Documentation Arcade

How to create a library ?

Any library must be called lib_arcade_name.so.

I – getInstance

Any library (game or graphic) must contain an “extern C getInstance function” which returns an instance of smart pointer of the current class implemented in the library.

Example:

```
extern "C" std::unique_ptr<SDL>getInstance(void)
{
    return std::make_unique<SDL>();
}
```

II – GameObject and Event

A – GameObject.hpp

Represents an entity in the game. An object contains 2 versions of itself at the same time: “terminal” version and “graphic” version.

```cpp
#ifndef GAMEOBJECT_H
#define GAMEOBJECT_H

#include <iostream>

namespace core
{
    enum GameObjectType
    {
        TEXT,
        IMAGE,
        MENU_ITEM,
        MENU_INPUT
    };

    class GameObject
    {
        public:
            GameObject(const std::string &name = "", const std::string &text = "",
                       int posX = 0, int posY = 0, int posXText = 0, int posYText = 0,
                       int originX = 0, int originY = 0, int width = 0, int height = 0,
                       GameObjectType type = TEXT);
            virtual ~GameObject();
        public:
            std::string _name;
            std::string _text;
            int _posX;
            int _posY;
            int _posXText;
            int _posYText;
            int _originX;
            int _originY;
            int _width;
            int _height;
            GameObjectType _type;
        };
} // namespace core

#endif
```

_name: the path to the sprite.

_text: contains the text to display (if the object is an image, contains the text to display for the “terminal’ version of the object).

_poX, _posY: position in pixels from the top left corner of the window. Used for graphic display.

_posXText, _posYText : position in characters from the upper left corner of the terminal. Used for terminal display.

_originX, _originY, _width, _height: Forms a rectangle that represents a sub-part of the sprite to draw. Used in graphic display.

_type: represents the type of the object. It can be a text, an image, a menuItem or a menuInput.

B – Event.hpp

List of events to use for handling events between graphics, games and the core.

```
#ifndef EVENT_H
#define EVENT_H

namespace core
{
    enum Event {
        LEFT,
        RIGHT,
        UP,
        DOWN,
        ENTER,
        SHOOT,
        PAUSE,      //go back to menu
        RESTART,    //restart game
        QUIT,       //exit
        PREV_GRAPH, //previous graphic library
        NEXT_GRAPH, //next graphic library
        PREV_GAME,  //previous game
        NEXT_GAME   //next game
    };
} // namespace core

#endif
```

II – Game Interface : IGame

Game class must inherit from IGame and library path must be ./games.

```hpp
#ifndef IGAME_H
#define IGAME_H

#include <memory>
#include <vector>
#include "GameObject.hpp"
#include "Event.hpp"

namespace core
{
    class IGame {
        public:
            virtual std::vector<GameObject> initGame(void) = 0;
            virtual std::vector<GameObject> updateGame(void) = 0;
            virtual void handleEvents(std::vector<Event> &events) = 0;
            virtual void setUserName(const std::string &name) = 0;
            virtual ~IGame(void) = default;
    };
} // namespace core

#endif
```

InitGame : initialize the game and return a gameObject vector to display.

UpdateGame : called at each frame. It represents the game loop and returns the gameObject vector to be displayed with the updated positions.

HandleEvent : called at each frame. It takes a vector of event (cf. Event.hpp) to manage and update the game accordingly.

SetUsername : call once at the start of the game. It updates the name of the player in the game (used to create scores).

~IGame : default destructor.

III – Graphic Interface : IGraph

Graphic class must inherit from IGraph and library path must be ./lib.

```cpp
#ifndef IGRAPH_H
#define IGRAPH_H

#include <memory>
#include <vector>
#include "Event.hpp"
#include "GameObject.hpp"

namespace core
{
    class IGraph {
        public:
            virtual void createWindow(int x, int y) = 0;
            virtual void deleteWindow(void) = 0;
            virtual bool initSprites(const std::vector<GameObject> &) = 0;
            virtual bool updateGraphics(const std::vector<GameObject> &) = 0;
            virtual std::vector<Event> eventLoop() = 0;
            virtual ~IGraph(void) = default;
    };
} // namespace core

#endif
```

CreateWindow : take an x and a y which represent width and height and create a window of this dimension.

DeleteWindow : delete the window previously created.

InitSprite : takes a vector of gameObject to load and store in memory.

Return false if it cannot load a sprite and true otherwise.

UpdateGraphics : called at each frame. It takes a vector of gameObject and displays them. Return false if it cannot display an object.

EventLoop : called at each frame. Returns an event vector composed of all the events passed since the last call to eventLoop.

~IGraph: default destructor.