Documentation Technique : Projet B-OOP-400 Arcade

Epitech

Année 2021

Membres : Corentin Petrau, Lyvia Mallereau, Raphael Risser

Sommaire

I- Introduction

Le présent document est à l'intention des futurs utilisateurs pour leur permettre de comprendre l'utilisation des différentes parties du programme.

Ce document décompose le programme en différentes parties afin de mieux le décrire. Il n'est qu'une explication partielle du code du programme et ne sera en aucun cas une démonstration de son intégralité ni même des différents résultats qui peuvent en découler.

Les rédacteurs et programmeurs ne seront pas tenus responsables des différents bugs qui peuvent résulter de l'utilisation du programme. Les utilisateurs doivent le prendre dans sa globalité, comprendre son fonctionnement et l'adapter à leur propre programme.

II- Présentation du projet

Le projet B-OOP-400 Arcade est un projet pour les étudiants de deuxième année (en l'occurrence promotion 2024) au sein de l'école Epitech.

Ce projet a pour but de recréer, de manière simplifiée, une borne d'arcade (sur son ordinateur) prenant en compte plusieurs jeux (en l'occurrence deux ici présents). De plus, il faut prendre en compte le changement en temps réel des différentes librairies, pouvoir changer facilement de jeux (et leur graphismes avec les différentes librairies disponibles).

D'un point de vue pédagogique, ce projet s'effectue en groupe de trois personnes, chaque groupe de travail ayant sa propre façon de concevoir le projet. Toutefois, ce projet peut être apparenté à un projet de type "modulaire": l'une de ses spécificités est de prendre en considération le programme d'autres groupes afin que les programmes se modulent entre eux.

Bien que comparable à un projet de type "jeu vidéo", ce projet est bien plus complexe que de recréer des jeux d'arcades. Il faut prendre en compte beaucoup de paramètres et finalement l'aspect "jeu" n'est pas aussi important que le nom l'indique.

III- Partie Core

La partie Core est la plus complexe du programme. En effet, cette partie doit le plus s'adapter aux programmes des autres groupes.

L'architecture doit être pensée de manière claire et elle doit faire preuve d'une grande adaptabilité.

Pour ce faire, notre groupe de travail s'est organisée de la manière suivante :

- Plusieurs dossiers ont été créés: un dossier de documentation (nommé **doc**), un dossier de jeu (nommé **games**), un dossier d'includes (nommé **inc**), un dossier de librairie (nommé **LIB**), et un dossier source (nommé **src**).
- Chaque dossier contient des fichiers (ou d'autres sous-dossiers) :
    - Le dossier ***doc*** contient un fichier de type txt permettant aux joueurs de connaitre les commandes de jeux.
    - Le dossier ***game*** contient deux sous-dossiers pour chaque jeu, Pac-Man et Snake, ainsi qu'un Makefile.
    - Le dossier ***inc*** contient tous les fichiers essentiels au fonctionnement du programme, notamment pour le fonctionnement du Core en lui-même.
    - Le dossier ***LIB*** il contient quatres sous-dossiers, un pour le menu de l'arcade, un pour la librairie Ncurse, un pour la librairie SDL2 et un pour la SFML et enfin un Makefile.
    - Le dossier ***src*** contient deux fichiers essentiels au lancement du jeu, grâce au fichiers main.cpp et core.cpp.
    - A l'extérieur de ces dossiers se trouve ce présent document. Le ***.gitignore*** permet d'ignorer les fichiers non essentiels au projet (notamment des fichiers cachés ou les résidus du Makefile). Et enfin le ***Makefile*** va permettre de compiler tous les autres Makefile.
- Chaque membre du groupe de travail s'est réparti le travail le plus équitablement possible. Les différentes compétences de chacun ont permis une entraide selon les difficultés rencontrées.

La partie Core: Le Core permet le fonctionnement global du programme : il va mettre en relation toutes les parties du programme et les faire fonctionner entre elles.

Pour schématiser le tout:

Librairies <-> ***Core*** <-> ***Jeux***.

De ce fait, on remarque ici que le Core prend bien tous les paramètres en fonction. Chaque script et chaque fichier vont correspondre à un point précis dans le fonctionnement du programme.

Nous vous invitons à voir notre diagramme de classe ainsi que notre diagramme de séquence afin d'avoir un meilleur apperçu de notre programme (pièces jointes ***Diagramme_Classe_Arcade.pdf***, ***Diagramme_Sequence_Arcade.pdf*** dans le dossier ***doc***).

IV- Partie Librairies Graphiques

Trois librairies graphiques ont été choisies.

- La **SDL2** : il s'agit d'une bibliothèque logicielle libre. Elle est souvent utilisée pour créer des applications multimédias en deux dimensions pouvant comprendre du son commes les jeux vidéos, les démos graphiques, les émulateurs, etc...
- la **Ncurse** : il s'agit d'une bibliothèque libre fournissant une API pour le développement d'interfaces utilisateur à menu déroulant, en utilisant les caractères et couleurs d'un mode semi-graphique. Ce type d'interface se conçoit de manière indépendante du terminal, mais il accélère le rafraîchissement d'écran, diminuant par là le temps de latence que subissent d'ordinaire les utilisateurs de shells à distance.
- la **SFML** : il s'agit d'une interface de programmation destinée à construire des jeux vidéo ou des programmes interactifs. Elle a, entre autres, pour but de proposer une alternative à la SDL. Elle a également la particularité de fournir un graphisme 2d accéléré en utilisant OpenGl en interne, qui permet à l'utilisateur de s'affranchir de la gestion d'une pseudo-3d.

Le but du projet est de recréer des librairies dynamiques, adaptées au jeu et qui peuvent être changées en cours de partie. En outre, chaque librairie doit posséder des fonctions **types** afin de les adapter facilement aux **Core** et que celui-ci les prennent bien en compte.

V- Partie Jeux

Pour ce projet, deux jeux ont été sélectionnés :

- **Pac-Man** : il s'agit d'un jeu vidéo créé par Toru Iwatani pour l'entreprise japonaise Namco, sorti le 22 mai 1980. Le but du jeu est très simple : il suffit de déplacer Pac-Man à l'intérieur d'un labyrinthe afin de lui faire manger toutes les Pac-gommes qui s'y trouvent en évitant d'être touché par les fantômes. Il s'agit d'un véritable repère dans l'histoire des jeux d'arcades mais aussi dans l'histoire du jeu vidéo en lui-même, créant à lui tout seul, un nouveau genre, faisant suite à Pong, Space Invaders et Asteroids.
- **Nibbler** : il s'agit d'un jeu vidéo de labyrinthe créé par SNK Corporation et sorti sur borne d'arcade (sur le système Rock-Ola) en 1982. On y contrôle un serpent, sur le même principe que le jeu Snake, mais avec une map plus approfondie.

Pour ce projet, on parle de ***jeux*** mais ils sont codés de la même manière, avec chaque spécificités propres à eux-mêmes.

VI- Partie Menu

Le menu est une des parties les plus importantes du programme, car il s'agit de la première fonction que l'utilisateur va voir en lançant le programme. Il faut donc qu'il soit clair et compréhensible pour l'utilisateur, afin qu'il puisse naviguer entre les différents éléments (en l'occurrence les **librairies** et les **jeux**).

Pour ce faire, nous avons conçu le menu de notre programme en **Ncurse**. La raison de ce choix est simple:

- Raison esthétique : la combinaison ***Arcade***/ ***Terminal*** est appréciable et dénote une qualité d'image, même simpliste, non négligeable qui, du point de vue de l'utilisateur, est simple à comprendre.
- Code : la **Ncurse** fut facile à mettre en place, car deux membres du groupe ont des facilités dans ce langage.

Plus précisément, le menu doit comporter plusieurs éléments :

- Les librairies: l'utilisateur doit être capable de choisir sa librairie dès le départ.
- Les jeux: l'utilisateur doit être capable de choisir ses jeux.
- Le nom de l'utilisateur.
- Les scores des parties joués par les utilisateurs.
- L'heure et la date.

VII- Description technique du code

Dans ce chapitre, nous allons expliquer en détail le code de notre programme. Il s'agit principalement de pseudo-code mais nous tenterons d'expliquer clairement notre approche afin de faciliter la compréhension pour les futurs utilisateurs et développeurs qui souhaiteront utiliser notre projet.

Nous verrons dans un premier temps la partie Core, la partie librairies graphiques, la partie jeux et enfin la partie menu.

Sommaire Sous-parties

1. Partie Core

Le Core est la partie la plus complexe mais il s'agit aussi de la partie la plus importante du programme.

Si on prend le fichier ***core.cpp*** dans le dossier **src**, on remarque plusieurs fonctions:

- La première fonction (***arcade::Core::Core***) est le constructeur qui va permettre de récupérer les chemins vers les différentes librairies graphiques et les jeux et va ainsi les stocker dans un vecteur chacun.
- La deuxième (***arcade::Core::SetGraphical_and_Game***) prend le dossier où se situent les différentes librairies, va vérifier s'il s'agit d'un jeu ou d'une librairie

graphique et va la mettre dans le bon vecteur associé.
- La troisième (***arcade::Core::launch_ncurses***) va permettre d'ouvrir la librairie du menu et vérifier qu'elle existe et qu'on peut l'ouvrir.
- La quatrième (***arcade::Core::load_graphical***) prend en paramètre l'index de la librairie que l'on souhaite ouvrir et la stocker dans la variable qui contient le **Graphical_Module** afin de pouvoir l'utiliser tout au long du programme.
- La cinquième (***arcade::Core::load_game***) va faire la même chose que la précédente mais va stocker les variables dans le **IGame_Module**.
- La sixième (***arcade::Core::game_loop***) est la boucle de jeu qui va permettre aussi de changer les librairies au **run_time**.
- La septième (**arcade::Core::menu**) va permettre de lancer le menu et qui va récupérer une paire de **int**.

On voit bien ici que chacune de ces fonctions est essentielle. Ce sont chacunes des boucles qui vont appeler les fonctions qui vont elles-mêmes appeler le reste des fonctions.

2. Partie librairies graphiques

Le dossier LIB : comme on peut le constater chaques librairies (en plus du **Menu**) à son propre dossier, chacun comportant eux même un **Makefile**, un **.cpp** et un **.hpp**.

Il s'agit de librairie dynamique, les fonctions contenues dans les fichiers doivent avoir pour unique but de s'implémenter aisément à d'autres programmes.

Chaque fichiers de chaques librairies, comportent des fonctions types :

- Un constructeur.
- Une fonction ***stop***: qui va quitter la fenêtre.
- Une fonction ***init***: qui comme son nom l'indique va initialiser toute les fonctions importantes (comme l'initialisation de la fenêtre, les couleurs, les touches de clavier, ...).
- Une fonction ***draw***: qui va permettre de dessiner les entités essentielles.
- Une fonction ***Input::Key***: cette fonction va permettre de mettre en place les touches clavier permettant de jouer.
- Une fonction ***display***: cette fonction va rafraîchir la fenêtre de la librairie.
- Une fonction ***extern "C"***: cette fonction va permettre de rendre la librairie dynamique et lui allouer la mémoire qui lui faut pour fonctionner.

3. Partie Jeux

Chaque dossiers (dans le dossier ***game***), contient, tout comme les librairies graphiques, un **Makefile**, un fichier **.cpp**, un fichier **.hpp** mais ces dossiers vont aussi posséder une map en **.txt** et un dossier **assets** (qui va contenir les images et les textures du jeu).

Comme pour les librairies graphiques, la plupart des fonctions dans les fichiers sont de type *générique*:

- Un constructeur.
- Une fonction ***init***: qui va initialiser les valeurs utiles telles que la map, la position, ...
- Une fonction ***vector[...]update*** qui va permettre de mettre à jour les données du jeu en prenant en compte de nombreux paramètres comme la position, la direction du personnage du joueur, ...
- Une fonction ***GetName*** qui va récupérer le nom du jeu pour l'affichage.
- Les fonctions ***go_up***, ***go_down***, ***go_right***, ***go_left*** qui vont déterminer la direction à laquelle le personnage du joueur va aller.
- Une fonction ***get_socre*** qui va tout simplement récupérer le score du joueur.
- Une fonction ***stop*** qui va stopper le jeu.
- Une fonction ***extern "C"*** qui est un rappel vers les librairies dynamiques.

Par la suite, nous allons retrouver des fichiers spécifiques aux jeux :

- Pour le Nibbler: le principe du jeu étant relativement simple, le plus complexe ici est de récupérer la bonne position du serpent (personnage du joueur) et aussi à quel moment le bout de sa queue va grandir à chaque fois qu'il récupère une pomme. On aura alors plusieurs fonctions dans ce but :
  - La fonction ***move_body_tail***: va faire bouger le serpent.
  - La fonction ***update***: comprend toutes les déclarations des **assets** (c'est-à- dire les textures nécessaires au fonctionnement du jeu).
- Pour le Pac-Man: bien que certaines fonctions semblent être relativement identiques, on peut ici observer des différences dans les fonctions ***go_up***, ***go_down***, ***go_left*** et ***go_right***. Le personnage du joueur va pouvoir récupérer les *". "* dans toute la map ainsi que les *pac-gommes*.

Chaque **.hpp** pour ces jeux possède les mêmes prototypes de fonctions mais sont différents grâce au ***private*** de leur classe.

4.Partie Menu

Le menu à été réalisé en ***Ncurses***. En outre, il permet de mettre en place une architecture visuelle simple à comprendre. Tout comme une librairie ou un jeu, son dossier va contenir un ***Makefile*** et un simple fichier ***.cpp***.

Le ***.cpp*** va contenir des fonctions types, comme un constructeur, l'initialisation de la fenêtre, des couleurs, des boites, les touches clavier, …

Fonctions spécifiques du fichier faisant fonctionner le menu :

- La fonction ***loop_menu*** permet de créer la boucle du menu, avec ses différentes indications.
- La fonction ***one_sec_loop*** permet de quitter ou valider un choix dans le menu, il s'agit d'une boucle de fonctionnement d'événement pour le menu.
- La fonction ***create_new_board*** permet de créer une boite qui va contenir les informations de jeux, libs ou autres.
- La fonction ***rtrim*** sert à enlever les espaces et les **\t** à droite.
- La fonction ***currentDateTime*** permet de récupérer la date et l'heure sur le terminal Ncurse.
- La fonction ***header***: il s'agit de l'endroit où vont s'écrire la date et le nom d'utilisateur.
- La fonction ***select*** permet l'utilisation, via les touches de clavier, des différents choix que contient le menu (jeu, librairies, quiter, ...).
- La fonction ***readscore*** permet de lire et d'afficher le score.
- La fonction ***scores*** permet d'afficher le score.
- La fonction ***disp_qna*** permet d'afficher le guide pour utiliser le menu.
- La fonction ***disp_maps***: cette fonction affiche l'intégralité des informations du menu.
- Une fonction ***extern "C"*** qui va rendre le menu dynamique comme les librairies.

VIII- Implémenter votre code, vos Librairies, vos jeux

Cette section est dédiée à votre future implémentation de votre code. Dans cette section, il sera décrit de manière détaillée comment vous allez pouvoir implémenter vos propres librairies et vos jeux avec le ***Core*** déjà présent dans notre programme.

Sachez que n'importe quel jeux peut s'adapter à ce programme, mais partez de préférence sur des jeux de types ***Arcades***.

N'importe quelle librairie peut aussi être implémentée.

Pour que votre programme fonctionne de manière optimale, il vous sera nécessaire d'adapter vos fonctions du *Core* et des fonctions déjà prédéfinies dans les différents *.hpp* et *.cpp* qui définissent le *Core*. De plus,respecter l'architecture déjà présente est essentielle. Libre à vous de vouloir changer la nature et le fondement de l'architecture du projet mais il risque d'en résulter des conflits de fusion entre les différents fichiers déjà présents et les vôtres.

Il ne vous est pas nécessaire de modifier le ***Core*** ni le ***menu***. En effet, les deux ont été conçus afin que d'autres personnes puissent implémenter leurs futurs jeux et librairies graphiques.

Ne vous mettez pas de barrière et si vous souhaitez modifier le ***Core*** et le ***menu***, libre à vous de le faire. Ici, il ne sera pas détaillé les modifications qui peuvent être apportées sur ces deux parties, nous allons nous concentrer sur les jeux et sur les librairies graphiques.

Sommaire Sous-parties 2

1.Partie Jeux

Comme nous l'avons déjà indiqué, n'importe quel jeu peut être intégré dans le programme. Basez vous sur l'architecture pour vos fichiers, Makefile, et assets.

En outre, nous vous recommandons de baser votre code à partir de jeux simples à mettre en place dans un premier temps. Il vous est tout à fait possible d'implémenter des jeux plus complexes dans le futur, mais nous vous recommandons d'améliorer la partie ***Core***.

Pour votre implémentation, n'oubliez pas les points suivants:

- Pensez à créer des fonctions "types" pour vos jeux, mais n'oubliez pas de mettre certains paramètres en privé dans vos ***.hpp*** afin de ne pas voir de conflits entre les différents jeux que vous allez complémenter.
- Nommez correctement vos fichiers/textures.
- Les noms des fonctions doivent être les mêmes : pour que les librairies puissent fonctionner correctement ainsi que vos jeux, il est nécessaire que les fonctions soient les mêmes. Pour le fonctionnement de chacun des jeux, il est important de compléter ces fonctions avec des paramètres qui leur sont propres. Par exemple, si on prend nos jeux, le Nibbler et le Pac-Man vont se déplacer de la même manière, manger de la même manière. Toutefois on notera des différences, le Pac-Man va manger tous les points devant lui et le serpent du Nibbler va uniquement manger les pommes qu'il trouve (et grandir à chaque bouchée).
- Créer une architecture claire et fluide pour vous retrouver dans vos fichiers: il y a peu de fichiers mais chacun à son importance et est placé au bon endroit pour bien s'y retrouver.

2.Partie Librairies Graphiques

Comme pour les jeux, les librairies graphiques doivent comporter les mêmes points de modification. Il n’est pas nécessaire de vous baser sur le même fonctionnement que nos propres fichiers. Cependant, si vous souhaitez avoir une optimisation de code pour éviter les conflits, il vous faudra peut-être revoir une partie du code des librairies pour les adapter au vôtre.

Comme pour les jeux, une clarification des noms des fonctions sera optimale pour vous permettre de mettre en place vos librairies. N'importe quelle librairie peut être implémentée, mais certaines librairies 3D peuvent entrer en conflit avec le programme. Libre à vous de modifier ces fichiers, mais attention aux fonctions que vous rajoutez et les conséquences qui peuvent en découler.

IX- Conclusion

Nous tenons à mettre en garde que ce projet n'est pas à but de tricherie, il s'agit d'une aide que vous pouvez utiliser pour votre propre projet et voir si vos librairies/jeux fonctionnent autant sur notre programme que sur le votre.

Pour conclure ce document, nous tenons à remercier tous les futurs utilisateurs ainsi que les membres des autres groupes qui nous ont aidés tout le long de ce projet et avec qui nous avons beaucoup échangés, sur nos difficultés mais aussi sur nos avancements.

X- Sources

- Wikipédia.
- Source interne Epitech.
- StackOverFlow.
- GeekForGeeks.
- Documentation SFML.
- Documentation SDL2.
- Documentation Ncurse.