Arcade

project by Maxime Dodin, Paul Baudet & Killian Fleury.

Description of the project:

Arcade is a gaming platform: a program that lets the user choose a game to play and keeps a register of player scores. To be able to deal with the elements of your gaming plateform at run-time, your graphics libraries and your games must be implemented as dynamic libraries, loaded at runtime.
Each GUI available for the program must be used as a shared library that will be loaded and used dynamically by the main program.

pdf link: https://intra.epitech.eu/module/2020/B-OOP-400/PAR-4-1/acti-437772/project/file/B-OOP-400_arcade.pdf

Our project:

Games:

- Nibbler (Snake)
- Pac Man

Libs:

- Ncurses (Killian)
- SDL2 (Maxime)
- SFML (Paul)

Excalidraw (Groupe: Bon Toutou):

https://excalidraw.com/#room=083618b6dba30c4d3015,y7rTSJ2e34O45Gda82s9bA

Excalidraw (Groupe: Les potes):

https://excalidraw.com/#room=4e12cb8705b475c7a732,O6KiSbROj55q1S9cL2oDNg

Interface & Class

```
IGame.hpp // with core rule in Makefile ?

#include "IGameObject.hpp"
#include <iostream>
#include <memory>
#include <vector>

namespace game {
    class IGame {
        public:
        virtual ~IGame() = default;

        virtual void init() = 0;

        virtual void update(std::size_t input) = 0;

        virtual void end() = 0;

        [[nodiscard]] virtual std::vector<std::shared_ptr<gameObject::IGameObject>> dumpObjects() const = 0;

        [[nodiscard]] virtual const std::string &getGameName() const noexcept = 0;
    };
}
```

```
IGraphic.hpp // with graphicals rule in Makefile

#include "IGraphObject.hpp"
#include <iostream>
#include <memory>
```

```
namespace graph {
    class IGraphic {
        public:
        virtual ~IGraphic() = default;

        virtual void init() = 0;

        virtual void destroy() = 0;

        virtual void updateWindow() = 0;

        [[nodiscard]] virtual const std::string &getLibName() const noexcept = 0;

        virtual void draw(object::IGraphObject &obj) = 0;

        [[nodiscard]] virtual int pollEvent() = 0;

        // animate();
    };
}
```

```
IGameObject.hpp // with graphicals rule in Makefile

#include "Core.hpp"
#include <utility>

namespace graph {
    using coord = std::pair<std::size_t, std::size_t>;

    class IDrawable {
        public:
        virtual ~IGameObject() = default;

        [[nodiscard]] virtual core::GameObject getType() const noexcept = 0;

        [[nodiscard]] virtual const coord &getPos() const noexcept = 0;

        virtual void setPos(const coord &pos) = 0;

        virtual const std::string &getUID() const noexcept = 0;

        virtual core::Color getColor() const noexcept = 0;
    };
}
```

```
#include <iostream>

namespace core {
    enum GameObject {
        SPRITE,
        TEXT,
        UI
    };

    enum Color {
        BLACK,
        WHITE,
        BLUE,
        GREEN,
        PURPLE,
        DARK_BLUE,
        RED,
        INVISIBLE
    };

    enum KeyBoard {
        PAUSE,
        MENU,
        EXIT,
        RESTART,
        ARROW_UP,
        ARROW_DOWN,
        ARROW_LEFT,
        ARROW_RIGHT,
        PREV_LIB,
        NEXT_LIB,
        PREV_GAME,
        NEXT_GAME,
        SELECT
    };
```

```
    class Core {
        Core() = default;

        ~Core() = default;

        void run();

        private:
        void _nextGame();

        void _prevGame();

        void _nextGraph();

        void _prevGraph();

        void _restart();

        void _toMenu();

        [[nodiscard]] std::size_t _getDeltaTime() const noexcept;

        void _setDeltaTime();

        std::size_t _deltaTime;
    };
}
```

```
AGame.hpp

namespace game {
    class AGame : public game::IGame {
        public:

        enum Asset {
            MAIN_MENU,
            WALL
        };

        enum KeyBoard {
            START = 's',
            PAUSE = 'p',
            MENU = 'm',
            EXIT = 'q'
        };

        explicit AGame(std::string gameName);

        ~AGame() override = default;

        [[nodiscard]] const std::string &getGameName() const noexcept override;

        static const std::map<Asset, std::string> mapAsset;

        protected:
        std::string _gameName;
    };
}
```

```
ADisplay.hpp

#include "IGraphic.hpp"
#include <vector>

namespace graph {
    class ADisplay : public graph::IGraphic {
        public:
        explicit ADisplay(std::string libName);

        ~ADisplay() override = default;

        [[nodiscard]] const std::string &getLibName() const noexcept override;

        protected:
        std::string _libName;

        std::string _title;

        std::size_t _height;

        std::size_t _width;
```

```
    };
}
```

```
AGameObject.hpp

#include "IGameObject.hpp"
#include <iostream>
#include <utility>
#include <vector>

namespace gameObject {
    class AGameObject : public gameObject::IGameObject {
        public:
        explicit AGameObject(core::GameObject type, std::string confPath);

        ~AGameObject() override = default;

        [[nodiscard]] core::GameObject getType() const noexcept override;

        void setType(core::GameObject type) override;

        [[nodiscard]] const std::pair<std::size_t, std::size_t> &getPos() const noexcept override;

        void setPos(const std::pair<std::size_t, std::size_t> &pos) override;

        void setPos(std::size_t x, std::size_t y) override;

        [[nodiscard]] const std::pair<std::string, core::Color> &loadSet() const noexcept override;

        void setBuffer() override;

        protected:
        core::GameObject _type;

        std::pair<std::size_t, std::size_t> _pos;

        std::string _confPath;

        std::pair<std::string, core::Color> _set;
    };
}
```

```
SdlEncapsulation.cpp / hpp

#include <SDL>

namespace graph {
  class SfmlEncapsulation final : public graph::ADisplay {
    void draw() const final {
      SDL_Rect rect = {0, 0, 100, 100}; // x, y, width, height
    };
  };
}
```

```
SfmlEncapsulation.cpp / hpp

#include <SFML>

namespace graph {
  class SfmlEncapsulation final : public graph::ADisplay {
    void draw() const final {
      sf::RectangleShape rectangle(sf::Vector2f(120, 50));
    };
  };
}
```

```
NcursesEncapsulation.cpp / hpp

#include <ncurses.h>

namespace graph {
  class NcursesEncapsulation final : public graph::ADisplay {
    void draw() const final {
      WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int y, int x);
    };
```

```
  };
}
```

```
Error.hpp

#include <exception>
#include <iostream>

namespace arc {
  class ArcadeError : public std::exception {
    public:
      explicit ArcadeError (std::string message, std::string component = "Unknown");

      ~ArcadeError () noexcept override = default;

      std::string const &where() const;

      const char *what() const noexcept final;

     protected:
       std::string _message;

       std::string _component;
    };

    class ParserError : public ArcadeError {
      public:
      explicit ParserError(std::string const &message, std::string const &component = "Unknown");

      ~ParserError() noexcept override = default;
    };
}

Error.cpp

#include <utility>

arc::ArcadeError::ArcadeError(std::string message, std::string component) : _message(std::move(message)), _component(std::move(compone
{
}

std::string const &arc::ArcadeError::where() const
{
    return (_component);
}

const char *arc::ArcadeError::what() const noexcept
{
    return (_message.c_str());
}

arc::ParserError::ParserError(const std::string &message, const std::string &component) : Arcade(message, component)
{
}
```

```
#include "IGame.hpp"
#include "IGraphic.hpp"
#include <memory>

int main()
{
  auto game = std::make_unique<game::Nibbler>();
  auto graph = std::make_unique<graph::NcursesEncapsulation>();

  while (true) { // Program loop
    graph.event() // receive event from keyboard
    if (checkChange()) // ch == '0' == SDL, ch == '1' == SFML , ch == '2' == Ncurses
      loadOtherLib(User::loadLib((graph || game)->getLibName())); // load another lib.so
     game.run(graph) {
        graph.draw(objects); // objects == std::vector<IObject>
      } // Game loop , if user exit return
  }
  return (0);
}
```

```
// leur game.so

auto lib = new Sdl();
auto pacman = new Sprite();
```

```
pacman.setTexturePath();
lib.draw(pacman);

// notre game.so

auto lib = new Sdl();
auto wall = new StatiObject();

lib.draw(wall);
```

- •••••••