Arcade

Arcade est un projet réalisé en 2ème année à Epitech, dans le cadre du module Object Oriented Programming.

L'objectif principal du projet est de créer une machine d'arcade virtuelle.

Le projet nous incite à créer un programme très modulable, le noyau est donc séparé des jeux et des bibliothèques graphiques. Les trois doivent pouvoir être compilés séparément et être indépendents les uns des autres.

Créer un jeux / bibliothèque graphique

Pour ajouter un jeux ou une bibliothèque graphique (bibliothèques dynamiques) à l'Arcade il faut tout d'abord une fonction qui crée le point d'entré s'appelant `create`. Elle doit renvoyer un `shared_ptr` vers une des deux interfaces et doit être liée comme une fonction C.

Voir dans Nibbler.hpp ou dans SDL2.hpp pour des exemples de fonction d'entrée.

Les namespaces: - Pour les jeux: `Games` - Pour les bibliothèques graphiques: `Libs`.

Pour un Jeux

Il suffit d'ajouter une bibliotèque partagée quelque part, préciser son **nom** et path dans Games.cpp (dans le `std::vector`) et créer un dossier avec le **nom** du jeux et y mettre un fichier `scores.txt`.

Exemple: Pour ajouter pacman, il faudrait rajouter dans le `std::vector`.

```
{"pacman", "lib/arcade_pacman.so}
```

et créer `assets/pacman/scores.txt`.

Pour une bibliotèque graphique

Il suffit d'ajouter une bibliotèque partagée quelque part, préciser son **nom** et path dans Graphics.cpp (dans le `std::vector`).

Exemple: Pour ajouter SDL2, il faudrait rajouter dans le `std::vector`.

```
{"sdl2", "lib/arcade_sdl2.so}
```

Interfaces

Les interfaces utilisent toutes des structures de données ci-dessous, également définies dans Interoperability.hpp

Interface pour les jeux

```
namespace Games {
    class IGameModule {

        public:
            virtual ~IGameModule() = default;
            virtual std::unordered_map<std::string, Arcade::Rect_t> getShapes() = 0;
            virtual std::unordered_map<std::string, Arcade::Sprites_t> getSprites() = 0;
            virtual std::unordered_map<std::string, Arcade::Text_t> getTexts() = 0;
            virtual std::pair<int, int> getSizeWindow() = 0;
            virtual std::pair<int, int> getSizePixel() = 0;
            virtual void update() = 0;
            virtual void handleEvents(std::vector<Arcade::KeyEvent_t> event) = 0;
    };
}
```

Les fonctions dans IGameModule

`std::unordered_map<std::string, Arcade::Rect_t> getShapes();`

`getShapes` renvoie une map de `std::string` vers des `Arcade::Rect_t`. Les `std::string` sont en général des noms de références pour une forme. Les `Arcade::Rect_t` représentent les propriétés de la formes (couleurs, position, etc ...). La valeur de retour de cette fonction est utilisée pour afficher des formes par les biobliotèques graphiques.

`std::unordered_map<std::string, Arcade::Sprites_t> getSprite();`

`getSprite` renvoie une map de `std::string` vers des `Arcade::Sprites_t`. Les `std::string` sont en général des paths d'images ou des noms de références pour un sprite. Les `Arcade::Sprites_t` représentent les propriétés du sprite (couleurs, position, etc ...). La valeur de retour de cette fonction est utilisée pour afficher des sprites par les biobliotèques graphiques.

`std::unordered_map<std::string, Arcade::Text_t> getTexts();`

`getSprite` renvoie une map de `std::string` vers des `Arcade::Text_t`. Les `std::string` sont en général des paths de police de caractères ou des noms de références pour un texte. Les `Arcade::Text_t` représentent les propriétés du texte (couleurs, position, etc ...). La valeur de retour de cette fonction est utilisée pour afficher des textes par les biobliotèques graphiques.

`std::pair<int, int> getSizeWindow();`

`getSizeWindow` renvoie une paire de `int`, le premier représente la largeur et le deuxième la hauteur. Grace à cette fonction chaque jeux peut avoir sa propre taille d'écran. La valeur de retour de cette fonction est utilisée pour définir la taille de la fenêtre par les biobliotèques graphiques.

```
std::pair<int, int> getSizePixel();
```

`getSizePixel` renvoie une paire de `int`, le premier représente la largeur et le deuxième la hauteur. Cette fonction est surtout utilisée par la `ncurses` La valeur de retour de cette fonction est utilisée pour dire combien de pixels une case en `ncurses` représente, ainsi la biobliotèque graphique peut régler la taille du contenu à afficher.

```
void update();
```

`update` appelée par le noyau du programme, elle met à jour toute les données des jeux comme par exemple les shapes, shapes ou textes à afficher qui seront renvoyés respectivement par les fonctions `getShapes`, `getSprite`, et `getTexts`. Elle est appelée dans la boucle principale du noyau avant la fonction d'affichage des biobliotèques graphiques, c'est une des fonctions principales des jeux.

```
void handleEvents(std::vector<Arcade::KeyEvent_t> event);
```

`handleEvents` reçoit une liste d'évènements par les biobliotèques graphiques. Elle permet au jeux d'effectuer une ou plusieurs actions en fonction des touches appuyées par l'utilisateur, de la position de la souris, etc…

Interface pour les bibliotèques graphiques

```
namespace Libs {
    class IDisplayModule {

        public:
            virtual ~IDisplayModule() = default;
            virtual void init(std::string name = "Unknown", int w = 800, int h = 600) = 0;
            virtual bool isOpen() const = 0;
            virtual void close() = 0;
            virtual void my_clear() const = 0;
            virtual void setBackgroundColor(int, int, int) = 0;
            virtual void display() const = 0;
            virtual void draw(std::unordered_map<std::string, Arcade::Rect_t> shapes,
                std::unordered_map<std::string, Arcade::Sprites_t> sprites,
                std::unordered_map<std::string, Arcade::Text_t> texts) = 0;
            virtual std::vector<Arcade::KeyEvent_t> getEvents() = 0;
            virtual void setSizePixel(std::pair<int, int> size) = 0;
            virtual void setSizeWindows(std::pair<int, int> size) = 0;
    };
}
```

Les fonctions dans IDisplayModule

```
void init(std::string name = "Unknown", int w = 800, int h = 600);
```

`init` est appelée pour ouvrir une fenêtre avec le titre `name`, une largeur `w` et une hauteur `h`. Elle est appelée à nouveau lors des changements de bibliotèque graphique.

```
bool isOpen() const;
```

La valeur de retour de la fonction `isOpen` est utilisée pour maintenir la fenêtre ouverte, si elle renvoie `false` la fenêtre se ferme et le programme s'arrête.

```
void close();
```

`close` permet de fermer une fenêtre.

```
void my_clear() const;
```

`my_clear` permet de réinitialiser l'espace mémoire utilisé pour l'affichage. Ainsi les nouveaux éléments affichés ne sont pas surperposés avec les anciens.

```
void setBackgroundColor(int, int, int);
```

`setBackgroundColor` est simplement utilisée pour appliquer une couleur à l'arrière plan. Les trois `int` passés en arguments représentent respectivement les canaux rouge, vert et bleu.

```
void display() const;
```

`display` met à jour l'écran avec le rendu effectué.

```
void draw(
    std::unordered_map<std::string, Arcade::Rect_t> shapes,
    std::unordered_map<std::string, Arcade::Sprites_t> sprites,
    std::unordered_map<std::string, Arcade::Text_t> texts
); const;
```

`draw` rempli l'espace mémoire utilisé pour l'affichage, les arguments `shapes`, `sprites` et `texts` sont les éléments à afficher reçues par les fonctions des jeux `getShapes`, `getSprite`, `getTexts`. Ils sont ajoutés pour l'affichage dans la boucle du noyau.

```
std::vector<Arcade::KeyEvent_t> getEvents();
```

`getEvents` renvoie une liste d'évènements utilisateurs qui va être passer a la fonction des jeux `handleEvents`.

```
void setSizePixel(std::pair<int, int> size);
```

`setSizePixel` défini la taille d'une case en `ncurses` en pixel.

```
void setSizeWindows(std::pair<int, int> size);
```

`setSizeWindows` défini la taille de la fenêtre en pixel.