DOCUMENTATION

ARCADE

SOMMAIRE :

I/ Introduction

Le projet est de reproduire une arcade. Le but est de lancer l'arcade avec une librairie graphique et de pouvoir lancer des librairies de jeu dynamiquement. On doit pouvoir changer de librairie graphique ou de jeux pendant que l'arcade marche. De plus l'arcade ne peut pas savoir quelle librairie graphique et de jeu elle fait marcher. De ce fait notre programme implémente une interface générique IDisplay et une interface générique IGame. Les 2 librairies doivent avoir une fonction « extern "C" entry_point » pour charger les bonnes librairies avec dlopen et dlsym.

II/ IDisplay, IText and ISprite

```cpp
class IDisplay {
  public:
    enum AvailableOptions {
        NO_OPTIONS = 0,
        SET_CHARACTER_SIZE = 1 << 0,
        MOUSE_MOVE = 1 << 1,
        SETTING_FONTS = 1 << 2,
    };
    #define isOptions(disp, opt) \
    ((disp)->availableOptions() & arcade::displayer::IDisplay::AvailableOptions::opt)

    virtual int availableOptions() const = 0;
    virtual void init(const std::string &winName, unsigned int framesLimit = 60) = 0;
    virtual void stop() = 0;
    virtual bool isOpen() const = 0;
    virtual void clearWindow() = 0;
    virtual void display() = 0;
    virtual void restartClock() = 0;
    virtual double getDeltaTime() const = 0;
    virtual arcade::data::Vector2u getWindowSize() const = 0;
    virtual std::vector<arcade::data::Event> getEvents() = 0;
    virtual void draw(std::unique_ptr<IText> &text) = 0;
    virtual void draw(std::unique_ptr<ISprite> &sprite) = 0;
    virtual std::unique_ptr<IText> createText() const = 0;
    virtual std::unique_ptr<IText> createText(const std::string &text) const = 0;
    virtual std::unique_ptr<ISprite> createSprite() const = 0;
    virtual std::unique_ptr<ISprite> createSprite(const std::string &spritePath,
        const std::vector<std::string> &asciiSprite,
        arcade::data::Vector2f scale = arcade::data::Vector2f{1, 1}) const = 0;
    virtual double scaleMoveX(double time) const = 0;
    virtual double scaleMoveY(double time) const = 0;
};
```

```cpp
virtual int availableOptions() const = 0;
```

Retourne une combinaison de l'enum AvailableOptions pour donner des informations sur ce qui est pris en charge par la librairie.

```cpp
virtual void init(const std::string &winName, unsigned int framesLimit = 60) = 0;
```

Cette fonction doit toujours être appeler en 1er, elle permet d'initialiser la fenêtre.

```cpp
virtual void stop() = 0;
```

Cette fonction doit toujours être appeler à la fin du programme pour stopper la fenêtre.

```
virtual bool isOpen() const = 0;
```

Indique si la fenêtre est ouverte.

```
virtual void clearWindow() = 0;
```

Efface ce qui est affiché sur la fenêtre.

```
virtual void display() = 0;
```

Affiche sur la fenêtre les éléments dessiner précédemment.

```
virtual void restartClock() = 0;
```

Permet de redémarrer la clock.

```
virtual double getDeltaTime() const = 0;
```

Retourne la durée depuis que la dernière image a été affiché.

```
virtual arcade::data::Vector2u getWindowSize() const = 0;
```

Retourne la taille de la largeur et de la hauteur de l'écran.

```
virtual std::vector<arcade::data::Event> getEvents() = 0;
```

Retourne un vecteur des événements.

```
virtual void draw(std::unique_ptr<IText> &text) = 0;
```

Permet de draw du texte contenue dans un unique_ptr. Il sera affiché la prochaine fois que display sera appelé.

```
virtual void draw(std::unique_ptr<ISprite> &sprite) = 0;
```

Permet de draw un sprite contenue dans un unique_ptr. Il sera affiché la prochaine fois que display sera appelé.

```
virtual std::unique_ptr<IText> createText() const = 0;
virtual std::unique_ptr<IText> createText(const std::string &text) const = 0;
```

Permet de créer un texte vide ou un texte spécifique.

```
virtual std::unique_ptr<ISprite> createSprite() const = 0;
virtual std::unique_ptr<ISprite> createSprite(const std::string &spritePath,
const std::vector<std::string> &asciiSprite, arcade::data::Vector2f scale =
arcade::data::Vector2f{1, 1}) const = 0;
```

Permet de créer un sprite vide ou un sprite spécifique.

```
virtual double scaleMoveX(double time) const = 0;
virtual double scaleMoveY(double time) const = 0;
```

Permet de déplacer un objet à une vitesse constante.

```
class IText {
  public:
    virtual void setText(const std::string &text) = 0;
    virtual std::string getText() const = 0;
    virtual void setPosition(arcade::data::Vector2f pos) = 0;
    virtual arcade::data::Vector2f getPosition() const = 0;
    virtual void setFont(const std::string &font) = 0;
    virtual void setColor(arcade::data::Color color) = 0;
    virtual arcade::data::Color getColor() const = 0;
    virtual void setCharacterSize(unsigned int size) = 0;
    virtual arcade::data::FloatRect getLocalBounds() const = 0;
    virtual arcade::data::FloatRect getGlobalBounds() const = 0;
    virtual void setOrigin(arcade::data::Vector2f origin) = 0;
    virtual arcade::data::Vector2f getOrigin() const = 0;
};
```

```
virtual void setText(const std::string &text) = 0;
```

Assigne un texte.

```
virtual std::string getText() const = 0;
```

Retourne le texte qui avait été assigné.

```
virtual void setPosition(arcade::data::Vector2f pos) = 0;
```

Place le texte à une position dans la fenêtre.

```
virtual arcade::data::Vector2f getPosition() const = 0;
```

Retourne la position où le texte est situé.

```
virtual void setFont(const std::string &font) = 0;
```

Assigne une police au texte.

```
virtual void setColor(arcade::data::Color color) = 0;
```

Assigne une couleur au texte.

```
virtual arcade::data::Color getColor() const = 0;
```

Retourne la couleur du texte.

```
virtual void setCharacterSize(unsigned int size) = 0;
```

Assigne la taille du texte.

```
virtual arcade::data::FloatRect getLocalBounds() const = 0;
```

Décrit le texte à son état original avant toute modifications. Left et top seront alors toujours fixés à 0.

```
virtual arcade::data::FloatRect getGlobalBounds() const = 0;
```

Décrit le texte tel qu'il est affiché. Les propriétés left et top correspondent à la position de l'objet, soit le coin supérieur gauche de l'objet. Width et heigth correspondent à la dimension réelle de du texte.

```
virtual void setOrigin(arcade::data::Vector2f origin) = 0;
```

Assigne une origine au texte.

```
virtual arcade::data::Vector2f getOrigin() const = 0;
```

Retourne l’origine du texte.

```cpp
class ISprite {
  public:
    virtual void setSprite(
        const std::string &spritePath, const std::vector<std::string> &asciiSprite) = 0;
    virtual void setPosition(arcade::data::Vector2f pos) = 0;
    virtual arcade::data::Vector2f getPosition() const = 0;
    virtual void move(arcade::data::Vector2f pos) = 0;
    virtual void move(float x, float y) = 0;
    virtual void setOrigin(arcade::data::Vector2f origin) = 0;
    virtual arcade::data::Vector2f getOrigin() const = 0;
    virtual arcade::data::FloatRect getLocalBounds() const = 0;
    virtual arcade::data::FloatRect getGlobalBounds() const = 0;
    virtual void setScale(arcade::data::Vector2f scale) = 0;
    virtual arcade::data::Vector2f getScale() const = 0;
    virtual void rotate(float anle) = 0;
    virtual void setRotation(float angle) = 0;
    virtual float getRotation() const = 0;
    virtual void setTextureRect(const arcade::data::IntRect &rect) = 0;
    virtual arcade::data::IntRect getTextureRect() const = 0;
    virtual void setColor(arcade::data::Color color,
        const std::vector<std::vector<arcade::data::Color>> &asciiColors) = 0;
};
```

```cpp
virtual void setSprite(
const std::string &spritePath, const std::vector<std::string> &asciiSprite) = 0;
```

Assigne un sprite.

```cpp
virtual void setPosition(arcade::data::Vector2f pos) = 0;
```

Assigne une position au sprite.

```cpp
virtual arcade::data::Vector2f getPosition() const = 0;
```

Retourne la position du sprite.

```cpp
virtual void move(arcade::data::Vector2f pos) = 0;
virtual void move(float x, float y) = 0;
```

Permettent de faire bouger un sprite.

```cpp
virtual void setOrigin(arcade::data::Vector2f origin) = 0;
```

Assigne une origine au sprite.

```cpp
virtual arcade::data::Vector2f getOrigin() const = 0;
```

Retourne l'origine du sprite.

```
virtual arcade::data::FloatRect getLocalBounds() const = 0;
```

Décrit l'objet à son état original avant toute modifications. Left et top seront alors toujours fixés à 0.

```
virtual arcade::data::FloatRect getGlobalBounds() const = 0;
```

Décrit le sprite tel qu'il est affiché. Les propriétés left et top correspondent à la position de l'objet, soit le coin supérieur gauche de l'objet. Width et heigth correspondent à la dimension réelle de du sprite.

```
virtual void setScale(arcade::data::Vector2f scale) = 0;
```

Assigne une scale au sprite.

```
virtual arcade::data::Vector2f getScale() const = 0;
```

Retourne la scale du sprite.

```
virtual void rotate(float angle) = 0;
```

Permet d'additionner un angle à l'angle du sprite.

```
virtual void setRotation(float angle) = 0;
```

Assigne un angle au sprite.

```
virtual float getRotation() const = 0;
```

Retourne l'angle du sprite.

```
virtual void setTextureRect(const arcade::data::IntRect &rect) = 0;
```

Assigne une texture au sprite.

```
virtual arcade::data::IntRect getTextureRect() const = 0;
```

Retourne la texture du sprite.

```
virtual void setColor(arcade::data::Color color,
const std::vector<std::vector<arcade::data::Color>> &asciiColors) = 0;
```

Assigne une couleur au sprite.

II/ IGame

```
namespace arcade
{
    namespace games
    {
        #define GAMES_ENTRY_POINT entry_point

        enum GameStatus {
            PLAYING,
            GAME_ENDED,
        };
        class IGame {
          public:
            virtual void init(std::shared_ptr<arcade::displayer::IDisplay> &disp) = 0;
            virtual GameStatus update() = 0;
            virtual void stop() = 0;
            virtual void restart() = 0;
            virtual unsigned int getScore() const = 0;

        };
    } // namespace games
} // namespace arcade
```

Le namespace games possède un enum GameStatus qui nous indique si la partie est en cours ou est finie.

```
virtual void init(std::shared_ptr<arcade::displayer::IDisplay> &disp) = 0;
```

Doit être la fonction appelé en 1er, elle permet d'initialiser le jeu.

```
virtual GameStatus update() = 0;
```

Permet de gérer le déplacement dans le jeu, de draw du texte et sprite et de réaliser tous les changements au cours du jeu.

```
virtual void stop() = 0;
```

Permet d'arrêter le jeu. Doit toujours être appelé à la fin du jeu.

```
virtual void restart() = 0;
```

Permet de recommencer le jeu.

```
virtual unsigned int getScore() const = 0;
```

Permet de récupérer le score de la partie passer.

IV/ Les events

```
enum EventType {
    WINDOW_CLOSED,
    KEY_PRESSED,
    MOUSE_MOVED,
    MOUSE_PRESSED,
    MOUSE_RELEASED,
};
enum KeyCode {
    ENTER = 10,
    ESCAPE = 27,
    SPACE = 32,
    SPECIAL_KEYS_START = 257,
    DOWN = 258,
    UP = 259,
    LEFT = 260,
    RIGHT = 261,
    BACKSPACE = 263,
};
enum MouseBtn {
    BTN_1,
    BTN_2,
    BTN_3,
    BTN_4,
};

struct Event {
    Event() : type(static_cast<EventType>(0)), x(0), y(0){};
    Event(EventType type) : type(type), x(0), y(0){};
    Event(EventType type, MouseBtn btn, int x, int y) : type(type), btn(btn), x(x), y(y){};
    Event(EventType type, int x, int y) : type(type), x(x), y(y){};
    Event(EventType type, KeyCode keyCode) : type(type), keyCode(keyCode), x(0), y(0){};
    Event(EventType type, char key) : type(type), key(key), x(0), y(0){};

    EventType type;
    union {
        KeyCode keyCode = static_cast<KeyCode>(0);
        char key;
        MouseBtn btn;
    };
    struct {
        int x;
        int y;
    };
};
```

V/ Les vecteurs

```cpp
template <typename T> struct Vector2 {
    Vector2() : x(0), y(0){};
    Vector2(T x) : x(x), y(0){};
    Vector2(T x, T y) : x(x), y(y){};
    template <typename U>
    Vector2(const Vector2<U> &vect)
        : x(static_cast<T>(vect.x)), y(static_cast<T>(vect.y)){};

    template <typename U> Vector2<T> &operator=(const Vector2<U> &other)
    {
        x = static_cast<T>(other.x);
        y = static_cast<T>(other.y);
        return *this;
    };

    template <typename U> Vector2<T> &operator+=(const Vector2<U> &other)
    {
        x += static_cast<T>(other.x);
        y += static_cast<T>(other.y);
        return *this;
    };

    Vector2<T> operator+(const Vector2<T> &other) const
    {
        return arcade::data::Vector2<T>{x + other.x, y + other.y};
    };

    template <typename U> bool operator==(const Vector2<U> &other) const
    {
        return x == static_cast<T>(other.x) && y == static_cast<T>(other.y);
    };

    Vector2<T> &move(T x)
    {
        this->x += x;
        return *this;
    };
    Vector2<T> &move(T x, T y)
    {
        this->x += x;
        this->y += y;
        return *this;
    };
    template <typename U> Vector2<T> &move(const Vector2<U> &other)
    {
        x += static_cast<T>(other.x);
        y += static_cast<T>(other.y);
        return *this;
    };

    T x;
    T y;
};
```

VI/ Les rectangles

```cpp
template <typename T> struct Rect {
    Rect() : top(0), left(0), width(0), height(0){};
    Rect(T width, T height) : top(0), left(0), width(width), height(height){};
    Rect(T top, T left, T width, T height)
        : top(top), left(left), width(width), height(height){};
    template <typename U>
    explicit Rect(U top, U left, U width, U height)
        : top(static_cast<T>(top)), left(static_cast<T>(left)),
          width(static_cast<T>(width)), height(static_cast<T>(height)){};
    template <typename U>
    Rect(const Rect<U> &rect)
        : top(static_cast<T>(rect.top)), left(static_cast<T>(rect.left)),
          width(static_cast<T>(rect.width)), height(static_cast<T>(rect.height)){};

    T top;
    T left;
    T width;
    T height;
};
```

VII/ Les couleurs

```
struct Color {
    Color() : r(255), g(255), b(255), a(255){};
    Color(uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha = 255)
        : r(red), g(green), b(blue), a(alpha){};

    bool operator==(const Color &other) const
    {
        return r == other.r && g == other.g && b == other.b && a == other.a;
    };

    uint8_t r;
    uint8_t g;
    uint8_t b;
    uint8_t a;

    static const Color Black;
    static const Color White;
    static const Color Red;
    static const Color Green;
    static const Color Blue;
    static const Color Yellow;
    static const Color Magenta;
    static const Color Cyan;
    static const Color Transparent;
};
```

VIII/ Les erreurs

```
class Error : public std::exception {
    public:
        Error(const std::string &msg);
        ~Error();

        const char *what() const noexcept override;

    protected:
    private:
        const std::string _msg;
};
```