Documentation Arcade

Sommaire

Un projet de Victor Sousa, Gaëtan Léandre et Lucas Gambini

EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Introduction

L'arcade est un projet de deuxième année à Epitech. Le but est de recréer une borne d'arcade comportant au minimum deux jeux et trois librairies graphiques différentes.

Chaque jeu ou librairie graphique doit pouvoir être changé en cours d'exécution, tout en sauvegardant les données de position du joueur pour les jeux, par exemple.

Le projet est fourni avec un Nibbler et un SolarFox, ainsi qu'avec une implémentation graphique en NCurse, SFML et OpenGL.

Compilation

Le projet est fourni avec un Makefile comportant plusieurs règles afin de compiler différentes parties du projet comme le binaire arcade, ou les libraires de jeux ou encore les libraires graphiques.

Voici un tableau résumant les différentes possibilités de compilation :

Règle	Action
all	Compile l'arcade et les jeux
clean	Supprime les .o de l'arcade et des jeux
fclean	Supprime les .o et les binaires de l'arcade et des jeux
re	Appel la règle fclean puis all
global	Compile l'arcade, les jeux et les librairies graphiques
global_clean	Supprime les .o crées
global_fclean	Supprimes les .o et les binaires crées
global_re	Appel la règle global_fclean puis global
games	Compile les jeux
games_clean	Supprime les .o des jeux
games_fclean	Supprime les .o et les binaires des jeux
games_re	Appelle la règle games_fclean puis games
snake	Compile le Snake / Nibbler
snake_clean	Supprime les .o du Snake
snake_fclean	Supprime les .o et le binaire du Snake
snake_re	Appelle la règle snake_fclean puis Snake
solar	Compile le SolarFox
solar_clean	Supprime les .o du SolarFox
solar_fclean	Supprime les .o et le binaire du SolarFox
solar_re	Appelle la règle solar_fclean puis solar
graph	Compile les librairies graphiques
graph_clean	Supprime les .o des librairies graphiques
graph_fclean	Supprime les .o et les binaires des librairies graphiques
graph_re	Appelle la règle graph_fclean puis graph
opengl	Compile la OpenGL
opengl_clean	Supprime les .o de la OpenGL
opengl_fclean	Supprime les .o et le binaire OpenGL
opengl_re	Appelle la règle opengl_fclean puis opengl
sfml	Compile la SFML
sfml_clean	Supprime les .o de la SFML
sfml_fclean	Supprime les .o et le binaire SFML
sfml_re	Appelle la règle sfml_fclean puis sfml
ncurses	Compile la NCurse
ncurses_clean	Supprime les .o de la NCurse
ncurses_fclean	Supprime les .o et le binaire NCurse
ncurses_re	Appelle la règle ncurse_fclean puis ncurse

Enfin, le projet nécessite certaines dépendances afin d'être compilé avec succès :

- Lib SFML
- Lib SDL
- Lib NCurse
- Lib OpenGL
- Lib GLU
- Lib GLFW
- Lib GLEW
- Lib GLFT
- Lib DevIL
- Lib Assimp

Utilisation de base

Dans un terminal, exécuter simplement la commande suivante après avoir compilé le binaire « arcade » ainsi qu'au moins une libraire graphique (voir section compilation) :

```
λ arcade_doc cpp_arcade → λ git master* → ./arcade
Usage: ./arcade [Startup Library]
```

Le binaire nécessite donc qu'on lui précise sur quelle librairie graphique il doit se lancer par défaut :

```
λ arcade_doc cpp_arcade → λ git master* → ./arcade ./lib/lib_arcade_opengl.so
```

Les librairies dynamiques graphiques doivent être présentes dans le dossier relatif « ./lib/ » pour pouvoir être prises en compte et chargées une fois l'arcade exécuté. Pour les librairies de jeux, celles-ci doivent être dans le répertoire « ./games/ ».

```
λ arcade_doc lib → λ git master* → pwd ; ls
/home/gambin_l/Shared/cpp_arcade/lib
lib_arcade_ncurses.so  lib_arcade_opengl.so  lib_arcade_sfml.so  NCurses  OpenGL  SFML
λ arcade_doc lib → λ git master* →
```

Architecture du programme

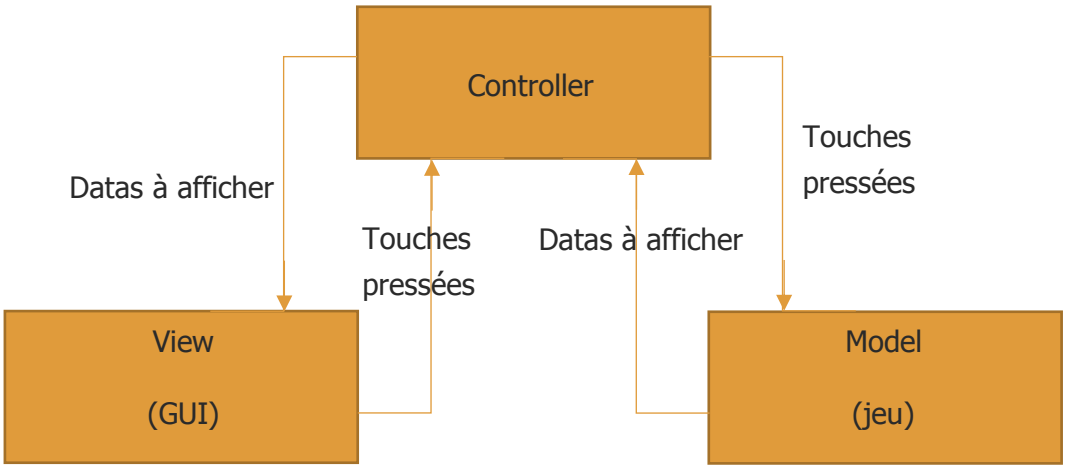
L'arcade charge au démarrage les jeux présents dans le répertoire ./games/ et les librairies graphiques dans le répertoire ./lib/.

Chaque entité externe au binaire arcade doit être une librairie dynamique partagée au format « .so ». Ainsi, en suivant la norme d'implémentation expliquée dans cette documentation vous serez capable d'ajouter vos propres libraires graphiques et vos propres jeux à notre arcade.

Le programme, pour qu'il soit fonctionnel doit être composé de trois parties :

- L'arcade, ce binaire est le cœur du programme, il va s'occuper de faire transiter les informations entre le jeu et le GUI (graphical user interface). Cette partie est nommée « controller ».
- Les jeux ou le menu donnent au « controller » les éléments qui doivent être affichés par notre troisième partie.
- Les librairies graphiques reçoivent du « controller » les éléments à afficher à l'écran et renvoient les touches pressées au jeu.

Notre programme respecte donc un pattern MVC (model-view-controller).



Les jeux doivent suivre une implémentation bien précise, pour cela une interface C++ est fournie (IGame) contenant les méthodes obligatoires au bon fonctionnement de celui-ci dans l'arcade. Évidement votre jeu doit hériter de cette interface.

De même pour les librairies graphiques, une interface (IGraph) est fournie vous imposant des méthodes nécessaires.

Les données circulant entre la vue et le model sont des « datas » définies par l'interface « IData » de laquelle hérite une classe abstraite « AData ». Puis, trois types de « datas » sont définies :

- Les « datas » visuelles, qui vont être affichées, celles-ci sont guidées par la classe abstraite « AVisual » de laquelle héritent différents objets tel que le Cube ou le Texte.
- Les « datas » concernant les scènes 3D, celles-ci héritent de « AScene » et ne contiennent que l'objet Camera et « Light » pour les lumières.
- Enfin, une classe permet de créer des objets audio, tel que de la musique ou un son particulier afin qu'il soit joué par la librairie graphique courante.

Norme d'implémentation

Cette section a pour but d'expliquer comment vous pouvez créer un jeu ou une GUI compatible avec notre binaire arcade.

Tout d'abord, pour créer un jeu vous devez créer la classe principale de votre jeu, celle-ci va hériter de notre interface « IGame » et vous devrez implémenter les méthodes virtuelles pures qui sont déclarées dans l'interface.

Notre controller communique avec le view et le model grâce à un « std::vector<AData*> ».

C'est-à-dire que c'est ce vector qui va circuler entre la partie jeu et la partie graphique grâce au controller qui est l'arcade.

Votre extension, qu'elle soit graphique ou un jeu, doit être compilée en tant que librairie dynamique .so compatible avec le système d'exploitation qui à compilé le reste de l'arcade.

Créer un jeu

Votre jeu devra donc en stocker un afin de pousser dedans les « datas » que vous voulez envoyer à votre view, à savoir le GUI actuellement utilisé.

Votre classe de jeu devra posséder en tout 16 méthodes ayant toutes un but bien précis afin que le controller puisse dialoguer avec votre jeu :

- **void InitGame();**
    - Cette méthode sera directement appelée après que le controller ait récupéré une instance de votre jeu et lui ai attribué un ScoreManager.
    - Vous devez ici initialiser votre jeu, map, joueur, etc et pousser dans votre vector les « datas » à transmettre.
- **std::string const &getGameName() const;**
    - Cette méthode retourne le nom de votre jeu.

- **int getFrameRatePerSecond() const;**
    - Cette méthode retourne le nombre de frames maximum que votre jeu supporte.
- **std::vector <AData *> const &getData() const;**
    - Cette méthode retourne le std::vector<AData*> afin qu'il soit récupéré par le controller pour être ensuite transmis à la partie graphique.
- **std::vector <std::string> const & getSprite() const;**
    - Cette méthode a pour but d'optimiser le chargement des sprites afin d'éviter des ralentissements en jeu. Retournez simplement un vector contenant les chemins vers vos sprites.
- **std::vector <std::string> const & getMusic() const;**
    - Cette méthode a pour but d'optimiser le chargement des musiques afin d'éviter des ralentissements en jeu. Retournez simplement un vector contenant les chemins vers vos musiques.
- **std::vector <std::string> const & getSModel3D() const;**
    - Cette méthode a pour but d'optimiser le chargement des models 3D afin d'éviter des ralentissements en jeu. Retournez simplement un vector contenant les chemins vers vos models 3D.
- **void setScoreManager(ScoreManager *scoreManager);**
    - Le controller va vous attribuer un ScoreManager avant d'appeler votre méthode InitGame(), grâce à celui-ci vous pourrez stocker et récupérer les scores fait pendant vos parties.
- **void updateNewScore() const;**
    - Cette méthode doit appeler la méthode pushNewScore(int) du ScoreManager.
- **void play();**
    - Cette méthode exécute une frame de votre jeu.
- **void getMap();**
    - Cette méthode écrit sur la sortie standard votre map de jeu pour exécuter des tests unitaires.
- **void whereIAm();**
    - Cette méthode écrit sur la sortie standard la position de votre joueur pour exécuter des tests unitaires.
- **void goUp();**
    - Cette méthode est appelée si la librairie graphique a détecté que la touche flèche du haut a été pressée.

- **void goDown();**
    - Cette méthode est appelée si la librairie graphique a détecté que la touche flèche du bas a été pressée.
- **void goLeft();**
    - Cette méthode est appelée si la librairie graphique a détecté que la touche flèche de gauche a été pressée.
- **void goRight();**
    - Cette méthode est appelée si la librairie graphique a détecté que la touche flèche de droite a été pressée.
- **void goForward();**
    - Cette méthode est appelée si la librairie graphique a détecté que la touche entrée a été pressée.
- **void shoot();**
    - Cette méthode est appelée si la librairie graphique a détecté que la touche espace a été pressée.

Créer une GUI

Comme pour créer un jeu, créer une GUI nécessite de respecter certaines méthodes et de donner les bonnes informations au controller.

Ainsi, la classe mère de votre wrapper doit implémenter au moins nos 21 méthodes données par l'interface :

- **void InitLib();**
    - Cette méthode vous permet d'initialiser votre librairie graphique.
- **std::string const & getLibName() const;**
    - Cette méthode doit retourner le nom de la librairie graphique que vous utilisez.
- **void giveData(std::vector <AData *> const &data);**
    - Cette méthode reçoit du controller le vector de AData*, vous devez le parcourir afin d'afficher tout ce qu'il contient.
- **void giveSprite(std::vector <std::string> const &spriteList);**
    - Cette méthode reçoit du controller le vector contenant les chemins des sprites à charger. L'intérêt est de les charger s'ils ne le sont pas déjà.

- **void giveMusic(std::vector <std::string> const &spriteList);**
    - Cette méthode reçoit du controller le vector contenant les chemins des musiques à charger. L'intérêt est de les charger si elles ne le sont pas déjà.
- **void giveModel3D(std::vector <std::string> const &spriteList);**
    - Cette méthode reçoit du controller le vector contenant les chemins des models 3D à charger. L'intérêt est de les charger s'ils ne le sont pas déjà.
- **void setBridge(IArcadeBridge * bridge);**
    - Cette méthode permet au controller de vous donner un « pont » afin d'appeler les méthodes de celui-ci (exemple : n'importe quelle touche).
- **void handleData(AData const & data);**
    - Cette méthode doit appeler la bonne méthode d'affichage en fonction de la data passée en paramètre.
- **void handleSphere(AData const & data);**
    - Cette méthode doit afficher la data de type sphère passée en paramètre.
- **void handleCube(AData const & data);**
    - Cette méthode doit afficher la data de type cube passée en paramètre.
- **void handleCamera(AData const & data);**
    - Cette méthode doit afficher la data de type camera passée en paramètre.
- **void handleLight(AData const & data);**
    - Cette méthode doit afficher la data de type light passée en paramètre.
- **void handleMusic(AData const & data);**
    - Cette méthode doit afficher la data de type music passée en paramètre.
- **void handleText(AData const & data);**
    - Cette méthode doit afficher la data de type text passée en paramètre.
- **void toggleRunning() const;**
    - Cette méthode n'est pas encore implémentée.
- **void prevGraph() const;**
    - Cette méthode ordonne au controller de passer sur la librairie graphique précédente.
- **void nextGraph() const;**
    - Cette méthode ordonne au controller de passer sur la librairie graphique suivante.
- **void prevGame() const;**
    - Cette méthode ordonne au controller de passer sur la librairie de jeu précédente.
- **void nextGame() const;**
    - Cette méthode ordonne au controller de passer sur la librairie de jeu suivante.
- **void goUp() const;**
    - Cette méthode signale au controller que la touche flèche du haut a été pressée.
- **void goDown() const;**
    - Cette méthode signale au controller que la touche flèche du bas a été pressée.
- **void goLeft() const;**

    - Cette méthode signale au controller que la touche flèche de gauche a été pressée.
- **void goRight() const;**
    - Cette méthode signale au controller que la touche flèche de droite a été pressée.
- **void goForward() const;**
    - Cette méthode signale au controller que la touche entrée a été pressée.
- **void shoot() const;**
    - Cette méthode signale au controller que la touche espace a été pressée.
- **void pressEchap() const;**
    - Cette méthode signale au controller que la touche echap a été pressée.
- **void pressEight() const;**
    - Cette méthode signale au controller que la touche 8 a été pressée.
- **void pressNine() const;**
    - Cette méthode signale au controller que la touche 9 a été pressée.