jun 12, 18 14:03 **GirderSize.cpp** Page 1/1

```cpp
#include "GirderSize.h"
#include "Math.h"
#include "ObjectSizes.h"

float GirderSize::getGirderWidthMeters(int size, int angle){
    angle = GirderSize::normalizeAngle(angle);
    return Math::cosDegrees(angle) * size +
                Math::sinDegrees(angle) * girder_height;
}

int GirderSize::getGirderWidthPixels(int size, int angle){
    return SCALE_FACTOR * GirderSize::getGirderWidthMeters(size, angle);
}

float GirderSize::getGirderHeightMeters(int size, int angle){
    angle = GirderSize::normalizeAngle(angle);
    return Math::sinDegrees(angle) * size +
                Math::cosDegrees(angle) * girder_height;
}

int GirderSize::getGirderHeightPixels(int size, int angle){
    return SCALE_FACTOR * GirderSize::getGirderHeightMeters(size, angle);
}

int GirderSize::normalizeAngle(int angle){
    return angle > 90 ? 180 - angle : angle;
}
```

jun 04, 18 21:12 **GirderSize.h** Page 1/1

```cpp
#ifndef __GIRDERSIZE_H__
#define __GIRDERSIZE_H__

class GirderSize{
    private:
        //Normaliza el angulo entre 0 y 90
        static int normalizeAngle(int angle);
    public:
        //Devuelve el ancho de una viga en metros
        static float getGirderWidthMeters(int size, int angle);

        //Devuelve el ancho de una viga en pixeles
        static int getGirderWidthPixels(int size, int angle);

        //Devuelve el alto de una viga en metros
        static float getGirderHeightMeters(int size, int angle);

        //Devuelve el alto de una viga en pixeles
        static int getGirderHeightPixels(int size, int angle);
};

#endif
```

jun 12, 18 14:03 **Position.cpp** Page 1/1

```cpp
#include "Position.h"
#include <cmath>

#define FACTOR 100

Position::Position(float x, float y): x(x), y(y){}

Position::~Position(){}

bool Position::operator==(const Position& other){
    return (int)(this->x * FACTOR) == (int)(other.x * FACTOR) &&
                    (int)(this->y * FACTOR) == (int)(other.y * FACTOR);
}

float Position::getX() const{
    return this->x;
}

float Position::getY() const{
    return this->y;
}
```

jun 12, 18 14:03 **Position.h** Page 1/1

```cpp
#ifndef __POSITION_H__
#define __POSITION_H__

/* Clase que se encarga de representar posiciones en el plano */
class Position{
    private:
        float x;
        float y;

    public:
        /* Constructor */
        Position(float x, float y);

        /* Destructor */
        ~Position();

        /* Devuelve true si las dos posiciones son iguales */
        bool operator==(const Position& other);

        /* Devuelve el valor en X de la posicion */
        float getX() const;

        /* Devuelve el valor en Y de la posicion */
        float getY() const;
};

#endif
```

```cpp
#include "ScrollHandler.h"
#include <gtkmm/adjustment.h>
#include <glibmm/main.h>

#define SPACE_TO_SCROLL 20
#define SCROLL_INCREMENT 25
#define TIMEOUT 50
#define WAIT_TO_SCROLL TIMEOUT * 10
const Position NO_SCROLL_POSITION(SPACE_TO_SCROLL * 2, SPACE_TO_SCROLL * 2);

ScrollHandler::ScrollHandler(Gtk::ScrolledWindow& window):
        window(window),
        last_mouse_position(NO_SCROLL_POSITION),
        mouse_in_window(false){
    this->current_time = 0;
    this->window.add_events(Gdk::POINTER_MOTION_MASK);
    this->window.add_events(Gdk::ENTER_NOTIFY_MASK);
    this->window.add_events(Gdk::ENTER_NOTIFY_MASK);
    this->window.signal_motion_notify_event().connect(
                                        sigc::mem_fun(*this, &ScrollHandler::mouseMotionEvent));
    this->window.set_policy(Gtk::POLICY_NEVER, Gtk::POLICY_NEVER);

    this->window.signal_enter_notify_event().connect(
                                        sigc::mem_fun(*this, &ScrollHandler::mouseEntered));
    this->window.signal_leave_notify_event().connect(
                                        sigc::mem_fun(*this, &ScrollHandler::mouseLeft));
    this->my_connection = Glib::signal_timeout().connect(
                                        sigc::mem_fun(*this, &ScrollHandler::scroll), TIMEOUT);
}

ScrollHandler::~ScrollHandler(){}

bool ScrollHandler::mouseMotionEvent(GdkEventMotion* motion_event){
    this->last_mouse_position = Position(motion_event->x, motion_event->y);
    this->mouse_in_window = true;
    return true;
}

bool ScrollHandler::mouseEntered(GdkEventCrossing* crossing_event){
    this->mouse_in_window = true;
    this->current_time = 0;
    this->last_mouse_position = NO_SCROLL_POSITION;
    return true;
}

bool ScrollHandler::mouseLeft(GdkEventCrossing* crossing_event){
    this->mouse_in_window = false;
    this->current_time = 0;
    return true;
}

bool ScrollHandler::scroll(){
    int window_width = window.get_hadjustment()->get_page_size();
    int window_height = window.get_vadjustment()->get_page_size();

    if (!this->mouse_in_window){
        //El mouse esta fuera de la pantalla
        this->current_time = 0;
        return true;
    }

    if (this->current_time < WAIT_TO_SCROLL){
```



```cpp
        this->current_time += TIMEOUT;
        return true;
    }

    int scrolled = 0;
    if (last_mouse_position.getX() < SPACE_TO_SCROLL){
        //Scroll a la izquierda
        this->window.get_hadjustment()->set_value(
                        this->window.get_hadjustment()->get_value() - SCROLL_INCREMENT);
        scrolled++;
    }

    if (last_mouse_position.getX() > window_width - SPACE_TO_SCROLL){
        //Scroll a la derecha
        this->window.get_hadjustment()->set_value(
                        this->window.get_hadjustment()->get_value() + SCROLL_INCREMENT);
        scrolled++;
    }

    if (last_mouse_position.getY() < SPACE_TO_SCROLL){
        //Scroll arriba
        this->window.get_vadjustment()->set_value(
                        this->window.get_vadjustment()->get_value() - SCROLL_INCREMENT);
        scrolled++;
    }

    if (last_mouse_position.getY() > window_height - SPACE_TO_SCROLL){
        //Scroll abajo
        this->window.get_vadjustment()->set_value(
                        this->window.get_vadjustment()->get_value() + SCROLL_INCREMENT);
        scrolled++;
    }

    if (!scrolled){
        this->current_time = 0;
    }
    return true;
}

void ScrollHandler::stop(){
    if (this->my_connection.connected()) {
        this->my_connection.disconnect();
    }
}
```

jun 12, 18 19:03 **ScrollHandler.h** Page 1/1

```
#ifndef __SCROLLHADNLER_H__
#define __SCROLLHADNLER_H__

#include <gtkmm/scrolledwindow.h>
#include <gdk/gdk.h>
#include "Position.h"

class ScrollHandler{
    private:
        Gtk::ScrolledWindow& window;
        Position last_mouse_position;
        bool mouse_in_window;
        sigc::connection my_connection;
        int current_time;

        /* Handler del movimiento del mouse */
        bool mouseMotionEvent(GdkEventMotion* motion_event);

        /* Handler de entrada en el area de desplazamiento */
        bool mouseEntered(GdkEventCrossing* crossing_event);

        /* Handler de salida del area de desplazamiento */
        bool mouseLeft(GdkEventCrossing* crossing_event);

        /* Realiza el desplazamiento de la pantalla */
        bool scroll();

    public:
        /* Constructor */
        explicit ScrollHandler(Gtk::ScrolledWindow& window);

        /* Destructor */
        ~ScrollHandler();

        /* Detiene el desplazamiento */
        void stop();
};

#endif
```

jun 12, 18 14:03 **ViewPositionTransformer.cpp** Page 1/1

```
#include "ViewPositionTransformer.h"
#include "ObjectSizes.h"

ViewPositionTransformer::ViewPositionTransformer(Gtk::Layout& layout):
    layout(layout){}

ViewPositionTransformer::~ViewPositionTransformer() {}

Position ViewPositionTransformer::transformToScreen(const Position& position){
    guint width, height;
    this->layout.get_size(width, height);
    float x = SCALE_FACTOR * position.getX();
    float y = height - SCALE_FACTOR * position.getY();
    return Position(x, y);
}

Position ViewPositionTransformer::transformToScreenAndMove(
                            const Position& position, float width, float height)
{
    Position pos = this->transformToScreen(position);
    Position moved(pos.getX() - SCALE_FACTOR * width / 2,
                            pos.getY() - SCALE_FACTOR * height / 2);
    return moved;
}

Position ViewPositionTransformer::transformToPosition(const Position& position){
    guint width, height;
    this->layout.get_size(width, height);
    float x = position.getX() / SCALE_FACTOR;
    float y = (height - position.getY()) / (SCALE_FACTOR);
    return Position(x, y);
}
```

jun 12, 18 14:03 **ViewPositionTransformer.h** Page 1/1

```h
#ifndef __VIEWTRANSFORMER_H__
#define __VIEWTRANSFORMER_H__

#include <gtkmm/layout.h>
#include "Position.h"

/* Clase que se encarga de transformar posiciones de la pantalla
 * en posiciones en metros */
class ViewPositionTransformer{
    private:
        Gtk::Layout& layout;

    public:
        /* Constructor */
        explicit ViewPositionTransformer(Gtk::Layout& layout);

        /* Destructor */
        ~ViewPositionTransformer();

        /* Dada una posicion en metros, devuelve una posicion en
         * pixeles que representa una posicion de la pantalla*/
        Position transformToScreen(const Position& position);

        /* Dada una posicion en metros, la transforma en una posicion
         * para la pantalla y la desplaza segun su ancho y alto */
        Position transformToScreenAndMove(const Position& pos, float w, float h)
;

        /* Dada una posicion en pixeles, devuelve una posicion en metros */
        Position transformToPosition(const Position& position);
};

#endif
```

may 31, 18 21:43 **Water.cpp** Page 1/1

```cpp
#include "Water.h"
#include "Path.h"
#include "ObjectSizes.h"

Water::Water(){}

Water::~Water(){}

void Water::show(Gtk::Layout& layout){
    this->images.clear();

    size_t pos = 0;
    guint width, height;
    layout.get_size(width, height);

    while(pos < width){
        Gtk::Image image;
        image.set(IMAGES_PATH + "Water.png");
        this->images.push_back(std::move(image));
        layout.put(this->images.back(), pos, height - water_height);
        this->images.back().show();
        pos += water_length;
    }
}
```

jun 12, 18 14:03 **Water.h** Page 1/1

```
1  #ifndef __WATER_H__
2  #define __WATER_H__
3
4  #include <gtkmm/image.h>
5  #include <gtkmm/layout.h>
6  #include <vector>
7
8  /* Clase que se encarga de controlar la vista del agua */
9  class Water{
10     private:
11         std::vector<Gtk::Image> images;
12
13     public:
14         /* Constructor */
15         Water();
16
17         /* Destructor */
18         ~Water();
19
20         /* Muestra la imagen del agua */
21         void show(Gtk::Layout& layout);
22 };
23
24 #endif
```

jun 26, 18 14:00 **Table of Content** Page 1/1