Documentation du Projet Arcade

Implémentation de Nouvelles Bibliothèques Graphiques et de Jeux

Introduction

Le projet Arcade est conçu pour être extensible, permettant l'ajout de nouvelles bibliothèques graphiques et de jeux compatibles. Cette documentation vise à fournir des instructions détaillées sur la façon d'implémenter de telles extensions.

1/LE CORE:

Le "Core" est le cœur de l'application Arcade. Il est responsable de la gestion générale du jeu, de la sélection des bibliothèques graphiques et des jeux, ainsi que de l'interaction entre eux.

Concrètement, le Core effectue les actions suivantes :

```cpp
void Arcade::Core::loadGameLibrary(const std::string& libraryPath) {
    if (game_handle) {
        dlclose(game_handle);
        game_handle = nullptr;
    }
    game_handle = dlopen(libraryPath.c_str(), RTLD_LAZY);
    if (!game_handle) {
        throw Exception(std::string("Failed to load library: ") + dlerror());
    }
    dlerror();
    create_game = reinterpret_cast<decltype(create_game)>(dlsym(game_handle, "create_game"));
    if (!create_game) {
        throw Exception(std::string("Failed to load symbol: ") + dlerror());
    }

    getType = reinterpret_cast<decltype(getType)>(dlsym(game_handle, "getType"));
    if (!getType) {
        throw Exception(std::string("Failed to load symbol for getType: ") + dlerror());
    }
}
```

Chargement des bibliothèques dynamiques :

-Le Core utilise les fonctions dlopen, dlclose, dlsym et dlerror pour charger dynamiquement les bibliothèques graphiques et de jeux à l'exécution.

-Il s'assure que les bibliothèques chargées sont situées dans le dossier lib à la racine du répertoire du projet.

-Il garantit que les bibliothèques sont traitées de manière uniforme et générique, sans distinction entre elles.

```cpp
std::vector<std::string> Arcade::Core::goto_menu(std::unique_ptr<IGame>& game, std::vector<std::string> map) {
    game.reset();
    loadGameLibrary("./lib/arcade_menu.so");
    game = create_game();
    map = game->init();
    return map;
}
```

Initialisation du jeu :

-Le Core crée une instance du jeu à jouer en utilisant la bibliothèque de jeu chargée dynamiquement.

-Il appelle la fonction d'initialisation du jeu (init) pour obtenir la carte initiale du jeu.

Initialisation de la bibliothèque graphique :

-Le Core crée une instance de la bibliothèque graphique sélectionnée en utilisant la bibliothèque graphique chargée dynamiquement.

-Il appelle la fonction d'initialisation de la bibliothèque graphique (init) pour configurer l'affichage.

Gestion des événements et mise à jour de l'affichage :

-Le Core récupère les entrées utilisateur à l'aide de la fonction get_key de la bibliothèque graphique.

-Il transmet ces entrées au jeu en appelant la fonction receive_info.

-Il met à jour l'état du jeu en appelant la fonction get_map_n_edit.
-Il récupère le score actuel et le meilleur score à afficher en appelant respectivement les fonctions get_score et get_highscore.

-Il affiche la carte de jeu et les scores en appelant la fonction display de la bibliothèque graphique.

Gestion des transitions entre les jeux :

-Le Core permet de passer d'un jeu à un autre en appelant la fonction change_game_handler.

-Il charge dynamiquement la bibliothèque du nouveau jeu sélectionné.

-Il réinitialise le jeu avec le nouveau jeu chargé.

```
std::unique_ptr<IGraphic> Arcade::Core::change_graphique_handler(std::unique_ptr <IGraphic> &graphique)
{
    graphique->destroy_lib();
    graphique.reset();
    loadGraphicLibrary(libs_path_list[currentLibIndex]);
    graphique = create_graphique();
    graphique->init();
    currentLibIndex = (currentLibIndex + 1) % libs_path_list.size();
    return std::move(graphique);
}
```

Gestion des transitions entre les bibliothèques graphiques :

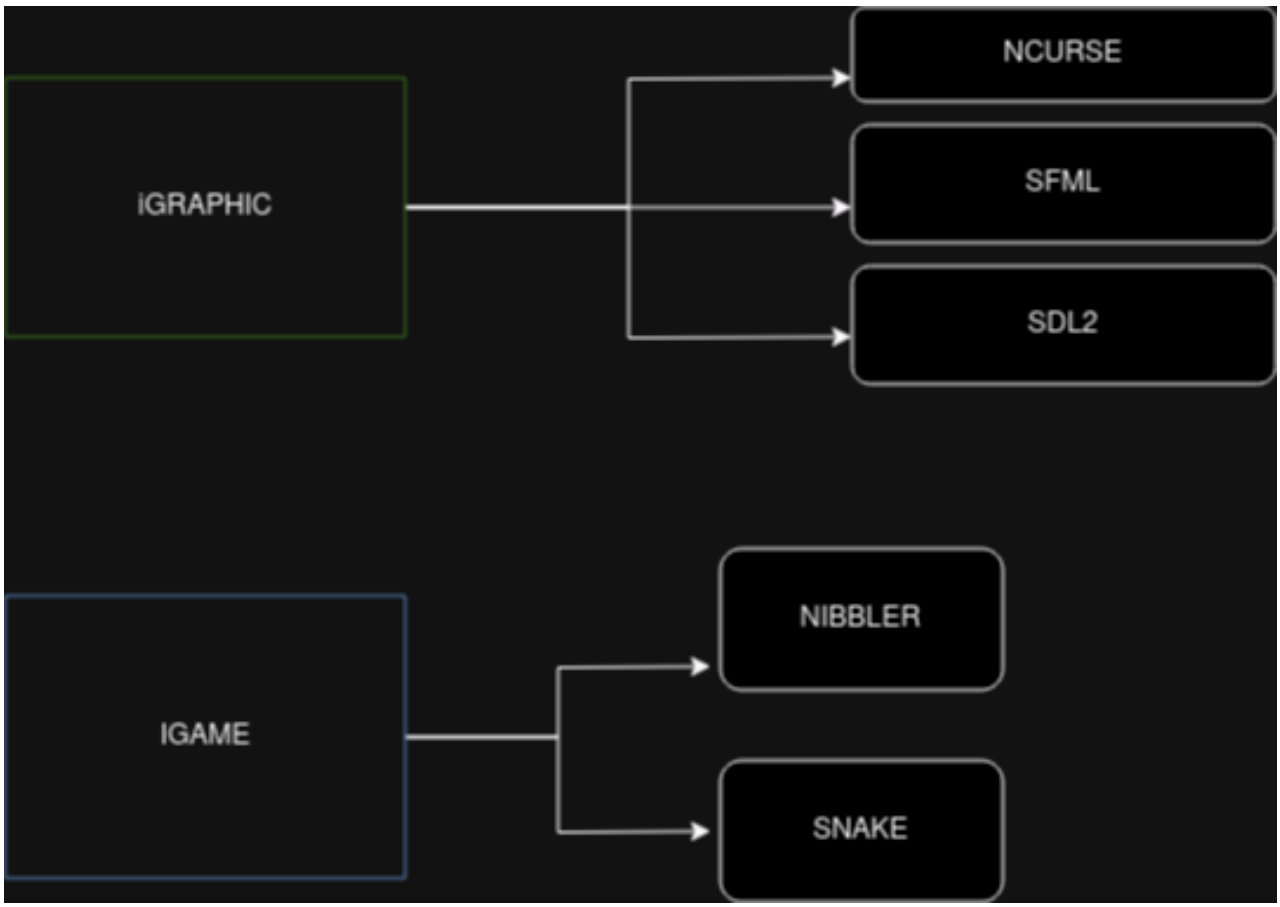
-Le Core permet de passer d'une bibliothèque graphique à une autre en appelant la fonction change_graphique_handler.

-Il charge dynamiquement la bibliothèque graphique sélectionnée.

-Il réinitialise l'affichage avec la nouvelle bibliothèque graphique chargée.

En résumé, le Core orchestre toute l'interaction entre les bibliothèques graphiques et les jeux, garantissant ainsi une expérience de jeu fluide et unifiée pour l'utilisateur.

Diagrammes des classes:



Interface Graphique - Classe IGraphic

La classe IGraphic est une interface virtuelle définissant les fonctionnalités nécessaires pour la gestion de la partie graphique dans le projet Arcade.

Chaque bibliothèque graphique utilise les mêmes fonctions pour permettre le lancement des différents jeux.

Elle compile un fichier .so dans le dossier lib à la racine du projet.

Voici une explication détaillée de chaque fonction membre :

```cpp
class IGraphic {
public:
    IGraphic() = default;
    virtual ~IGraphic() = default;
    virtual void my_refresh() = 0;
    virtual void init() = 0;
    virtual int get_key() = 0;
    virtual bool display(std::vector<std::string> map, int score, int highscore) = 0;
    virtual void destroy_lib() = 0;

    virtual void assign_texture(std::vector<std::pair<char, std::vector<std::string>>> tab_texture) = 0;

    class Exception : public std::exception {
    public:
        Exception(std::string message) : _message(std::move(message)) {}
        const char *what() const noexcept override {
            return _message.c_str();
        }
    private:
        std::string _message;
    };

private:
    std::map<std::string, std::string> list_texture;
    std::vector<std::string> map;
};
```

1/my_refresh()

- *Description*: Cette fonction est utilisée pour rafraîchir l'affichage graphique.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Aucun.

2/init()

- *Description*: Initialise la bibliothèque graphique.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Aucun.

3/get_key()

- *Description*: Récupère la touche saisie par l'utilisateur.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Un entier représentant la touche saisie.

4/display(std::vector<std::string> map, int score, int highscore)

- *Description*: Affiche la carte de jeu ainsi que les scores.
- *Type*: void.
- *Paramètres*:
    - map: Vecteur de chaînes de caractères représentant la carte de jeu.
    - score: Score actuel du joueur.
    - highscore: Meilleur score atteint.
- *Retour*: Un booléen indiquant si l'affichage a été effectué avec succès.

5/destroy_lib()

- *Description*: Libère les ressources allouées par la bibliothèque graphique.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Aucun.

6/assign_texture(std::vector<std::pair<char, std::vector<std::string>>> tab_texture)

- *Description*: Associe des textures aux caractères spécifiés.
- *Type*: void.
- *Paramètres*:
    - tab_texture: Vecteur de paires caractère-texture.
- *Retour*: Aucun.

Cette interface fournit une base solide pour l'implémentation de bibliothèques graphiques dans le cadre du projet Arcade. Chaque fonction membre est essentielle pour assurer le bon fonctionnement et l'intégration des différentes bibliothèques graphiques.

Interface de Jeu - Classe IGame

La classe IGame est une interface virtuelle définissant les fonctionnalités nécessaires pour la gestion des jeux dans le projet Arcade.

Chaque game utilise les mêmes fonctions pour permettre leur lancement.

Elle compile un fichier .so dans le dossier lib à la racine du projet.

Voici une explication détaillée de chaque fonction membre :

```
class IGame {
public :
    IGame() = default;
    virtual ~IGame() = default;
    virtual std::vector<std::string> init() = 0;
    virtual std::vector<std::string> get_map_n_edit(std::vector<std::string> map) = 0;
    virtual void receive_info(int key) = 0;
    virtual int get_score() = 0;
    virtual int get_highscore() = 0;

    virtual std::vector<std::pair<char, std::vector<std::string>>> get_texture() = 0;
};
```

1/init()

- *Description*: Initialise le jeu et retourne la carte initiale.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Un vecteur de chaînes de caractères représentant la carte initiale du jeu.

2/get_map_n_edit(std::vector<std::string> map)

- *Description*: Récupère la carte de jeu actuelle et permet l'édition de la carte.
- *Type*: void.
- *Paramètres*:
    - map: Vecteur de chaînes de caractères représentant la carte de jeu actuelle.
- *Retour*: Un vecteur de chaînes de caractères représentant la carte de jeu modifiée.

3/receive_info(int key)

- *Description*: Reçoit les informations des touches saisies par l'utilisateur.
- *Type*: void.
- *Paramètres*:
    - key: Entier représentant la touche saisie par l'utilisateur.
- *Retour*: Aucun.

4/get_score()

- *Description*: Récupère le score actuel du joueur.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Un entier représentant le score actuel du joueur.

5/get_highscore()

- *Description*: Récupère le meilleur score atteint.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Un entier représentant le meilleur score atteint.

6/get_texture()

- *Description*: Récupère les textures utilisées dans le jeu.
- *Type*: void.
- *Paramètres*: Aucun.
- *Retour*: Un vecteur de paires caractère-texture.

Cette interface fournit une structure de base pour la création de jeux compatibles avec le projet Arcade. Chaque fonction membre est cruciale pour la gestion des différentes fonctionnalités d'un jeu et garantit une intégration harmonieuse dans le projet global.

Intégration:

L'intégration d'un nouveau jeu dans le projet Arcade se fait de manière fluide et bien structurée.

Tout d'abord, le jeu est compilé en tant que bibliothèque partagée (.so) et placé dans le répertoire `lib` du projet.

Ensuite, le Core, le cœur de l'application Arcade, charge dynamiquement la bibliothèque du jeu lorsqu'il est sélectionné par l'utilisateur.

Une fois chargé, le Core initialise le jeu en appelant sa fonction d'initialisation, lui permettant ainsi de fournir la carte de jeu initiale.

Par la suite, le Core gère l'interaction entre l'utilisateur et le jeu en récupérant les entrées utilisateur, en transmettant ces entrées au jeu et en mettant à jour l'état du jeu en conséquence.

Enfin, le Core se charge d'afficher le jeu à l'utilisateur, en récupérant les scores actuels et en affichant la carte de jeu ainsi que les scores.

Cette approche bien définie et cohérente permet l'intégration harmonieuse de nouveaux jeux dans le projet Arcade, offrant ainsi aux utilisateurs une expérience de jeu variée et immersive.