# **Unify** Documentation

**Arcade** Epitech - Promo **2023**

# What is **Unify** ?

Unify is a video game console interface, and a game engine that can run a game on multiple graphic libraries. As a developer, with this documentation, you can add new games and new libraries easily.

# How to install **Unify** ?

1. Clone the repository
2. Make sure all default graphics libraries are installed:
   SDL 2->https://wiki.libsdl.org/Installation
   SFML->https://www.sfml-dev.org/download-fr.php
   Ncurses->https://www.ostechnix.com/how-to-install-ncurses-library-in-linux

## What do you need to know about **Unify** ?

First of, if you want to use sprites on your games, make sure they are PNGs. To implement games or libraries, you will have to follow some rules which will be detailed below, you will see, it is pretty easy.

# Table of **contents** :

# How to add a **new library**

To make your library working with Unify, you will have to compile it as a shared library (.so file). Plus, an **extern “C” entryPoint()** function is needed, it returns a pointer to your library:

Then you need to know that your library will not communicate directly with the games. An abstract object is provided to help you in the implementation. Named ADisplayLibrary, it is composed of 2 methods and 6 attributes:

```
class ADisplayLibrary {
    public:
        ~ADisplayLibrary() = default;

        IRectangle *_rect;
        ICircle *_circle;
        IWindow *_window;
        IText *_text;
        ISprite *_sprite;
        IAudio *_audio;

        virtual void loadAsset(const std::string &path, const std::string &name, AssetType type) = 0;
        virtual void unloadAsset(const std::string &name, AssetType type) = 0;
        virtual Events updateEvents(Events *events) = 0;
    protected:
        std::map<std::string, void *> _assets;
    private:
};
```

**loadAsset() :** Loads an asset using its path in the _assets map with the name given as key. An AssetType is also needed, it define if you load a font, a sprite or a sound.

**unloadAsset :** Unloads a given asset from the asset map.

**updateEvents() :** Fills an Events structure which contains keyboard and mouse inputs

Every attributes beginning by ‘I’ are interfaces that separate use of simple object as a window or a text. Here are their composition:

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

        virtual void draw(IWindow *window) = 0;
        virtual void setPosition(Vector2 position) = 0;
        virtual void setSize(Vector2 size) = 0;
        virtual void setColor(Color color) = 0;

    protected:
    private:
};
```

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

        virtual void draw(IWindow *window) = 0;
        virtual void setPosition(Vector2 position) = 0;
        virtual void setRadius(float radius) = 0;
        virtual void setColor(Color color) = 0;

    protected:
    private:
};
```

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

        virtual void draw(IWindow *window) = 0;
        virtual void setPosition(Vector2 position) = 0;
        virtual void setColor(Color color) = 0;
        virtual void setFont(const std::string &fontIdx) = 0;
        virtual void setFontSize(int fontSize) = 0;
        virtual void setText(const std::string &text) = 0;
    protected:
    private:
};
```

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

        virtual void draw(IWindow *window) = 0;
        virtual void setPosition(Vector2 position) = 0;
        virtual void setSize(Vector2 size) = 0;
        virtual void setSize(Box body, Box frame) = 0;
        virtual void setSprite(const std::string &sprite) = 0;
        virtual void setOpacity(unsigned char opacity) = 0;
    protected:
    private:
};
```

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

        virtual bool isOpen() = 0;
        virtual float height() = 0;
        virtual float width() = 0;
        virtual void clear() = 0;
        virtual void display() = 0;
        virtual void close() = 0;
        virtual void create() = 0;
    protected:
    private:
};
```

All of these object will have to be implemented to make it all work correctly, they are simple of use but they still are not the ones being used by the game maker. Another object, links your Library and its 5 main objects, it is called Builder, let's talk about it in the next section.

# How to develop a game using **Unify** ?

Let us introduce what will be your best ally: the Builder. This object does everything you need. It gathers every functionalities of any library. With it you can easily access to all inputs (mouse, keyboard, buttons…), display and all the things you need for your game development. What does that mean? It means that even if you don't know how to use a graphical library such as SFML or SDL, you will be able to make a game just focusing on your game system. Let's understand how it is supposed to work:

First of all, the base of your game has to inherit from the Start class, which contains all of the basic methods for a game to be run on Unify

# **Unify** builder members :

- **_gameObjects** : the map of game objects of the builder, the key is the gameObject name (std::string)
- **_musicsPlaying** : a vector containing all the sounds currently played
- **_events** : a structure used to get the events currently happening
- **_tick** : a time_t holding a timestamp
- **_tickDiff** : a time_t used to compute time difference with the _tick member
- **_library** : a pointer to an ADisplayLibrary that holds the current display library used
- **_volume** : an int holding the current general volume

# **Unify** window management routines :

- **reloadLibrary** : takes an ADisplayLibrary as argument to swap display library
- **windowIsOpen** : returns a bool that tells if the window is open or not
- **windowClear** : clears the display
- **windowClose** : closes the window
- **windowCreate** : creates and initiates the window
- **windowDisplay** : displays the elements set on the window
- **windowHeight** : returns the current height of the window
- **windowWidth** : returns the current width of the window

# **Unify** basic drawing functions :

There are 5 basic drawing components in the builder : rectangles, circles, rectangles with border radius, text and sprites.

All of those components have 4 similar functions :

1. An **Init** function : the first function to be called when creating a new component. It instantiates the component in the map of components of the builder.
2. A **setPosition** function : sets the position of the component on the display
3. A **setSize** function : sets the size of the component on the display
4. A **draw** function : asks, through the builder, the graphic library to display the component

Then, each component has special functions to fit its own specifications :

- **Rectangle** :
  - **rectInit** : Initializes a new rectangle in the component map
  - **rectSetPosition** : Sets the position of a given rectangle on the display
  - **rectSetSize** : Sets the size of a given rectangle on the display
  - **rectSetColor** : Sets the color of a given rectangle
  - **rectDraw** : Draws a given rectangle on the display

- **Circle** :
  - **circleInit** : Initializes a new circle in the component map
  - **circleSetPosition** : Sets the position of a given circle on the display
  - **circleSetRadius** : Sets the radius of a given circle on the display
  - **circleSetColor** : Sets the color of a given circle
  - **circleDraw** : Draws a given circle on the display

- **Radius Rect**:
  - **radiusRectInit** : Initializes a new radius rect in the component map
  - **radiusRectSetPosition** : Sets the position of a given radius rect on the display
  - **radiusRectSetSize** : Sets the size of a given radius rect on the display
  - **radiusRectSetRadius** : Sets the border radius of a given radius rect on the display
  - **radiusRectSetColor** : Sets the color of a given radius rect
  - **radiusRectDraw** : Draws a given radius rect on the display

- **Text** :
  - **textInit** : Initializes a new text component in the component map
  - **textSetPosition** : Sets the position of a given text component on the display
  - **textSetFontSize** : Sets the font size of a given text component on the display
  - **textSetFont** : Sets the font of a given text component on the display
  - **textSetText** : Sets a given string to a given text component
  - **textSetColor** : Sets the color of a given text component
  - **textDraw** : Draws a given text on the display

- **Sprite** :
  - **spriteInit** : Initializes a new sprite in the component map
  - **spriteSetPosition** : Sets the position of a given sprite on the display
  - **spriteSetSize** (2 overloads) : Either sets the size of a given sprite on the display or sets its position in a spritesheet and the size of it
  - **spriteSetOpacity** : Sets the opacity of a given sprite
  - **spriteDraw** : Draws a given sprite on the display

## **Unify** collider functions :

- **isMouseInBox** : takes a Box (width, height, x and y) object as parameter and tells wether the mouse is in the box or not
- **gameObjectCollide** : tells wether two gameObjects are colliding or not
- **gameObjectCollideToBox** : tells wether a gameObject and a Box object are colliding or not
- **gameObjectCollideToRadius** : tells wether a gameObjects and a Circle are colliding or not
- **circleToCircleCollide** : tells wether two Circles are colliding or not
- **circleToRectCollide** : tells wether a Circle and a Rectangle are colliding or not
- **rectToRectCollide** : tells wether two Rectangles are colliding or not

## **Unify** Events functions :

- **updateEvents** : refreshes the events recorded
- **getEvents** : returns a list of events, used to check for user inputs

## **Unify** Time functions :

- **getTime** : returns the current time

## **Unify** Assets management functions :

- **loadAsset** : loads a given asset in the asset map
- **unloadAsset** : unloads a given asset from the asset map

## **Unify** Sound functions :

- **playSound** : plays a given sound once
- **playMusic** : plays a given sound as a loop
- **stopMusic** : stops a given sound
- **setVolume** : sets the general volume
- **getVolume** : returns the current general volume

## **Unify** GameObject functions :

- **objectExists** : tells wether a game object exists or not
- **deleteGameObject** : deletes a given game object from the gameObject map

## **Unify** Utils functions :

- **hexToColor** : converts a hexadecimal int into a Color object
- **getBody** : returns the Box object of a given game object

# **Unify** UI management :

Unify also comes with a bundle of ui management methods, to enrich your games experience and looks, which are :

**Button functions**, used to create two types of button : basic buttons and sprite based buttons.

## **Unify** basic buttons functions :

- **basicButtonInit** : initiates a button with its name in the gameObject map
- **basicButtonSetDisplayBox** : sets the Box of a given basic button
- **basicButtonSetRadius** : sets the border radius of a given basic button
- **basicButtonSetBackgroundColors** : sets the background colors of a given basic button (colors for idle, hover and clicked states)
- **basicButtonSetTextColors** : sets the font color of a given basic button
- **basicButtonSetText** : sets the string to be displayed in a given basic button
- **basicButtonSetFontSize** : sets the font size of the given basic button text
- **basicButtonSetFont** : sets the font of a given basic button text
- **getBody** : returns the Box object of a given game object

## **Unify** sprite buttons functions :

- **spriteButtonInit** : initiates a button with its name in the gameObject map
- **spriteButtonSetDisplayBox** : sets the Box of a given sprite button
- **spriteButtonSetSpriteBoxes** : sets the Boxes of a given sprite button, each box corresponds to a button state (idle, hover, clicked)
- **spriteButtonSetSprite** : sets the sprite to be linked to a spriteButton

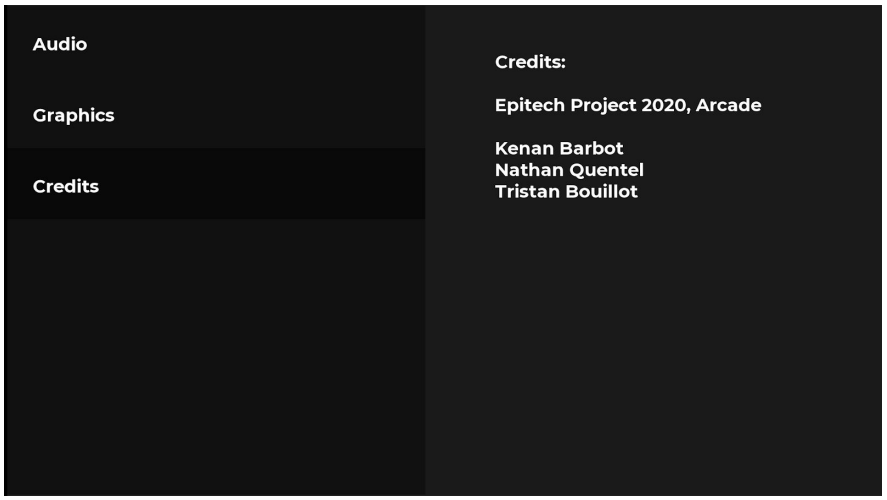
### **buttonDraw :** a generic function to draw any type of button

**Slider functions**, used to draw for example a volume slider :


## **Unify** slider functions :

- **sliderInit** : initiates a slider with its name in the gameObject map
- **sliderSetWidth** : sets the width of a given slider
- **sliderSetPosition** : sets the position on the display of a given slider
- **sliderSetBackgroundColor** : sets the background color of a given slider
- **sliderSetSliderColor** : sets the color of a given slider
- **sliderDraw** : draws a given slider on the display

Finally, **selector functions**, which are used to design simple selector menus like this :




# **Unify** selector functions :

- **selectorInit** : initiates a selector with its name in the gameObject map
- **selectorSetDisplayBox** : sets the Box (dimensions and position) of a given selector
- **selectorSetBackgroundColors** : sets the colors of a given selector for each state (idle, hover and click)
- **selectorSetFont** : sets the font for the texts to be displayed in a given selector menu
- **selectorSetItems** : sets the different categories of a given selector menu
- **selectorDraw** : draws a given selector menu on the display

**N.B :** in every function that uses the builder object, **the builder has to be named ‘b’** in order for your game to function properly and to be able to develop a responsive game using the **VW** and **VH** macros.