Documentation Arcade

Create a game

To implement a new game, you must create a new class that inherits from the IGame interface. You must implement each methods with the same comportment as described below.
All of your game logic should be put in the Arcade::IGame::tick() method and all of your rendering logic should be put in the Arcade::IGame::render() method.
You must compile it as a .so file and put it in the games directory.

Create a graphical library

To create a new graphical library, you must create new classes that inherits from IGraphicLib, IRenderer, ATexture and ASprite each. The methods of theses classes must have the same comportment as described below.
You must compile your graphic library as a .so file and put it in the lib directory.

Namespace Arcade

Data structs

Arcade::Color

```
struct Color final {
   uint8_t r;
   uint8_t g;
   uint8_t b;
   uint8_t a;

   Color(uint8_t r, uint8_t g, uint8_t b, uint8_t a = 255) noexcept;
   ~Color() = default;

   Color &operator=(const Color& rhs) noexcept = default;

   uint32_t getValue() const noexcept;
};
```

Data

uint8_t r
    Contains the red color. Number between 0 and 255 included.
uint8_t g
    Contains the green color. Number between 0 and 255 included.
uint8_t b
    Contains the blue color. Number between 0 and 255 included.
uint8_t a
    Contains the alpha color. Number between 0 and 255 included.

Arcade::Color::Color()

```
Color(uint8_t r, uint8_t g, uint8_t b, uint8_t a = 255) noexcept;
```

This is the Color constructor. It takes as arguments numbers composed of a red, green and blue uint8_t (value between 0 and 255). It can also take an alpha argument.

Arcade::Color::getValue()

```
uint32_t getValue() const noexcept;
```

getValue() takes 0 arguments. It returns an uint32_t which is the four rgba uint8_t assembled.

```
uint32_t      [ ff ff ff ff ]
                r  g  b  a
```

Arcade::Vector

```
struct Vector {
   double x;
   double y;

   Vector(const Vector &v) noexcept = default;
   Vector(double x, double y) noexcept;
   ~Vector() = default;

   Vector &operator+=(const Vector &vec) noexcept;
   Vector &operator+=(const double &val) noexcept;
   Vector &operator-=(const Vector &vec) noexcept;
   Vector &operator-=(const double &val) noexcept;
   Vector &operator*=(const Vector &vec) noexcept;
   Vector &operator*=(const double &val) noexcept;
   Vector &operator/=(const Vector &vec);
   Vector &operator/=(const double &val);
   Vector operator+(const Vector &vec) const noexcept;
   Vector operator+(const double &val) const noexcept;
   Vector operator-(const Vector &vec) const noexcept;
   Vector operator-(const double &val) const noexcept;
   Vector operator*(const Vector &vec) const noexcept;
   Vector operator*(const double &val) const noexcept;
   Vector operator/(const Vector &vec) const;
   Vector operator/(const double &val) const;
   Vector &operator=(const Vector &vec) noexcept;
   Vector &operator=(const double &val) noexcept;
   bool operator==(const Vector &vec) const noexcept;
   bool operator!=(const Vector &vec) const noexcept;
};
```

Data

double x
double y

Arcade::Vector::Vector()

```
Vector(double x, double y) noexcept;
```

Vector() is the Vector struct constructor. It stores the two given arguments in to the struct attributes with the same name.

Arcade::Vector operators

operator+=

```
Vector &operator+=(const Vector &vec) noexcept;
```

Add **vec.x** to **this.x** and **vec.y** to **this.y**. It returns a reference to itself with the new added values.

```
Vector &operator+=(const double &val) noexcept;
```

Add **val** to **this.x** and **this.y**. It returns a reference to itself with the new added values.

operator-=

```
Vector &operator-=(const Vector &vec) noexcept;
```

Subtract **vec.x** to **this.x** and **vec.y** to **this.y**. It returns a reference to itself with the new subtracted values.

```
Vector &operator-=(const double &val) noexcept;
```

Add **val** to **this.x** and val to **this.y**. It returns a reference to itself with the new added values.

operator*=

```
Vector &operator*=(const Vector &vec) noexcept;
```

Multiply **vec.x** by **this.x** and **vec.y** by **this.y**. It returns a reference to itself with the new multiplied values.

```
Vector &operator*=(const double &val) noexcept;
```

Multiply **val** by **this.x** and **val** by **this.y**. It returns a reference to itself with the new multiplied values.

operator/=

```
Vector &operator/=(const Vector &vec);
```

Divide **this.x** by **vec.x** and **this.y** by **vec.y**. It returns a reference to itself with the new divided values.
If a division by 0 occurs, the function throws a **std::runtime_error()**.

```
Vector &operator/=(const double &val);
```

Divide **this.x** by **val** and **this.y** by **val**. It returns a reference to itself with the new divided values.
If a division by 0 occurs, the function throws a **std::runtime_error()**.

Operator+

```
Vector operator+(const Vector &vec) const noexcept;
```

Returns a new Vector containing the addition of **vec.x** to **this.x** in **x** and **vec.y** to **this.y** in **y**.

```
Vector operator+(const double &val) const noexcept;
```

Returns a new Vector containing the addition of **val** to **this.x** in **x** and **val** to **this.y** in **y**.

operator-

```
Vector operator-(const Vector &vec) const noexcept;
```

Returns a new Vector containing the subtraction of **vec.x** to **this.x** in **x** and **vec.y** to **this.y** in **y**.

```
Vector operator-(const double &val) const noexcept;
```

Returns a new Vector containing the subtraction of **val** to **this.x** in **x** and **val** to **this.y** in **y**.

operator*

```
Vector operator*(const Vector &vec) const noexcept;
```

Returns a new Vector containing the multiplication of **vec.x** by **this.x** in **x** and **vec.y** by **this.y** in **y**.

```
Vector operator*(const double &val) const noexcept;
```

Returns a new Vector containing the multiplication of **val** by **this.x** in **x** and **val** by **this.y** in **y**.

operator/

```
Vector operator/(const Vector &vec) const;
```

Returns a new Vector containing the division of **vec.x** by **this.x** in **x** and by **vec.y** by **this.y** in **y**.
If a division by 0 occurs, the function throws a **std::runtime_error()**.

```
Vector operator/(const double &val) const;
```

Returns a new Vector containing the division of **val** by **this.x** in **x** and by **val** by **this.y** in **y**.
If a division by 0 occurs, the function throws a **std::runtime_error()**.

operator=

```
Vector &operator=(const Vector &vec) noexcept;
```

Set **vec.x** to **this.x** and **vec.y** to **this.y**. Returns a reference to itself.

```
Vector &operator=(const double &val) noexcept;
```

Set **val** to **this.x** and **val** to **this.y**. Returns a reference to itself.

operator==

```
bool operator==(const Vector &vec) const noexcept;
```

Returns true if **this.x** is equal to **vec.x** and **this.y** is equal to **vec.y**. Otherwise, returns false.

operator!=

```
bool operator!=(const Vector &vec) const noexcept;
```

Returns true if **this.x** is different from **vec.x** or if **this.y** is different from **vec.y**. Otherwise, returns false.

Arcade::Rect

```
struct Rect final {
   Vector pos;
   Vector size;

   Rect() noexcept;
   Rect(const Vector &pos, const Vector &size) noexcept;
   Rect(double x, double y, double w, double h) noexcept;
   ~Rect() = default;

   bool operator==(const Rect &rhs) const noexcept;
   bool operator!=(const Rect &rhs) const noexcept;
   Rect &operator=(const Rect &rhs) noexcept = default;
};
```

Data

Vector pos

Vector containing the pos **x** and **y** of the Rect.

Vector size

Vector containing the size of the Rect. **x** is the width and **y** is the height.

Arcade::Rect::Rect()

Rect() is the constructor of the struct. It sets the struct values to the given ones.

```
Rect() noexcept;
```

Sets all the **pos** and **size** values to 0.

```
Rect(const Vector &pos, const Vector &size) noexcept;
```

Sets the struct **pos** value to the **pos** given as arguments and set the struct **size** value to **size** given as arguments.

```
Rect(double x, double y, double w, double h) noexcept;
```

Sets the struct **pos.x** value to **x**, **pos.y** value to **y**, **size.x** value to **w** and **size.y** value to **h**.

Arcade::Rect operators

operator==

```
bool operator==(const Rect &rhs) const noexcept;
```

Returns true if **this.pos** is equal to **rhs.pos** and **this.size** is equal to **rhs.size**. Otherwise, returns false.

operator!=

```
bool operator!=(const Rect &rhs) const noexcept;
```

Returns true if **this.pos** if different from **rhs.pos** or if **this.size** is different of **rhs.size**. Otherwise, returns false.

Interfaces

Arcade::IGame

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

   virtual void init(IGraphicLib *graphic) = 0;
   virtual void tick(IGraphicLib *graphic, double deltaTime) = 0;
   virtual void render(IGraphicLib *graphic) = 0;
   virtual void reloadResources(IGraphicLib *graphic) = 0;
   virtual bool isCloseRequested() const noexcept = 0;
};
```

Arcade::IGame::init()

```
virtual void init(IGraphicLib *graphic) = 0;
```

The init method is called before the game loop. It can be used for sprite initialization for example.

Arcade::IGame::tick()

```
virtual void tick(IGraphicLib *graphic, double deltaTime) = 0;
```

The tick method is called in each games loops, before the render method. It take a deltaTime as parameter corresponding of the time passed since the previous loop.

Arcade::IGame::render()

```
virtual void render(IGraphicLib *graphic) = 0;
```

The render method is called in each games loops, after the tick method. It’s meant to be the method which renders all the graphics operations (ex: Sprite, Rect, …).

Arcade::IGame::reloadResources()

```
virtual void reloadResources(IGraphicLib *graphic) = 0;
```

The reloadResources method is called when the graphical library is changed.

Arcade::IGame::isCloseRequested()

```cpp
virtual bool isCloseRequested() const noexcept = 0;
```

If the isCloseRequested method returns true, the game will be closed.

Arcade::IGraphicLib

```
class IGraphicLib {
public:
   enum GameKey {
       UP = (1 << 0),
       DOWN = (1 << 1),
       LEFT = (1 << 2),
       RIGHT = (1 << 3),
       PRIMARY = (1 << 4),
       SECONDARY = (1 << 5),
       START = (1 << 6),
       SELECT = (1 << 7)
   };

   enum CoreKey {
       PREV_GRAPHICAL_LIB = (1 << 0),
       NEXT_GRAPHICAL_LIB = (1 << 1),
       PREV_GAME_LIB = (1 << 2),
       NEXT_GAME_LIB = (1 << 3),
       RESTART_GAME = (1 << 4),
       BACK_TO_MENU = (1 << 5),
       EXIT = (1 << 6)
   };

   virtual ~IGraphicLib() = default;

   virtual uint8_t getGameKeyState() const noexcept = 0;
   virtual uint8_t getCoreKeyState() const noexcept = 0;
   virtual void sendGameKeyInput(GameKey input) noexcept = 0;
   virtual void sendCoreKeyInput(CoreKey input) noexcept = 0;
   virtual void pollEvents() = 0;

   virtual ATexture *createTexture(const void *buffer, const size_t &len) = 0;
   virtual ASprite *createSprite(const ATexture *texture, const Rect
                                 &spriteSheetRect, const Rect &posAndSize) =
0;

   virtual IRenderer &getRenderer() noexcept = 0;
   virtual bool isCloseRequested() const noexcept = 0;
};
```

Data

enum GameKey

Keybinds for the game.

Each value of this enum is on a different bit. This is useful for storing multiple pressed keys

and get them easily with a & mask.

enum CoreKey

Keybinds for the core.

Working like the GameKey

Arcade::IGraphicLib::getGameKeyState()

```
virtual uint8_t getGameKeyState() const noexcept = 0;
```

The getGameKeyState method returns an uint8_t containing the game keys.

Arcade::IGraphicLib::getCoreKeyState()

```
virtual uint8_t getCoreKeyState() const noexcept = 0;
```

The getCoreKeyState method returns an uint8_t containing the core keys.

Arcade::IGraphicLib::sendGameKeyState()

```
virtual void sendGameKeyInput(GameKey input) noexcept = 0;
```

The sendGameKeyInput method adds a GameKey input to the next getGameKeyState return.

Arcade::IGraphicLib::sendCoreKeyState()

```
virtual void sendCoreKeyInput(CoreKey input) noexcept = 0;
```

The sendCoreKeyInput method adds a CoreKey input to the next getCoreKeyState return.

Arcade::IGraphicLib::pollEvent()

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

The pollEvents method fetches all events (keys, close, …).

Arcade::IGraphicLib::createTexture()

```
virtual ATexture *createTexture(const void *buffer, const size_t &len) = 0;
```

The createTexture creates a texture from a buffer given as parameter of the given length. It returns the created ATexture.

Arcade::IGraphicLib::createSprite()

```
virtual ASprite *createSprite(const ATexture *texture, const Rect
                              &spriteSheetRect, const Rect &posAndSize) =
0;
```

The createSprite takes a texture as parameter, applies the Rect spriteSheetRect to it and sets it’s pos and size given is posAndSize. It returns the newly created sprite.

Arcade::IGraphicLib::getRenderer()

```
virtual IRenderer &getRenderer() noexcept = 0;
```

The getRenderer method returns the graphic library renderer.

Arcade::IGraphicLib::isCloseRequested()

```
virtual bool isCloseRequested() const noexcept = 0;
```

The isCloseRequested method returns true if the close event was catched.

Arcade::IRenderer

```
class IRenderer {
public:
   virtual ~IRenderer() = default;

   virtual void drawRectangle(const Rect &rect, const Color &color, bool
fill = true) = 0;
   virtual void drawSprite(const ASprite *sprite) = 0;
   virtual void drawText(const std::string &text, uint8_t fontSize, const
Vector &pos, const Color &color) = 0;

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

Arcade::IRenderer::drawRectangle()

```
virtual void drawRectangle(const Rect &rect, const Color &color,
                           bool fill = true) = 0;
```

The drawRectangle method adds a rectangle at the position **rect.pos** and the size **rect.size** to the renderer. The color of the rectangle is given as the second parameter. If **fill** is equal to true, the rectangle must be filled with the given color, otherwise, only the edges must be drawn.

Arcade::IRenderer::drawSprite()

```
virtual void drawSprite(const ASprite *sprite) = 0;
```

The drawSprite method adds the sprite given as parameter to the renderer.

Arcade::IRenderer::drawText()

```
virtual void drawText(const std::string &text, uint8_t fontSize,
                      const Vector &pos, const Color &color) = 0;
```

The drawText method adds a text to the renderer. The **text, fontSize**, **pos** and **color** are given as parameters.

Arcade::IRenderer::display()

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

The display method displays the renderer to the window.

Arcade::IRenderer::clear()

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

The clear method clears the renderer.

Abstract classes

Arcade::Atexture

```
class ATexture {
public:
   ATexture(const void *buffer, const size_t &len);
   virtual ~ATexture() = default;

   ATexture(const ATexture &) = delete;
   ATexture &operator=(const ATexture &) = delete;

   virtual const void *getBuffer() const noexcept final;
   virtual const size_t &getBufferLength() const noexcept final;
protected:
   const void *_buffer;
   const size_t _len;
};
```

Data

const void *_buffer
	The texture memory buffer
const size_t _len
	The texture memory buffer length

Arcade::ATexture::ATexture()

```
ATexture(const void *buffer, const size_t &len);
```

The ATexture constructor takes the texture buffer and its length as parameter.

Arcade::ATexture::getBuffer()

```
virtual const void *getBuffer() const noexcept final;
```

The getBuffer getter returns the texture buffer.

Arcade::ATexture::getBufferLength()

```
virtual const size_t &getBufferLength() const noexcept final;
```

The getBufferLength getter returns the texture buffer length.

Arcade::ASprite

```
class ASprite {
public:
   ASprite(const ATexture *texture, const Rect &spriteSheetRect, const Rect
&posAndSize);
   virtual ~ASprite() = default;

   ASprite(const ASprite &) = delete;
   virtual ASprite &operator=(const ASprite &) = delete;

   virtual void setPosAndSize(const Rect &posAndSize) = 0;
   virtual void setTextureRect(const Rect &newRect) = 0;

   virtual const ATexture *getTexture() const noexcept final;
   virtual const Rect &getSpriteSheetRect() const final;
   virtual const Rect &getPosAndSize() const final;
   virtual const Color &getFallbackColor() const final;
   virtual void setFallbackColor(const Color &color) noexcept final;
protected:
   const ATexture *_texture;
   Rect _spriteSheetRect;
   Rect _posAndSize;

   Color _fallbackColor{255, 255, 255, 255};
};
```

Data

const ATexture *_texture

A pointer to the sprite texture.

Rect _spriteSheetRect

The Rect to get from the texture for the sprite.

Rect _posAndSize

The position and the size of the sprite.

Color _fallbackColor

The fallback color for the graphic libraries that does not support texture display. (Ncurses)

Arcade::ASprite::ASprite()

```
ASprite(const ATexture *texture, const Rect &spriteSheetRect, const Rect
        &posAndSize);
```

The ASprite constructor takes a pointer the a texture, a first rect to apply to the texture and a second rect for the position and size.

Arcade::ASprite::setPosAndSize()

```
virtual void setPosAndSize(const Rect &posAndSize) = 0;
```

The setPosAndSize setter sets the position and the size of the sprite to the Rect **posAndSize** given as parameter.

Arcade::ASprite::setTextureRect()

```
virtual void setTextureRect(const Rect &newRect) = 0;
```

The setTextureRect setter sets the rect of the texture **newRect** to apply to the texture.

Arcade::ASprite::getTexture()

```
virtual const ATexture *getTexture() const noexcept final;
```

The getTexture getter returns the sprite's **_texture**.

Arcade::ASprite::getSpriteSheetRect()

```
virtual const Rect &getSpriteSheetRect() const final;
```

The getSpriteSheetRect getter returns the sprite **_spriteSheetRect**.

Arcade::ASprite::getPosAndSize()

```
virtual const Rect &getPosAndSize() const final;
```

The getPosAndSize getter returns the sprite’s **_posAndSize**.

Arcade::ASprite::getFallbackColor()

```
virtual const Color &getFallbackColor() const final;
```

The getFallbackColor getter returns the sprite’s **_fallbackColor**.

Arcade::ASprite::setFallbackColor()

```
virtual void setFallbackColor(const Color &color) noexcept final;
```

The setFallBackColor setter sets the **_fallbackColor** to the **color** parameter.