Arcade

Documentation complète du projet Arcade, créé par les personnes ci-dessous :

- Gianni Henriques
- Elouan Rigomont
- Augustin Piffeteau
- Sacha Polerowicz
- Isaac Lévy-piquet
- Jules Weishaus
- Hugo Denis
- Robin Caboche
- Nicolas Nguyen

Ce document a pour but d'expliquer dans les grandes lignes comment implémenter vos propres librairies graphiques et de jeux à notre projet. 3 librairies graphiques de base ont été implémentées par nos trois groupes qui sont la SFML, la SDL2 et la Ncurses mais rien ne vous empêche de les refaire à votre manière.

Documentation de l'interface *IGraphicalModule*

Introduction

L'interface *IGraphicalModule* définit les méthodes nécessaires pour l'interaction avec un module graphique dans le cadre d'un projet de jeu.

Architecture globale

Ce projet implique l'utilisation d'un module graphique pour afficher des éléments tels qu'une fenêtre, une carte, et pour gérer les événements clavier.

Description des classes et des modules

IGraphicalModule

Cette interface définit les méthodes nécessaires pour créer, afficher et détruire une fenêtre, ainsi que pour gérer les événements clavier et afficher une carte sur la fenêtre.

Documentation du code

Le code est documenté en utilisant des commentaires Doxygen pour chaque méthode de l'interface *IGraphicalModule*. Voici la documentation de chaque méthode :

```
/**
 * @brief Create the window
 * @param name Title of the window
 * @param size Size of the window (default: 1920x1080)
*/
void createWindow(const std::string& name, const std::vector<int>& size);
```

```
/**
 * @brief Display the window
*/
void displayWindow();
```

```
/**
 * @brief Destroy the window
*/
void destroyWindow();


/**
 * @brief Return true if the window is open, false otherwise
 * @return bool
*/
bool isWindowOpen();


/**
 * @brief Parse keyboard events
 * @return Input The keyboard input parsed
*/
Input parseKeyboard();


/**
 * @brief Display the map passed as parameter onto the window
 * @param map Map to show
*/
void showMap(const std::vector<std::vector<Tiles>>& map);


/**
 * @brief Return the library type
 * @return std::string The type of library
*/
std::string getLibraryType() const;


/**
 * @brief Initialize the assets
 * @param entities Entities to initialize
*/
void initAssets(const std::vector<std::shared_ptr<AEntities>>& entities);
```

Configuration et installation

Pour utiliser cette interface, il est nécessaire de l'implémenter dans un module graphique spécifique à votre projet. Assurez-vous d'installer toutes les dépendances nécessaires pour ce module.

Documentation de l'interface: *IGameModule*

Introduction

L'interface *IGameModule* définit les méthodes nécessaires pour l'interaction avec un module de jeu dans le cadre d'un projet graphique.

Architecture globale

Le module de jeu sera conçu pour offrir une expérience immersive aux utilisateurs, en intégrant des mécanismes de jeu interactifs et en affichant des éléments tels que des entités, des décors et des interfaces utilisateur. Il sera également chargé de gérer les règles du jeu: les collisions ou encore les interactions entre les différents éléments.

Description des classes et des modules

IGameModule

Cette interface définit les méthodes nécessaires pour gérer le score, le statut du jeu, récupérer la map, gérer le pseudo, de récupérer les inputs entrer par l'utilisateur et enfin initialiser toutes les entités qui permettront de créer la map

Documentation du code

Le code est documenté en utilisant des commentaires Doxygen pour chaque méthode de l'interface *IGameModule*. Voici la documentation de chaque méthode :

```
/**
* @brief Set the score of the game
* @param score The score to set
* @return void
* @class IGameModule
*/
void setScore(int score);
```

```
/**
* @brief Get the score of the game
* @return int
* @class IGameModule
*/
int getScore() const;


/**
* @brief Set the highscore of the game
* @param score The score to set
* @param playerName The name of the player
* @return void
* @class IGameModule
*/
void setHighScore(int score, std::string = "");


/**
* @brief Get the highscore of the game
* @return std::map<std::string, int>
* @class IGameModule
*/
std::map<std::string, int> getHighScore() const;


/**
* @brief Set the game status
* @param status The status to set
* @return void
* @class IGameModule
*/
void setGameStatus(GameStatus status);


/**
* @brief Get the game status
* @return GameStatus
* @class IGameModule
*/
GameStatus getGameStatus() const;
```

```
/**
* @brief Get the map of the game
* @return std::vector<std::vector<Tiles>>
* @class IGameModule
*/
std::vector<std::vector<Tiles>> getMap() const;


/**
* @brief Get the player name
* @return std::string
* @class IGameModule
*/
std::string getPlayerName() const;


/**
* @brief Set the player name
* @param name The name to set
* @return void
* @class IGameModule
*/
void setPlayerName(std::string name);


/**
* @brief Get the game name
* @return std::string
* @class IGameModule
*/
std::string getGameName() const;


/**
* @brief Set the game name
* @param name The name to set
* @return void
* @class IGameModule
*/
void setGameName(std::string name);


/**
* @brief Initialize all the entities of the game
* @return std::vector<std::shared_ptr<AEntities>>
* @class IGameModule
*/
std::vector<std::shared_ptr<AEntities>> initAllEntities() const;
```

```
/**
* @brief Catch the input of the player
* @param key The input to catch
* @return void
* @class IGameModule
*/
void catchInput(Input key = NONE);
```

Configuration et installation

Pour utiliser cette interface, il est nécessaire de l'implémenter dans un module de jeu spécifique à votre projet. Assurez-vous d'installer toutes les dépendances nécessaires pour ce module.

Relier les librairies au core

Le core est la partie centrale de notre projet, c'est l'exécutable qui permet de lancer les jeux et les librairies graphiques donc il est très important de comprendre comment s'y connecter.

Accompagné du core nous avons des classes abstraites très importantes nommées respectivement AGraphicalModule pour les librairies graphiques et AGameModule pour les librairies de jeux qui elles-mêmes héritent de deux interface nommées IGraphicalModule et IGameModule. Toutes les classes que vous allez créer par la suite doivent descendre de l'une de ces deux classes abstraites pour pouvoir fonctionner correctement avec le core.

Vous aurez également besoin d'implémenter une fonction appelée « extern "C" » à côté de vos classes.

Classes héritant de AGraphicalModule

La fonction extern "C" doit être déclarée dans le header de votre librairie graphique, doit être nommée "createLib()" et doit return un std::shared_ptr<AGraphicalModule> comme le montre l'exemple suivant:

```
extern "C" std::shared_ptr<AGraphicalModule> createLib() {
    return std::make_shared<YourGraphicalLib>();
}
```

Classes héritant de AGameModule

La fonction extern "C" doit être déclarée dans le header de votre librairie de jeu, doit être nommée "createGame()" et doit return un std::shared_ptr<AGameModule> comme le montre l'exemple suivant:

```
extern "C" std::shared_ptr<AGameModule> createLib() {
    return std::make_shared<YourGameLib>();
}
```

Le core se chargera ensuite de créer votre jeu/votre window à partir du pointeur que vous lui avez envoyé.

Classes et enums tierces

Tiles

La classe Tiles représente chaque pixel d'une map. C'est là où l'on va définir toutes les entités présentes à un endroit, comment doit-elle être affichée , leur taille etc...

```cpp
class Tiles {
    public:
        Tiles(std::vector<std::shared_ptr<AEntities>> entities, int size = 1);

        std::vector<std::shared_ptr<AEntities>> getEntities() const;
        void setEntities(std::vector<std::shared_ptr<AEntities>> entities);
        int getSize() const;
        void setSize(int size);

    private:
        std::vector<std::shared_ptr<AEntities>> _entities;
        int _size;
        std::pair<int, int> _pos = {0, 0};
};
```

AEntities

AEntities est une classe abstraite qui permet de créer toutes les entités dont vous aurez besoin pour votre jeu. Elles seront stockées dans la map, précisément dans chaque Tiles où elles doivent être affichées.

```cpp
class AEntities : public IEntities {
    public:
        AEntities(double speed = 1, std::pair<int, int> pos = {0, 0}, std::string texturePath = "", ASCII ascii = ASCII(' ', Color(255, 255, 255)), std::string name = "entity");

        double getSpeed() const;
        void setSpeed(double speed);
        std::pair<int, int> getPos() const;
        void setPos(std::pair<int, int> pos);
        std::pair<std::string, ASCII> imageToDisplay() const;
        std::string getName() const;
        void setName(std::string name);
        virtual EntityType getType() const = 0;

    protected:
        double _speed;
        std::pair<int, int> _pos;
        std::string _texturePath;
        ASCII _ascii;
        std::string _name;
};
```

GameStatus

Cette enum permet de spécifier l'état actuel du jeu lancé.

```
enum GameStatus {
    OVER,
    WIN,
    PAUSE,
    RESTART,
    RUNNING
};
```

Input

Cette enum permet aux librairies graphiques de gérer les events liés au clavier mais aussi au core de récupérer ces events et de les associer à certaines actions ou de les envoyer vers le jeu en cours.

```
enum Input {
    UP,
    DOWN,
    LEFT,
    RIGHT,
    ESC,
    ENTER,
    SPACE,
    MENU,
    CHANGE_LIB,
    CHANGE_GAME,
    RELOAD,
    NONE,
    DELETE,
```

Les inputs affichés ci-dessus sont les plus importants mais il est tout à fait possible de gérer le reste du clavier.