Arcade

Arcade is a project realized in 2nd year at Epitech, in the framework of the Object Oriented Programming module.

The main objective of the project is to create a virtual arcade machine.

The project encourages us to create a very modudable program, the kernel is thus separated from the games and the graphic libraries. The three must be able to be compiled separately and be independent of each other.

Create a game / graphic library

To add a game or a graphic library (dynamic libraries) to the Arcade you first need a function that creates the entry point called `create`. It must return a `shared_ptr` to one of the two interfaces and must be bound as a C function.

See Nibbler.hpp or SDL2.hpp for examples of input functions.

Namespaces: - For games: `Games` - For graphics libraries: `Libs`.

For a Games

Just add a shared library somewhere, specify its **name** and path in Games.cpp (in the `std::vector`) and create a folder with the **name** of the game and put a `scores.txt` file there.

Example: To add pacman, you would have to add in the `std::vector`.

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

and create `assets/pacman/scores.txt`.

For a graphical library

Just add a shared library somewhere, specify its **name** and path in Graphics.cpp (in the `std::vector`).

Example: To add SDL2, you would have to add in the `std::vector`.

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

Translated with www.DeepL.com/Translator (free version)

Interfaces

The interfaces all use the following data structures, also defined in Interoperability.hpp

Interface for games

```
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;
    };
}
```

The functions in IGameModule

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

`getShapes` returns a map from `std::string` to `Arcade::Rect_t`. The `std::string` are usually reference names for a shape. The `Arcade::Rect_t` represent properties of the shape (colors, position, etc ...). The return value of this function is used to display shapes by graphical bioblicks.

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

`getSprite` returns a map from `std::string` to `Arcade::Sprites_t`. The `std::string` are usually image paths or reference names for a sprite. The `Arcade::Sprites_t` represent the properties of the sprite (colors, position, etc ...). The return value of this function is used to display sprites by the graphical bioblicks.

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

`getSprite` returns a map from `std::string` to `Arcade::Text_t`. The `std::string` are usually font paths or reference names for a text. The `Arcade::Text_t` represent the properties of the text (colors, position, etc ...). The return value of this function is used to display texts by the graphical bioblicks.

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

getSizeWindow returns a pair of `int`, the first representing the width and the second the height. With this function each game can have its own screen size. The return value of this function is used to set the window size by the graphical bioblicks.

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

`getSizePixel` returns a pair of `int`, the first representing the width and the second the height. This function is mostly used by the `ncurses`. The return

value of this function is used to tell how many pixels an `ncurses` box represents, so the graphics library can set the size of the content to display.

```
void update();
```

called by the program core, it updates all the data of the sets such as shapes, shapes or texts to be displayed which will be returned respectively by the functions `getShapes`, `getSprite`, and `getTexts`. It is called in the main loop of the kernel before the function of displaying the graphical bioblicks, it is one of the main functions of the games.

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

`handleEvents` receives a list of events by the graphical bioblicks. It allows the game to perform one or more actions depending on the keys pressed by the user, the position of the mouse, etc…

Interface for graphic libraries

```
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;
    };
}
```

Functions in IDisplayModule

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

`init` is called to open a window with title `name`, width `w` and height `h`. It is called again when the graphic library changes.

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

The return value of the `isOpen` function is used to keep the window open, if it returns `false` the window closes and the program stops.

```
void close();
```

close‘ is used to close a window.

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

`my_clear` allows to reset the memory space used for the display. Thus the new elements displayed are not overlaid with the old ones.

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

setBackgroundColor`is simply used to apply a color to the background. The three`int‘ passed as arguments represent the red, green and blue channels respectively.

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

`display` updates the screen with the rendered image.

```
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` fills the memory space used for display, the `shapes`, `sprites` and `texts` arguments are the elements to be displayed received by the `getShapes`, `getSprite`, `getTexts` set functions. They are added for display in the kernel loop.

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

`getEvents` returns a list of user events that will be passed to the `handleEvents` set function.

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

`setSizePixel` sets the size of a square in `ncurses` in pixels.

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

`setSizeWindows` sets the size of the window in pixels.