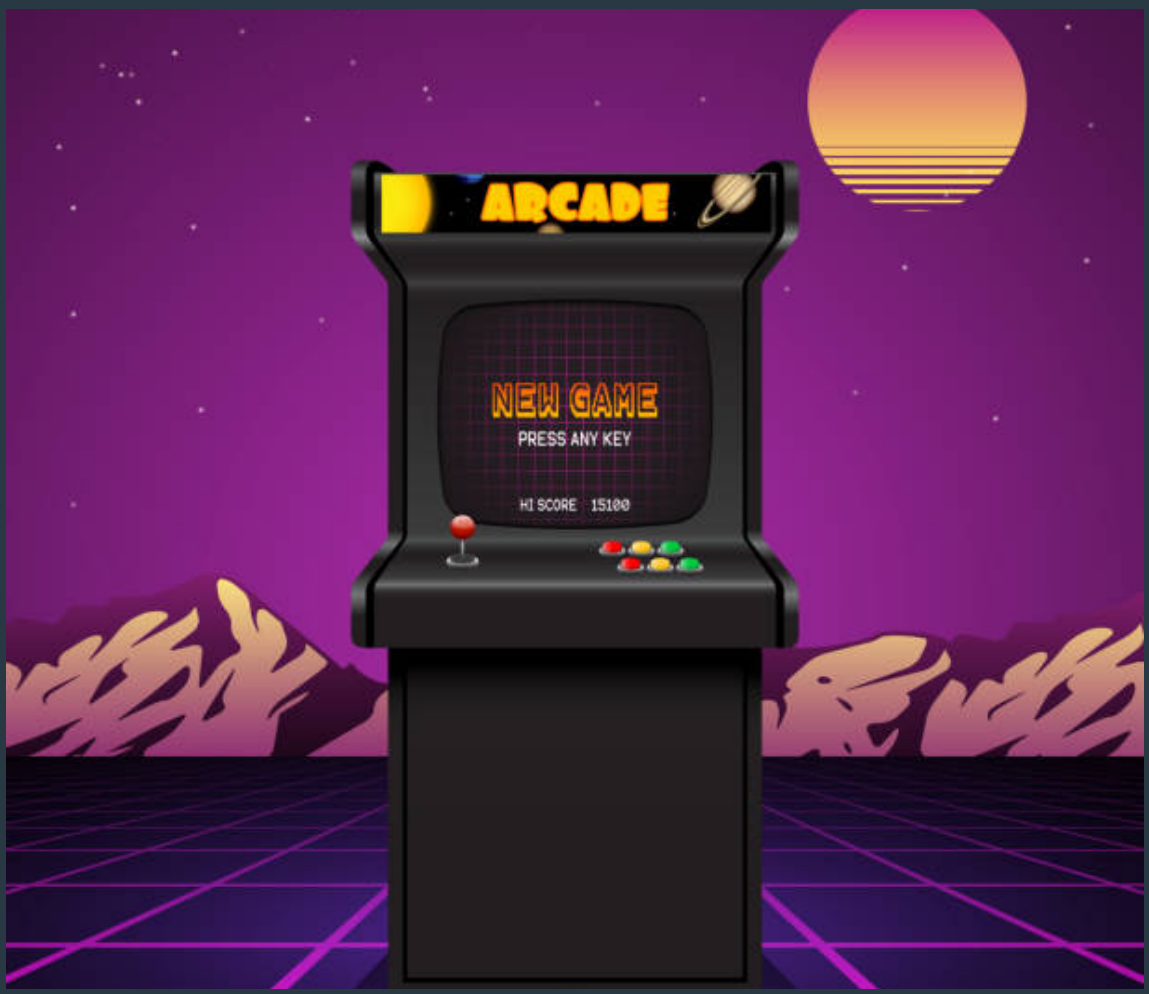
4/5/2023

# Arcade Project Documentation

Implementing New Graphics Libraries or Game Libraries Compatible with the System




Gautier BONHUR
Hugo DUBOIS

# Implementing a Game Librarie Compatible with the System

This document explains how to implement new graphics libraries or game libraries compatible with the Arcade project system. The Arcade project is designed to load shared libraries (.so files) containing game implementations that use the IWindow interface. Before diving into implementation, please make sure you understand the provided code and the overall structure of the project.

# Interface: IWindow

The IWindow interface is a vital part of the Arcade project. It represents the window used for displaying games and provides an abstraction layer for different graphics libraries. To implement a new graphics library, you need to create a new class that inherits from the IWindow interface and implement all its virtual methods.

The IWindow interface includes the following methods:

- create(std::string const &title, int framerateLimit, int width, int height)
- getEvent()
- getTitle()
- setTitle(std::string const &title)
- isOpen()
- clear()
- draw()
- display()
- close()
- drawCharacter(int x, int y, char character)

These methods are responsible for creating the window, handling events, changing the title, checking if the window is open, clearing the window, drawing characters, and closing the window. Make sure to implement these methods according to the specific graphics library you are using.

# Implementation of "New Game Library"

To implement a new game library compatible with the Arcade project, you must create a new class that inherits from the IGameModule interface. The class should have the following methods:

- init(std::vector<std::unique_ptr<Display::IWindow>> &windows)
- stop()
- updateGame()

The init method is responsible for initializing the game with a vector of unique pointers to available windows (using the IWindow interface). The stop method should close the active window, and the updateGame method should handle the game's main loop, updating the game state and rendering the game.
When implementing a new game library, ensure that it is designed as a shared library (.so file) that can be loaded by the main Arcade project.

# Example: Implementing a Game with IWindow

Let's consider a simple example of implementing a game using the IWindow interface. We will create a game called "ExampleGame" that renders a moving character on the screen.

Create a header file for the ExampleGame class, which inherits from the IGameModule interface:

```
// example_game.hpp

#pragma once

#include "IGameModule.hpp"
#include <vector>
#include <memory>

class ExampleGame : public IGameModule {
public:
    ExampleGame();
    ~ExampleGame();

    void init(std::vector<std::unique_ptr<Display::IWindow>> &windows) override;
    void stop() override;
    void updateGame() override;

private:
    std::vector<std::unique_ptr<Display::IWindow>> _windows;
    Display::IWindow *_window;
};
```

## Implement the ExampleGame class in a new source file:

```
void ExampleGame::init(std::vector<std::unique_ptr<Display::IWindow>> &windows)
{
    if (windows.empty()) {
        std::cout << "No window available" << std::endl;
        return;
    }
    _windows = std::move(windows);
    _window = _windows[0].get();
    _window->create("Example Game", 60, 800, 600); updateGame();
    return;
}

void ExampleGame::stop()
{
    _window->close();
}
```

## The game loop and the handling of events retrieved from the window. This is where we send what to display to the window:

```
void ExampleGame::updateGame()
{
    int x = 0;
    int y = 0;
    while (_window->isOpen()) {
        _window->clear();
        Display::KeyType key = _window->getEvent();
        _window->drawCharacter(x, y, 'C');
        _window->display();
    }
    stop();
}
extern "C" std::unique_ptr<IGameModule> createGame() { return
std::make_unique<ExampleGame>(); }
```

# Conclusion Game Module

This simple example demonstrates how to implement a game called "ExampleGame" that uses the IWindow interface provided by the Arcade project. The game renders a moving character on the screen, controlled by the arrow keys.

To integrate this game with the Arcade project, ensure it is compiled as a shared library (.so file) that can be loaded by the main Arcade project.

By following this guide, you can implement new  game libraries compatible with the Arcade project system. Remember to create classes that inherit from the IWindow and IGameModule interfaces, and implement their methods according to the specific requirements of your graphics library or game.

When designing a new game library, ensure it is compiled as a shared library (.so file) to be loaded by the main Arcade project.

# Display Module

The Display Module is responsible for providing an interface to create and manage windows and user input events for the Arcade project. It consists of an IWindow interface, which defines the methods required to create a window, handle events, and draw graphics.

**IWindow**

The IWindow interface is the base class for creating new window implementations. It provides a common set of methods that should be implemented by any graphics library used in the Arcade project. Here is the **IWindow interface**:

```
namespace Display {
    class IWindow {
        public:
            virtual ~IWindow() = default;
            virtual void create(
                std::string const &title,
                int framerateLimit,
                int width,
                int height
            ) = 0;
            virtual Display::KeyType getEvent() = 0;
            virtual std::string getTitle() = 0;
            virtual void setTitle(std::string const &title) = 0;
            virtual bool isOpen() = 0;
            virtual void clear() = 0;
            virtual void draw() = 0;
            virtual void display() = 0;
            virtual void close() = 0;
            virtual void drawCharacter(int x, int y, char character) = 0;
    };
};
```

# NCursesWindow

This example provides an implementation of the IWindow interface using the NCurses library. The NCursesWindow class inherits from the IWindow interface and implements its methods to create a window, handle events, and draw graphics using the NCurses library.

Here is the **NCursesWindow class**:

```
// Include header file
#include "../include/NCursesWindow.hpp"

// ... Implementation of NCursesWindow methods ...

extern "C" std::unique_ptr<Display::NCursesWindow> createWindow() {
    return std::make_unique<Display::NCursesWindow>();
}
```

To integrate this NCursesWindow implementation with the Arcade project, ensure it is compiled as a shared library (.so file) that can be loaded by the main Arcade project.

# Conclusion

By following this guide, you can implement new display libraries compatible with the Arcade project system. Remember to create classes that inherit from the IWindow interface and implement their methods according to the specific requirements of your graphics library.

When designing a new display library, ensure it is compiled as a shared library (.so file) to be loaded by the main Arcade project.