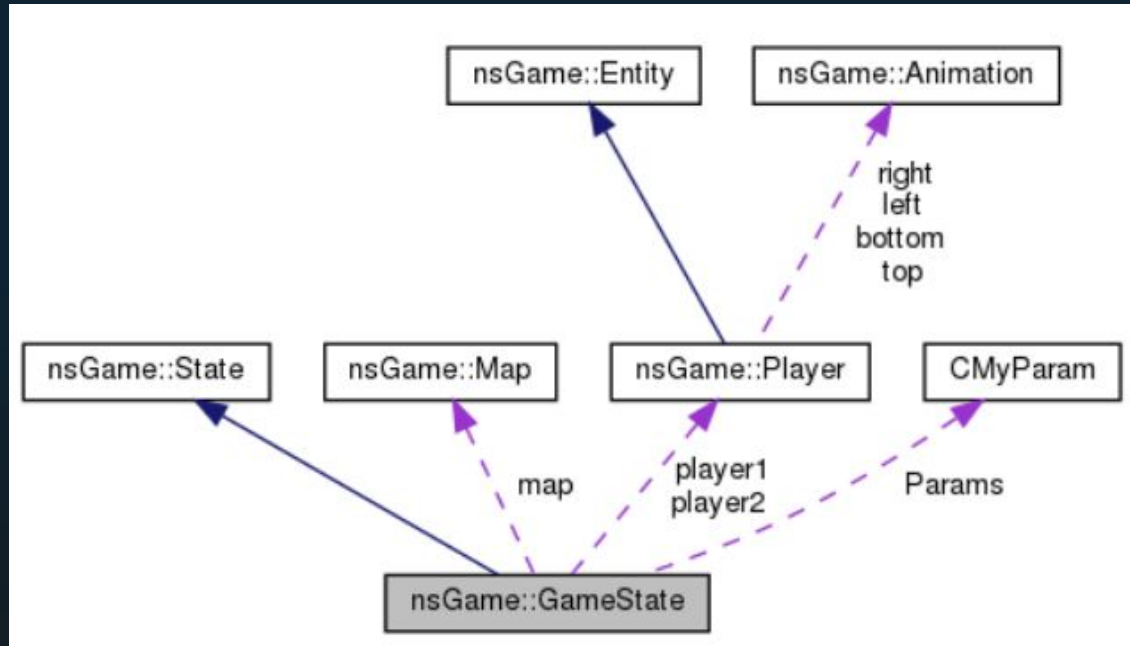


Diagramme de collaboration entre les classes

UN TABLEAU DES SCORES INNOVANT

*La classe **GameState** gère aussi la sidebar - barre à droite - ainsi que son tableau des scores; un objet qui réfère chaque chiffre allant de 0 à 9 à une image, minGL2 ne supportant pas les polices personnalisées*

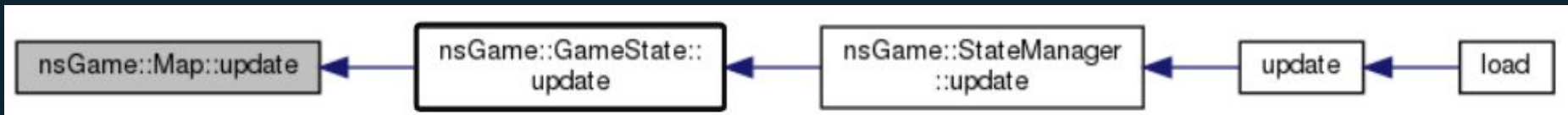


Les coeurs des joueurs

- Affichage des coeurs des joueurs - de 0 à 3 coeurs
- Un score des joueurs affiché de manière intelligente
- Un côté rétro

UNE CARTE PERCUTANTE

La classe **Map**, appelée depuis la logique du jeu **GameState**, gère les coordonnées des entités ainsi que les objets.



Modélisation de l'appel de la classe (ici pour la fonction **Map#update()**)

- 8 items (fruits, power-up) à manger max.
- Un seul power-up - permettant de manger les monstres
- Une carte chargée aléatoirement depuis un dossier, où chaque caractère est lu et est inséré dans la grille avec son bloc associé (murs, sols...)
- De la nourriture posée à tous les endroits vides
- Une grille, **CMap** (reprise du jeu de base), répercutée dans **Map#render()**

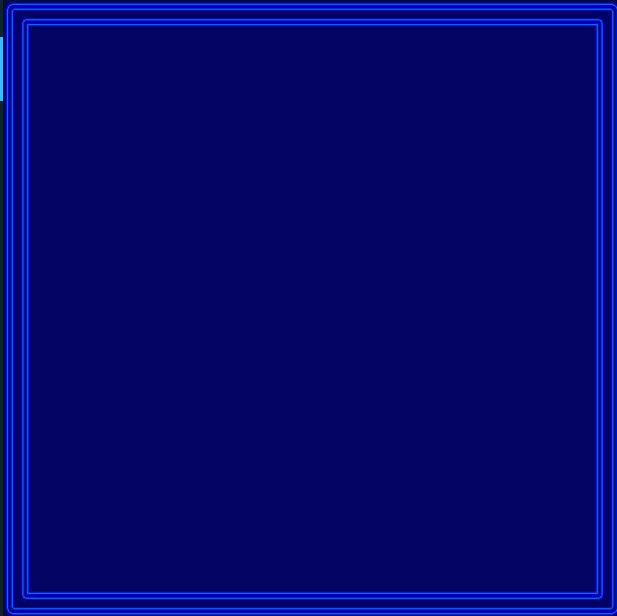
03

GRAPHISMES

Ou “comment rendre
vivant quelque chose qu’il
ne l’est pas”



Préparer la scène



■ Des règles simples

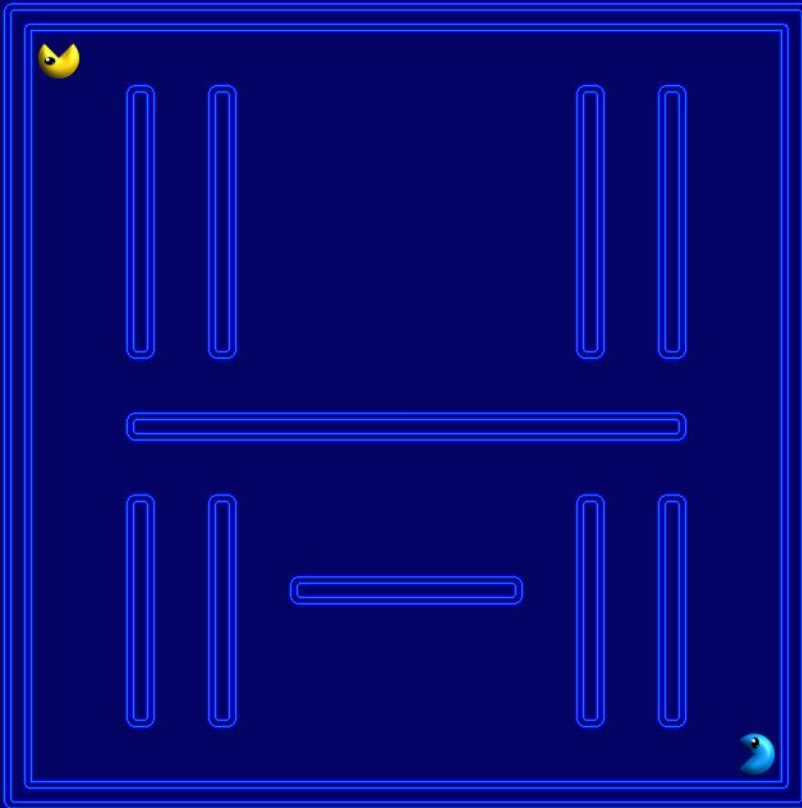
Taille limité, chaque mur se termine

■ Modifications

Rien ne bouge, et pourtant tout change

La création de carte est facile à prendre en main, et devient presque intuitif au fil du temps.

Mise en pratique



La carte jouable (à gauche)
Et sa représentation dans le
fichier .map (à droite)

Chaque caractère
correspond
à une case, qu'elle soit vide,
représentée alors par un 0
ou pleine, une multitude
d'obstacle pouvant alors
prendre place dans le jeu.

Ces fichiers sont lu dans un
dossier map, et ils un d'entre
eux est choisi aléatoirement
par le jeu. Tout le monde
peut
donc en créer.

```
1 | /=====\  
2 | 00000000000000000000 |  
3 | 00^0^00000000^0^00 |  
4 | 00|0|00000000|0|00 |  
5 | 00|0|00000000|0|00 |  
6 | 00|0|00000000|0|00 |  
7 | 00|0|00000000|0|00 |  
8 | 00|0|00000000|0|00 |  
9 | 00-0-00000000-0-00 |  
10 | 00000000000000000000 |  
11 | 00#=====~00 |  
12 | 00000000000000000000 |  
13 | 00^0^00000000^0^00 |  
14 | 00|0|00000000|0|00 |  
15 | 00|0|0#=====~0|0|00 |  
16 | 00|0|00000000|0|00 |  
17 | 00|0|00000000|0|00 |  
18 | 00-0-00000000-0-00 |  
19 | 00000000000000000000 |  
20 | (=====)  
21 |
```

Règles des cartes

0 = cellule sans rien

/ = bloc coin supérieur gauche

\ = bloc coin supérieur droit

(= bloc coin inférieur gauche

) = bloc coin inférieur droit

^ = bloc ouverture mur vertical

| = bloc mur vertical

- = bloc fermeture mur vertical

= bloc ouverture mur horizontal

= = bloc mur horizontal

~ = bloc fermeture mur horizontal

Chaque caractère ayant une utilisation précise, les obstacles et les objets contribuent, de ce fait, à la diversité des cartes lors de leur création.

Malgré la scène fixe, il est alors impossible d'être à court d'idées pour les cartes et les niveaux à suivre.

```
Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py right-3.png right-3.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py right-4.png right-4.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py right-5.png right-5.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py right-6.png right-6.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py top-1.png top-1.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py top-2.png top-2.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
Done!
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$ python3 ./img2sl.py top-3.png top-3.l2s
Source Image: .png
Output Image: .sl2
Image size is 32x32

Starting conversion...
alexandre@galex-linux:~/Documents/projet-c--2020-amu/GX_Hamidi_Derrives_Arnaud_Labidi_Cardon/Mes_Fichiers/res/entities/player2$
```

Un long processus nécessaire à l'importation d'images

Le moteur graphique minGL2 utilise son propre format de fichier, et contient donc un script python permettant de convertir nos images en .png en images compatibles pour le moteur. C'est une étape longue qui peut être facilement raccourci grâce à un script cependant.

Animation des images

Utilisation de classe permettant
d'effectuer des animations
grâce à une rotation de sprites
effectuées toutes les n
secondes

```
/**
 * \file   animation.h
 * \author Thomas Cardon
 * \date   12 janvier 2020
 * \version 1.0
 * \brief   Animation
 */
#include <mingl/gui/sprite.h>

namespace nsGame {
    /**
     * @class Animation
     * @brief Defines animations
     * @author Thomas Cardon
     */
    class Animation
    {
    private:
        unsigned _currentTime;
        bool inReverse = false;

        nsGraphics::Vec2D _pos;

    public:
        unsigned delay = 500, currentSprite = 0;
        bool alternate = true;

        std::vector<nsGui::Sprite> sprites;

        Animation();
        Animation(nsGraphics::Vec2D pos);
        Animation(unsigned delay, bool alternate); // 642
        Animation(unsigned delay, bool alternate, nsGraphics::Vec2D pos);

        /**
         * @brief Updates animation, changes sprite according to delta and direction
         * @fn void update(unsigned delta);
         */
        void update(unsigned delta);

        /**
         * @brief Sets coordinates on the window
         * @fn void setCoordinates(int x, int y);
         */
        void setCoordinates(int x, int y);

        /**
         * @brief Sets position on the map
         * @fn setPosition(int x, int y);
         */
        void setPosition(int x, int y);

        /**
         * @brief Sets position on the map
         * @fn setPosition(nsGraphics::Vec2D pos);
         */
        void setPosition(nsGraphics::Vec2D pos);

        /**
         * @brief Renders resources
         * @fn void render(MinGL & window);
         */
        void render(MinGL & window);
    };
}
```



04

FINALITE

Critique, retrospective

AUTOCRITIQUE



TRAVAIL EN ÉQUIPE
MÊMES OBJECTIFS, ENVIES
PROGRAMMATION C++
LIBERTÉ A LA CRÉATIVITÉ

PAS DE CRÉNEAUX EN COMMUN
MANQUE DE DYNAMISME, D'AUTONOMIE

