

Arcade Project

Generated by Doxygen 1.9.7

1 README	1
1.1 Arcade	1
1.1.1 Subject	1
1.1.2 Build	1
1.1.3 Documentation	2
1.1.4 Libraries	2
1.1.5 Usage	2
1.1.6 Documentation	2
1.1.7 Contributors	2
2 IGrid	3
3 IText	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Class Index	9
5.1 Class List	9
6 File Index	11
6.1 File List	11
7 Class Documentation	13
7.1 IGrid::Cell Struct Reference	13
7.1.1 Member Data Documentation	13
7.1.1.1 color	13
7.1.1.2 x	13
7.1.1.3 y	13
7.2 IClock Class Reference	14
7.2.1 Constructor & Destructor Documentation	14
7.2.1.1 ~IClock()	14
7.2.2 Member Function Documentation	14
7.2.2.1 getTimeElapsed()	14
7.2.2.2 initClock()	15
7.2.2.3 resetClock()	15
7.2.2.4 startClock()	15
7.3 IEntity Class Reference	15
7.3.1 Member Enumeration Documentation	16
7.3.1.1 Color	16
7.3.2 Constructor & Destructor Documentation	16
7.3.2.1 ~IEntity()	16
7.3.3 Member Function Documentation	16
7.3.3.1 destroy()	16

7.3.3.2 displayEntity()	17
7.3.3.3 setPosition()	17
7.4 IGame Class Reference	17
7.4.1 Constructor & Destructor Documentation	18
7.4.1.1 ~IGame()	18
7.4.2 Member Function Documentation	18
7.4.2.1 getEventBinding()	18
7.4.2.2 processGameTick()	18
7.4.2.3 restart()	19
7.5 IGraphicalFactory Class Reference	19
7.5.1 Constructor & Destructor Documentation	19
7.5.1.1 ~IGraphicalFactory()	19
7.5.2 Member Function Documentation	20
7.5.2.1 createIClock()	20
7.5.2.2 createWindow()	21
7.5.2.3 destroyRessource()	21
7.5.2.4 loadResource()	21
7.6 IGrid Class Reference	22
7.6.1 Constructor & Destructor Documentation	23
7.6.1.1 ~IGrid()	23
7.6.2 Member Function Documentation	23
7.6.2.1 create()	23
7.6.2.2 updateCell()	23
7.7 IText Class Reference	24
7.7.1 Member Enumeration Documentation	24
7.7.1.1 EntityType	24
7.7.2 Constructor & Destructor Documentation	25
7.7.2.1 ~IText()	25
7.7.3 Member Function Documentation	25
7.7.3.1 changeString()	25
7.7.3.2 create()	25
7.8 IWindow Class Reference	26
7.8.1 Member Typedef Documentation	27
7.8.1.1 EventCallBack	27
7.8.1.2 EventHandler	27
7.8.2 Member Enumeration Documentation	27
7.8.2.1 EventType	27
7.8.3 Constructor & Destructor Documentation	27
7.8.3.1 ~IWindow()	27
7.8.4 Member Function Documentation	28
7.8.4.1 callEvent()	28
7.8.4.2 clear()	28

7.8.4.3 closeWindow()	28
7.8.4.4 createIGrid()	28
7.8.4.5 createIText()	29
7.8.4.6 display()	29
7.8.4.7 eventPollEvent()	29
7.8.4.8 initWindow()	29
7.8.4.9 loadEventBindings()	30
7.8.4.10 windowIsOpen()	30
8 File Documentation	31
8.1 Interface/Game/IGame.hpp File Reference	31
8.2 IGame.hpp	31
8.3 Interface/Graphical/IClock.hpp File Reference	31
8.4 IClock.hpp	32
8.5 Interface/Graphical/IEntity.hpp File Reference	32
8.6 IEntity.hpp	32
8.7 Interface/Graphical/IGraphicalFactory.hpp File Reference	33
8.8 IGraphicalFactory.hpp	33
8.9 Interface/Graphical/IGrid.md File Reference	34
8.10 Interface/Graphical/IText.md File Reference	34
8.11 Interface/Graphical/IWindow.hpp File Reference	34
8.12 IWindow.hpp	34
8.13 README.md File Reference	35
Index	37

Chapter 1

README

A gaming platform.

1.1 Arcade

Arcade is a gaming platform, a program that lets the user choose a game to play and keeps a register of player scores.

1.1.1 Subject

1.1.2 Build

Requirements

- C++20 COMPILER
- cmake VERSION 3.5.1

Libraries

- SFML
- SDL2
- SDL2_TTF
- NCURSES

CMake

Simple guide to setup the project via CMake as follows:

```
$ mkdir ./build/ && cd ./build/
$ cmake .. -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release
[. . .]
$ cmake --build .
[. . .]
$ cd ..
$ ls ./arcade ./lib/
./arcade

./lib/:
arcade_ncurses.so
arcade_pacman.so
arcade_sdl2.so
arcade_sfml.so
arcade_solarfox.so
[. . .]
```

1.1.3 Documentation

This project is documented with Doxygen, which is the de facto standard tool for generating documentation from annotated C++ sources.

There is also a `./doc` directory, explaining how to implement new graphics libraries or game libraries compatible with the system.

1.1.4 Librairies

Graphicals libraries

The nCurses, SDL2 and SFML graphical libraries have been implemented.

- nCurses (arcade_ncurses.so)
- SDL2 (arcade_sdl2.so)
- SFML (arcade_sfml.so)

Game libraries

The Snake and Nibbler game libraries have been implemented.

- Snake (arcade_snake.so)
- Nibbler (arcade_nibbler.so)

1.1.5 Usage

USAGE:

```
./arcade path_to_graphical_lib
```

DESCRIPTION:

```
/lib folder          path to the initial graphical library to load (./lib/arcade_lib_name.so)
```

COMMANDS (azerty layout):

Y	Next graphics lib.
U	Next game.
I	Restart the game.
O	Go back to the menu.
P	Exit.
D	Move right.
Q	Move left.
Z	Move up.
S	Move down.

1.1.6 Documentation

1.1.7 Contributors

- [Raphael Camblong](#)
- [Leopold Sallan Gemard](#)
- [Pierre Bouillard](#)

Chapter 2

IGrid

Chapter 3

IText

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

IGrid::Cell	13
IClock	14
IEntity	15
IGrid	22
IText	24
IGame	17
IGraphicalFactory	19
IWindow	26

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

IGrid::Cell	13
IClock	14
IEntity	15
IGame	17
IGraphicalFactory	19
IGrid	22
IText	24
IWindow	26

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Interface/Game/ IGame.hpp	31
Interface/Graphical/ IClock.hpp	31
Interface/Graphical/ IEntity.hpp	32
Interface/Graphical/ IGraphicalFactory.hpp	33
Interface/Graphical/ IWindow.hpp	34

Chapter 7

Class Documentation

7.1 IGrid::Cell Struct Reference

```
#include <IEntity.hpp>
```

Public Attributes

- [Color color](#)
- [size_t x](#)
- [size_t y](#)

7.1.1 Member Data Documentation

7.1.1.1 color

```
Color IGrid::Cell::color
```

7.1.1.2 x

```
size_t IGrid::Cell::x
```

7.1.1.3 y

```
size_t IGrid::Cell::y
```

The documentation for this struct was generated from the following file:

- Interface/Graphical/[IEntity.hpp](#)

7.2 IClock Class Reference

```
#include <IClock.hpp>
```

Public Member Functions

- `~IClock()`=default
- virtual void `startClock()`=0
Start the clock used to measure time in the game or simulation. Its purpose is to start the clock used to measure time in the game or simulation.
- virtual double `getTimeElapsed()`=0
Get the time elapsed since the start of the clock. Its purpose is to retrieve the time elapsed since the start of the clock used to measure time in the game or simulation.
- virtual void `resetClock()`=0
Reset the clock used to measure time in the game or simulation. Its purpose is to reset the clock used to measure time in the game or simulation to its initial state.
- virtual void `initClock()`=0
Initialize the clock used to measure time in the game or simulation. Its purpose is to initialize the clock used to measure time in the game or simulation, setting it to its initial state.

7.2.1 Constructor & Destructor Documentation

7.2.1.1 ~IClock()

```
IClock::~IClock ( ) [default]
```

7.2.2 Member Function Documentation

7.2.2.1 getTimeElapsed()

```
virtual double IClock::getTimeElapsed ( ) [pure virtual]
```

Get the time elapsed since the start of the clock. Its purpose is to retrieve the time elapsed since the start of the clock used to measure time in the game or simulation.

Returns

The time elapsed, in seconds.

7.2.2.2 initClock()

```
virtual void IClock::initClock ( ) [pure virtual]
```

Initialize the clock used to measure time in the game or simulation. Its purpose is to initialize the clock used to measure time in the game or simulation, setting it to its initial state.

7.2.2.3 resetClock()

```
virtual void IClock::resetClock ( ) [pure virtual]
```

Reset the clock used to measure time in the game or simulation. Its purpose is to reset the clock used to measure time in the game or simulation to its initial state.

7.2.2.4 startClock()

```
virtual void IClock::startClock ( ) [pure virtual]
```

Start the clock used to measure time in the game or simulation. Its purpose is to start the clock used to measure time in the game or simulation.

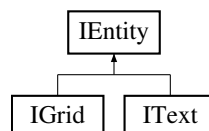
The documentation for this class was generated from the following file:

- [Interface/Graphical/IClock.hpp](#)

7.3 IEntity Class Reference

```
#include <IEntity.hpp>
```

Inheritance diagram for IEntity:



Public Types

- enum class `Color` {
 `Blue` , `Green` , `Orange` , `Red` ,
 `Brown` , `Yellow` }

Public Member Functions

- `~IEntity()`=default
- virtual void `setPosition` (int x, int y)=0
Sets the position of the entity. Its purpose is to set the position of the entity to the specified x and y coordinates. The function takes two integer parameters representing the x and y coordinates to set as the new position of the entity.
- virtual void `destroy` ()=0
Destroys any resources used in the implementation. Its purpose is to destroy any resources used in the implementation, such as windows or textures.
- virtual void `displayEntity` ()=0
Displays an entity on the screen. Its purpose is to display an entity on the screen, such as a sprite or image.

7.3.1 Member Enumeration Documentation

7.3.1.1 Color

```
enum class IEntity::Color [strong]
```

Enumerator

Blue	
Green	
Orange	
Red	
Brown	
Yellow	

7.3.2 Constructor & Destructor Documentation

7.3.2.1 ~IEntity()

```
IEntity::~IEntity ( ) [default]
```

7.3.3 Member Function Documentation

7.3.3.1 destroy()

```
virtual void IEntity::destroy ( ) [pure virtual]
```

Destroys any resources used in the implementation. Its purpose is to destroy any resources used in the implementation, such as windows or textures.

7.3.3.2 displayEntity()

```
virtual void IEntity::displayEntity ( ) [pure virtual]
```

Displays an entity on the screen. Its purpose is to display an entity on the screen, such as a sprite or image.

7.3.3.3 setPosition()

```
virtual void IEntity::setPosition (
    int x,
    int y ) [pure virtual]
```

Sets the position of the entity. Its purpose is to set the position of the entity to the specified x and y coordinates. The function takes two integer parameters representing the x and y coordinates to set as the new position of the entity.

Parameters

x	
y	

The documentation for this class was generated from the following file:

- Interface/Graphical/[IEntity.hpp](#)

7.4 IGame Class Reference

```
#include <IGame.hpp>
```

Public Member Functions

- [~IGame](#) ()=default
- virtual [IWindow::EventHandler](#) & [getEventBinding](#) ()=0

Returns the event handler object for the window. Its purpose is to return the event handler object for the window, allowing the client to modify or access event bindings. The function returns a reference to the [IWindow::EventHandler](#) object associated with the window.
- virtual bool [processGameTick](#) ([IGrid](#) &grid, [IText](#) &scoreText, [IText](#) &timeText, [IClock](#) &clock)=0

Process a single tick of the game or simulation. Its purpose is to process a single tick of the game or simulation, updating the state of the game or simulation accordingly. The function takes four parameters: a reference to an [IGrid](#) object representing the game board or simulation space, a reference to an [IText](#) object representing the score or other information to be displayed, a reference to an [IClock](#) object representing the game or simulation timer, and a reference to an [IText](#) object representing the time or other information to be displayed. The function returns a boolean value indicating whether the game or simulation should continue running. If false, the game or simulation should end and any necessary cleanup should be performed.
- virtual void [restart](#) ()=0

Restarts the implementation's game or simulation. Its purpose is to restart the game or simulation implemented by the class, resetting all necessary variables and objects to their initial state.

7.4.1 Constructor & Destructor Documentation

7.4.1.1 ~IGame()

```
IGame::~IGame ( ) [default]
```

7.4.2 Member Function Documentation

7.4.2.1 getEventBinding()

```
virtual IWindow::EventHandler & IGame::getEventBinding ( ) [pure virtual]
```

Returns the event handler object for the window. Its purpose is to return the event handler object for the window, allowing the client to modify or access event bindings. The function returns a reference to the [IWindow::EventHandler](#) object associated with the window.

Returns

[IWindow::EventHandler&](#)

7.4.2.2 processGameTick()

```
virtual bool IGame::processGameTick (
    IGrid & grid,
    IText & scoreText,
    IText & timeText,
    IClock & clock ) [pure virtual]
```

Process a single tick of the game or simulation. Its purpose is to process a single tick of the game or simulation, updating the state of the game or simulation accordingly. The function takes four parameters: a reference to an [IGrid](#) object representing the game board or simulation space, a reference to an [IText](#) object representing the score or other information to be displayed, a reference to an [IClock](#) object representing the game or simulation timer, and a reference to an [IText](#) object representing the time or other information to be displayed. The function returns a boolean value indicating whether the game or simulation should continue running. If false, the game or simulation should end and any necessary cleanup should be performed.

Parameters

<i>grid</i>	
<i>scoreText</i>	
<i>timeText</i>	
<i>clock</i>	

Returns

true
false

7.4.2.3 restart()

```
virtual void IGame::restart ( ) [pure virtual]
```

Restarts the implementation's game or simulation. Its purpose is to restart the game or simulation implemented by the class, resetting all necessary variables and objects to their initial state.

The documentation for this class was generated from the following file:

- [Interface/Game/IGame.hpp](#)

7.5 IGraphicalFactory Class Reference

```
#include <IGraphicalFactory.hpp>
```

Public Member Functions

- [~IGraphicalFactory](#) ()=default
- virtual void [loadResource](#) ()=0
Loads the necessary resources required for the implementation of the program. Its purpose is to load any required resources that are used in the implementation, such as textures, sounds, or fonts, into memory so they can be used later on.
- virtual void [destroyRessource](#) ()=0
Releases the resources held by the implementation. Its purpose is to release any resources that have been allocated or loaded by the implementation, such as textures, sounds, or fonts that were loaded during the course of the program. It ensure that all resources are properly released to avoid any memory leaks or resource conflicts.
- virtual std::unique_ptr< [IWindow](#) > [createWindow](#) (std::string name, size_t width, size_t height)=0
This function creates a window object required for the implementation.
- virtual std::unique_ptr< [IClock](#) > [createIClock](#) ()=0
Creates an [IClock](#) object required for the implementation. Its purpose is to create a new [IClock](#) object, which is used to track time in the implementation.

7.5.1 Constructor & Destructor Documentation**7.5.1.1 ~IGraphicalFactory()**

```
IGraphicalFactory::~IGraphicalFactory ( ) [default]
```

7.5.2 Member Function Documentation

7.5.2.1 createIClock()

```
virtual std::unique_ptr< IClock > IGraphicalFactory::createIClock ( ) [pure virtual]
```

Creates an [IClock](#) object required for the implementation. Its purpose is to create a new [IClock](#) object, which is used to track time in the implementation.

Parameters

<i>width</i>	
<i>height</i>	

Returns

`std::unique_ptr<IClock>` A `unique_ptr<IClock>` object pointing to the newly created [IClock](#).

7.5.2.2 createWindow()

```
virtual std::unique_ptr< IWindow > IGraphicalFactory::createWindow (
    std::string name,
    size_t width,
    size_t height ) [pure virtual]
```

This function creates a window object required for the implementation.

Parameters

<i>name</i>	A string specifying the name of the window to be created.
<i>width</i>	A <code>size_t</code> specifying the width of the window to be created.
<i>height</i>	A <code>size_t</code> specifying the height of the window to be created.

Returns

`std::unique_ptr<IWindow>` A `unique_ptr<IWindow>` object pointing to the newly created window.

7.5.2.3 destroyResource()

```
virtual void IGraphicalFactory::destroyResource ( ) [pure virtual]
```

Releases the resources held by the implementation. Its purpose is to release any resources that have been allocated or loaded by the implementation, such as textures, sounds, or fonts that were loaded during the course of the program. It ensure that all resources are properly released to avoid any memory leaks or resource conflicts.

7.5.2.4 loadResource()

```
virtual void IGraphicalFactory::loadResource ( ) [pure virtual]
```

Loads the necessary resources required for the implementation of the program. Its purpose is to load any required resources that are used in the implementation, such as textures, sounds, or fonts, into memory so they can be used later on.

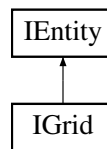
The documentation for this class was generated from the following file:

- [Interface/Graphical/IGraphicalFactory.hpp](#)

7.6 IGrid Class Reference

```
#include <IEntity.hpp>
```

Inheritance diagram for IGrid:



Classes

- struct [Cell](#)

Public Member Functions

- [~IGrid](#) ()=default
- virtual void [create](#) (int width, int height)=0
Creates a new object in the implementation with the given width and height. Its purpose is to create a new object in the implementation with the given width and height. The function takes two integer parameters, representing the width and height of the new object to be created.
- virtual void [updateCell](#) (int x, int y, [IEntity::Color](#))=0
a pure virtual function that must be implemented by the derived class. Its purpose is to update the color of a cell in the implementation's grid, identified by its x and y coordinates. The function takes three parameters: two integer parameters representing the x and y coordinates of the cell, and a parameter of type [IEntity::Color](#) representing the new color to assign to the cell.

Public Member Functions inherited from [IEntity](#)

- [~IEntity](#) ()=default
- virtual void [setPosition](#) (int x, int y)=0
Sets the position of the entity. Its purpose is to set the position of the entity to the specified x and y coordinates. The function takes two integer parameters representing the x and y coordinates to set as the new position of the entity.
- virtual void [destroy](#) ()=0
Destroys any resources used in the implementation. Its purpose is to destroy any resources used in the implementation, such as windows or textures.
- virtual void [displayEntity](#) ()=0
Displays an entity on the screen. Its purpose is to display an entity on the screen, such as a sprite or image.

Additional Inherited Members

Public Types inherited from [IEntity](#)

- enum class [Color](#) {
[Blue](#) , [Green](#) , [Orange](#) , [Red](#) ,
[Brown](#) , [Yellow](#) }

7.6.1 Constructor & Destructor Documentation

7.6.1.1 ~IGrid()

```
IGrid::~IGrid ( ) [default]
```

7.6.2 Member Function Documentation

7.6.2.1 create()

```
virtual void IGrid::create (
    int width,
    int height ) [pure virtual]
```

Creates a new object in the implementation with the given width and height. Its purpose is to create a new object in the implementation with the given width and height. The function takes two integer parameters, representing the width and height of the new object to be created.

Parameters

<i>width</i>	
<i>height</i>	

7.6.2.2 updateCell()

```
virtual void IGrid::updateCell (
    int x,
    int y,
    IEntity::Color ) [pure virtual]
```

a pure virtual function that must be implemented by the derived class. Its purpose is to update the color of a cell in the implementation's grid, identified by its x and y coordinates. The function takes three parameters: two integer parameters representing the x and y coordinates of the cell, and a parameter of type [IEntity::Color](#) representing the new color to assign to the cell.

Parameters

<i>x</i>	
<i>y</i>	

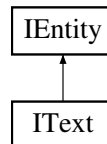
The documentation for this class was generated from the following file:

- Interface/Graphical/[IEntity.hpp](#)

7.7 IText Class Reference

```
#include <IEntity.hpp>
```

Inheritance diagram for IText:



Public Types

- enum class [EntityType](#) { [GridEntity](#) , [SpriteEntity](#) , [TextEntity](#) }

Public Types inherited from IEntity

- enum class [Color](#) {
[Blue](#) , [Green](#) , [Orange](#) , [Red](#) ,
[Brown](#) , [Yellow](#) }

Public Member Functions

- [~IText](#) ()=default
- virtual void [create](#) (std::string str)=0
Creates a new object in the implementation. Its purpose is to create a new object in the implementation based on the given string parameter. The function take a string as parameter, which can be used to determine the type or properties of the object to be created.
- virtual void [changeString](#) (std::string str)=0
Changes the string property of the object in the implementation. Its purpose is to change the string property of the object in the implementation. The function takes a string parameter, which represents the new value for the string property of the object.

Public Member Functions inherited from IEntity

- [~IEntity](#) ()=default
- virtual void [setPosition](#) (int x, int y)=0
Sets the position of the entity. Its purpose is to set the position of the entity to the specified x and y coordinates. The function takes two integer parameters representing the x and y coordinates to set as the new position of the entity.
- virtual void [destroy](#) ()=0
Destroys any resources used in the implementation. Its purpose is to destroy any resources used in the implementation, such as windows or textures.
- virtual void [displayEntity](#) ()=0
Displays an entity on the screen. Its purpose is to display an entity on the screen, such as a sprite or image.

7.7.1 Member Enumeration Documentation

7.7.1.1 EntityType

```
enum class IText::EntityType [strong]
```

Enumerator

GridEntity	
SpriteEntity	
TextEntity	

7.7.2 Constructor & Destructor Documentation

7.7.2.1 ~IText()

```
IText::~IText ( ) [default]
```

7.7.3 Member Function Documentation

7.7.3.1 changeString()

```
virtual void IText::changeString (
    std::string str ) [pure virtual]
```

Changes the string property of the object in the implementation. Its purpose is to change the string property of the object in the implementation. The function takes a string parameter, which represents the new value for the string property of the object.

Parameters

<i>str</i>	
------------	--

7.7.3.2 create()

```
virtual void IText::create (
    std::string str ) [pure virtual]
```

Creates a new object in the implementation. Its purpose is to create a new object in the implementation based on the given string parameter. The function take a string as parameter, which can be used to determine the type or properties of the object to be created.

Parameters

<i>str</i>	
------------	--

The documentation for this class was generated from the following file:

- [Interface/Graphical/IEntity.hpp](#)

7.8 IWindow Class Reference

```
#include <IWindow.hpp>
```

Public Types

- enum class [EventType](#) {
 [UP_pressed](#) , [DOWN_pressed](#) , [LEFT_pressed](#) , [RIGHT_pressed](#) ,
 [QUIT](#) , [PAUSE](#) , [RESUME](#) , [NEXT_GAME](#) ,
 [NEXT_LIB](#) , [RESTART](#) , [GO_TO_MENU](#) }
- using [EventCallback](#) = std::function< void()>
- using [EventHandler](#) = std::map< [IWindow::EventType](#), [EventCallback](#) >

Public Member Functions

- [~IWindow](#) ()=default
- virtual void [initWindow](#) (std::string name, size_t width, size_t height)=0
Initializes the window object required for the implementation. Its purpose is to initialize the window object with the specified name, width, and height.
- virtual void [closeWindow](#) ()=0
loses the window object used in the implementation. Its purpose is to close and destroy the window object used in the implementation.
- virtual bool [windowsOpen](#) ()=0
Checks whether the window object used in the implementation is currently open. Its purpose is to check whether the window object used in the implementation is currently open.
- virtual void [clear](#) ()=0
Clears the rendering buffer used in the implementation. Its purpose is to clear the rendering buffer used in the implementation. This is typically done at the beginning of each frame to ensure that the previous frame's rendering is not visible in the current frame.
- virtual void [display](#) ()=0
Displays the rendered content on the window object used in the implementation. Its purpose is to display the rendered content on the window object used in the implementation. This is typically done at the end of each frame to show the newly rendered content on the screen.
- virtual void [callEvent](#) (const [IWindow::EventType](#))=0
Calls the event handler function for the specified event type. Its purpose is to call the event handler function for the specified event type. This function is typically called by the application's main loop to handle user input or other events.
- virtual void [loadEventBindings](#) ([EventHandler](#) &)=0
loads the event bindings for the specified event handler object. Its purpose is to load the event bindings for the specified event handler object. This function is typically called at the beginning of the application to register event handlers for various types of events (e.g. mouse clicks, key presses, etc.).
- virtual void [eventPollEvent](#) ()=0
Polls for any pending events on the window object used in the implementation. Its purpose is to poll for any pending events on the window object used in the implementation. This function is typically called by the application's main loop to handle user input or other events.
- virtual std::unique_ptr< [IText](#) > [createIText](#) ()=0
Creates a new instance of the [IText](#) interface for the implementation. Its purpose is to create a new instance of the [IText](#) interface for the implementation. The returned object can be used to draw text on the window.
- virtual std::unique_ptr< [IGrid](#) > [createIGrid](#) ()=0
Creates a new instance of the [IGrid](#) interface for the implementation. Its purpose is to create a new instance of the [IGrid](#) interface for the implementation. The returned object can be used to draw a grid on the window.

7.8.1 Member Typedef Documentation

7.8.1.1 EventCallback

```
using IWindow::EventCallback = std::function<void()>
```

7.8.1.2 EventHandler

```
using IWindow::EventHandler = std::map<IWindow::EventType, EventCallback>
```

7.8.2 Member Enumeration Documentation

7.8.2.1 EventType

```
enum class IWindow::EventType [strong]
```

Enumerator

UP_pressed	
DOWN_pressed	
LEFT_pressed	
RIGHT_pressed	
QUIT	
PAUSE	
RESUME	
NEXT_GAME	
NEXT_LIB	
RESTART	
GO_TO_MENU	

7.8.3 Constructor & Destructor Documentation

7.8.3.1 ~IWindow()

```
IWindow::~IWindow ( ) [default]
```

7.8.4 Member Function Documentation

7.8.4.1 callEvent()

```
virtual void IWindow::callEvent (
    const IWindow::EventType ) [pure virtual]
```

Calls the event handler function for the specified event type. Its purpose is to call the event handler function for the specified event type. This function is typically called by the application's main loop to handle user input or other events.

Parameters

<i>eventType</i>	An object of type IWindow::EventType specifying the type of the event to be handled.
------------------	--

7.8.4.2 clear()

```
virtual void IWindow::clear ( ) [pure virtual]
```

Clears the rendering buffer used in the implementation. Its purpose is to clear the rendering buffer used in the implementation. This is typically done at the beginning of each frame to ensure that the previous frame's rendering is not visible in the current frame.

7.8.4.3 closeWindow()

```
virtual void IWindow::closeWindow ( ) [pure virtual]
```

loses the window object used in the implementation. Its purpose is to close and destroy the window object used in the implementation.

7.8.4.4 createIGrid()

```
virtual std::unique_ptr< IGrid > IWindow::createIGrid ( ) [pure virtual]
```

Creates a new instance of the [IGrid](#) interface for the implementation. Its purpose is to create a new instance of the [IGrid](#) interface for the implementation. The returned object can be used to draw a grid on the window.

Returns

`std::unique_ptr<IGrid>`

7.8.4.5 createIText()

```
virtual std::unique_ptr< IText > IWindow::createIText ( ) [pure virtual]
```

Creates a new instance of the [IText](#) interface for the implementation. Its purpose is to create a new instance of the [IText](#) interface for the implementation. The returned object can be used to draw text on the window.

Returns

`std::unique_ptr<IText>`

7.8.4.6 display()

```
virtual void IWindow::display ( ) [pure virtual]
```

Displays the rendered content on the window object used in the implementation. Its purpose is to display the rendered content on the window object used in the implementation. This is typically done at the end of each frame to show the newly rendered content on the screen.

7.8.4.7 eventPollEvent()

```
virtual void IWindow::eventPollEvent ( ) [pure virtual]
```

Polls for any pending events on the window object used in the implementation. Its purpose is to poll for any pending events on the window object used in the implementation. This function is typically called by the application's main loop to handle user input or other events.

7.8.4.8 initWindow()

```
virtual void IWindow::initWindow (
    std::string name,
    size_t width,
    size_t height ) [pure virtual]
```

Initializes the window object required for the implementation. Its purpose is to initialize the window object with the specified name, width, and height.

Parameters

<i>name</i>	
<i>width</i>	
<i>height</i>	

7.8.4.9 loadEventBindings()

```
virtual void IWindow::loadEventBindings (
    EventHandler & ) [pure virtual]
```

loads the event bindings for the specified event handler object. Its purpose is to load the event bindings for the specified event handler object. This function is typically called at the beginning of the application to register event handlers for various types of events (e.g. mouse clicks, key presses, etc.).

Parameters

<i>handler</i>	An object of type EventHandler that contains the event bindings to be loaded.
----------------	---

7.8.4.10 windowIsOpen()

```
virtual bool IWindow::windowIsOpen ( ) [pure virtual]
```

Checks whether the window object used in the implementation is currently open. Its purpose is to check whether the window object used in the implementation is currently open.

Returns

true
false

The documentation for this class was generated from the following file:

- Interface/Graphical/[IWindow.hpp](#)

Chapter 8

File Documentation

8.1 Interface/Game/IGame.hpp File Reference

```
#include "../Graphical/IGraphicalFactory.hpp"
```

Classes

- class [IGame](#)

8.2 IGame.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** B-OOP-400-BDX-4-1-arcade-leopold.sallan-gemard
00004 ** File description:
00005 ** IGame
00006 */
00007
00008 #pragma once
00009 #include "../Graphical/IGraphicalFactory.hpp"
00010
00011 class IGame
00012 {
00013     public:
00014         ~IGame() = default;
00015
00016         virtual IWindow::EventHandler &getEventBinding() = 0;
00017
00018         virtual bool processGameTick(IGrid &grid, IText &scoreText, IText &timeText,
00019                                     IClock &clock) = 0;
00020
00021         virtual void restart() = 0;
00022 };
```

8.3 Interface/Graphical/IClock.hpp File Reference

Classes

- class [IClock](#)

8.4 IClock.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** arcade
00004 ** File description:
00005 ** IClock
00006 */
00007
00008 #pragma once
00009
00010 class IClock
00011 {
00012     public:
00013         ~IClock() = default;
00014
00015         virtual void startClock() = 0;
00016
00017         virtual double getTimeElapsed() = 0;
00018
00019         virtual void resetClock() = 0;
00020
00021         virtual void initClock() = 0;
00022 };
```

8.5 Interface/Graphical/IEntity.hpp File Reference

```
#include <string>
```

Classes

- class [IEntity](#)
- class [IText](#)
- class [IGrid](#)
- struct [IGrid::Cell](#)

8.6 IEntity.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2023
00003 ** arcade
00004 ** File description:
00005 ** IEntity
00006 */
00007
00008 #pragma once
00009 #include <string>
00010
00011 class IEntity
00012 {
00013     public:
00014         enum class Color {
00015             Blue,
00016             Green,
00017             Orange,
00018             Red,
00019             Brown,
00020             Yellow,
00021         };
00022
00023         ~IEntity() = default;
00024
00025         virtual void setPosition(int x, int y) = 0;
00026 }
```

```

00040         virtual void destroy() = 0;
00041
00046         virtual void displayEntity() = 0;
00047     };
00048
00049     class IText : public IEntity
00050     {
00051     public:
00052         enum class EntityType {
00053             GridEntity,
00054             SpriteEntity,
00055             TextEntity,
00056         };
00057         ~IText() = default;
00058
00067         virtual void create(std::string str) = 0;
00068
00076         virtual void changeString(std::string str) = 0;
00077     };
00078
00079     class IGrid : public IEntity
00080     {
00081     public:
00082         struct Cell {
00083             Color color;
00084             size_t x;
00085             size_t y;
00086         };
00087         ~IGrid() = default;
00088
00098         virtual void create(int width, int height) = 0;
00099
00110         virtual void updateCell(int x, int y, IEntity::Color) = 0;
00111     };

```

8.7 Interface/Graphical/IGraphicalFactory.hpp File Reference

```

#include "IClock.hpp"
#include "IEntity.hpp"
#include "IWindow.hpp"
#include <memory>

```

Classes

- class [IGraphicalFactory](#)

8.8 IGraphicalFactory.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2023
00003  ** B-OOP-400-BDX-4-1-arcade-leopold.sallan-gemard
00004  ** File description:
00005  ** IGraphicalFactorys
00006  */
00007
00008  #pragma once
00009  #include "IClock.hpp"
00010  #include "IEntity.hpp"
00011  #include "IWindow.hpp"
00012  #include <memory>
00013
00014  class IGraphicalFactory
00015  {
00016  public:
00017       ~IGraphicalFactory() = default;
00018
00024       virtual void loadResource() = 0;

```

```

00025
00034     virtual void destroyRessource() = 0;
00035
00044     virtual std::unique_ptr<IWindow> createWindow(std::string name, size_t width,
00045                                                    size_t height) = 0;
00046
00056     virtual std::unique_ptr<IClock> createIClock() = 0;
00057 };

```

8.9 Interface/Graphical/IGrid.md File Reference

8.10 Interface/Graphical/IText.md File Reference

8.11 Interface/Graphical/IWindow.hpp File Reference

```

#include "IEntity.hpp"
#include <functional>
#include <map>
#include <memory>

```

Classes

- class [IWindow](#)

8.12 IWindow.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  ** EPITECH PROJECT, 2023
00003  ** arcade
00004  ** File description:
00005  ** IWindow
00006  */
00007
00008 #pragma once
00009 #include "IEntity.hpp"
00010 #include <functional>
00011 #include <map>
00012 #include <memory>
00013
00014 class IWindow
00015 {
00016     public:
00017         ~IWindow() = default;
00018
00027         virtual void initWindow(std::string name, size_t width, size_t height) = 0;
00028
00033         virtual void closeWindow() = 0;
00034
00042         virtual bool windowIsOpen() = 0;
00043
00050         virtual void clear() = 0;
00051
00058         virtual void display() = 0;
00059
00068         virtual void callEvent(const IWindow::EventType) = 0;
00069
00078         virtual void loadEventBindings(EventHandler &) = 0;
00079
00086         virtual void eventPollEvent() = 0;
00087
00094         virtual std::unique_ptr<IText> createIText() = 0;

```



```
00095
00102     virtual std::unique_ptr<IGrid> createIGrid() = 0;
00103
00104     enum class EventType {
00105         UP_pressed,
00106         DOWN_pressed,
00107         LEFT_pressed,
00108         RIGHT_pressed,
00109         QUIT,
00110         PAUSE,
00111         RESUME,
00112         NEXT_GAME,
00113         NEXT_LIB,
00114         RESTART,
00115         GO_TO_MENU,
00116     };
00117
00118     using EventCallback = std::function<void()>;
00119     using EventHandler = std::map<IWindow::EventType, EventCallback>;
00120 };
```

8.13 README.md File Reference

Index

- ~IClock
 - IClock, [14](#)
- ~IEntity
 - IEntity, [16](#)
- ~IGame
 - IGame, [18](#)
- ~IGraphicalFactory
 - IGraphicalFactory, [19](#)
- ~IGrid
 - IGrid, [23](#)
- ~IText
 - IText, [25](#)
- ~IWindow
 - IWindow, [27](#)
- Blue
 - IEntity, [16](#)
- Brown
 - IEntity, [16](#)
- callEvent
 - IWindow, [28](#)
- changeString
 - IText, [25](#)
- clear
 - IWindow, [28](#)
- closeWindow
 - IWindow, [28](#)
- Color
 - IEntity, [16](#)
- color
 - IGrid::Cell, [13](#)
- create
 - IGrid, [23](#)
 - IText, [25](#)
- createIClock
 - IGraphicalFactory, [20](#)
- createIGrid
 - IWindow, [28](#)
- createIText
 - IWindow, [28](#)
- createWindow
 - IGraphicalFactory, [21](#)
- destroy
 - IEntity, [16](#)
- destroyRessource
 - IGraphicalFactory, [21](#)
- display
 - IWindow, [29](#)
- displayEntity
 - IEntity, [16](#)
- DOWN_pressed
 - IWindow, [27](#)
- EntityType
 - IText, [24](#)
- EventCallBack
 - IWindow, [27](#)
- EventHandler
 - IWindow, [27](#)
- eventPollEvent
 - IWindow, [29](#)
- EventType
 - IWindow, [27](#)
- getEventBinding
 - IGame, [18](#)
- getTimeElapsed
 - IClock, [14](#)
- GO_TO_MENU
 - IWindow, [27](#)
- Green
 - IEntity, [16](#)
- GridEntity
 - IText, [25](#)
- IClock, [14](#)
 - ~IClock, [14](#)
 - getTimeElapsed, [14](#)
 - initClock, [14](#)
 - resetClock, [15](#)
 - startClock, [15](#)
- IEntity, [15](#)
 - ~IEntity, [16](#)
 - Blue, [16](#)
 - Brown, [16](#)
 - Color, [16](#)
 - destroy, [16](#)
 - displayEntity, [16](#)
 - Green, [16](#)
 - Orange, [16](#)
 - Red, [16](#)
 - setPosition, [17](#)
 - Yellow, [16](#)
- IGame, [17](#)
 - ~IGame, [18](#)
 - getEventBinding, [18](#)
 - processGameTick, [18](#)
 - restart, [19](#)

- IGraphicalFactory, 19
 - ~IGraphicalFactory, 19
 - createIClock, 20
 - createWindow, 21
 - destroyRessource, 21
 - loadResource, 21
- IGrid, 22
 - ~IGrid, 23
 - create, 23
 - updateCell, 23
- IGrid::Cell, 13
 - color, 13
 - x, 13
 - y, 13
- initClock
 - IClock, 14
- initWindow
 - IWindow, 29
- Interface/Game/IGame.hpp, 31
- Interface/Graphical/IClock.hpp, 31, 32
- Interface/Graphical/IEntity.hpp, 32
- Interface/Graphical/IGraphicalFactory.hpp, 33
- Interface/Graphical/IGrid.md, 34
- Interface/Graphical/IText.md, 34
- Interface/Graphical/IWindow.hpp, 34
- IText, 24
 - ~IText, 25
 - changeString, 25
 - create, 25
 - EntityType, 24
 - GridEntity, 25
 - SpriteEntity, 25
 - TextEntity, 25
- IWindow, 26
 - ~IWindow, 27
 - callEvent, 28
 - clear, 28
 - closeWindow, 28
 - createIGrid, 28
 - createIText, 28
 - display, 29
 - DOWN_pressed, 27
 - EventCallBack, 27
 - EventHandler, 27
 - eventPollEvent, 29
 - EventType, 27
 - GO_TO_MENU, 27
 - initWindow, 29
 - LEFT_pressed, 27
 - loadEventBindings, 30
 - NEXT_GAME, 27
 - NEXT_LIB, 27
 - PAUSE, 27
 - QUIT, 27
 - RESTART, 27
 - RESUME, 27
 - RIGHT_pressed, 27
 - UP_pressed, 27
- windowIsOpen, 30
- LEFT_pressed
 - IWindow, 27
- loadEventBindings
 - IWindow, 30
- loadResource
 - IGraphicalFactory, 21
- NEXT_GAME
 - IWindow, 27
- NEXT_LIB
 - IWindow, 27
- Orange
 - IEntity, 16
- PAUSE
 - IWindow, 27
- processGameTick
 - IGame, 18
- QUIT
 - IWindow, 27
- README.md, 35
- Red
 - IEntity, 16
- resetClock
 - IClock, 15
- RESTART
 - IWindow, 27
- restart
 - IGame, 19
- RESUME
 - IWindow, 27
- RIGHT_pressed
 - IWindow, 27
- setPosition
 - IEntity, 17
- SpriteEntity
 - IText, 25
- startClock
 - IClock, 15
- TextEntity
 - IText, 25
- UP_pressed
 - IWindow, 27
- updateCell
 - IGrid, 23
- windowIsOpen
 - IWindow, 30
- x
 - IGrid::Cell, 13
- y

IGrid::Cell, [13](#)
Yellow
 IEntity, [16](#)