|  |  |
| --- | --- |
| https://studfiles.net/html/2706/219/html_4NnFGVyFmL.LWVf/img-KuTuVC.png | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-01 |

Факультет И Информационные и управляющие системы

шифр наименование

Кафедра И5 Информационные системы и программные технологии

шифр наименование

Дисциплина Программирование на языке высокого уровня

КУРСОВАЯ РАБОТА

|  |
| --- |
| Создание GUI-библиотеки |
| на базе библиотеки SDL2 |
|  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | | | | *И-582* |
| *Махнев П.С.* | | | | | | |
| Фамилия И.О. | | | | | | |
|  | | | | | | |
| **РУКОВОДИТЕЛЬ** | | | | | | |
|  | | | | | | |
| *Спирин Д.О.* | | | |  |  | |
| Фамилия И.О. | | | |  | Подпись | |
|  | | | | | | |
| Оценка | | |  | | |  |
| « |  | » |  | | | 2019 г. |

САНКТ-ПЕТЕРБУРГ

2019 г.

# ВВЕДЕНИЕ

Целью данной работы является создание функциональной и простой в использовании GUI-библиотеки на базе библиотеки SDL2, включая библиотеки SDL\_ttf, SDL\_image и SDL\_gfx.

Изначально работа была разбита на следующие задачи:

1. Разработка ТЗ;
2. Разработка требований к исходной программе, которая будет отражать возможности итоговой библиотеки;
3. Разработка требований к документации;

# Техническое задание

Разработать GUI-библиотеку (GUI – графический пользовательский интерфейс) с возможность создавать пользовательские окна, с настраиваемым макетом из элементов графического интерфейса. Библиотека должна предоставлять возможность создания своих элементов на базе общего класса.

Также необходимо обеспечить простой путь наследования для создания новых элементов на базе класса Компонент. Элементы должны иметь возможность отслеживать следующие события:

1. Нажатие кнопки мыши;
2. Отпускание кнопки мыши;
3. Движение курсора мыши;
4. Попадание курсора мыши на элемент;
5. Выход курсора мыши из элемента;
6. Движение колесика мыши;
7. Клик по элементу;

Библиотека должна предоставлять простой путь создания собственных классов окон с настраиваемым макетом на основе базового класса окна.

Окно должно уметь обрабатывать следующие события:

1. Нажатие кнопки мыши;
2. Отпускание кнопки мыши;
3. Движение мыши;
4. Нажатие клавиши на клавиатуре;
5. Отпускание клавиши на клавиатуре;

# Итоговая реализация

При разработке требований к библиотеке, встал очень важный вопрос. А именно что использовать для стилизации элементов? Есть два возможных варианта. Один из них заключается в прописывании стилей в исходном коде, рядом с созданием самого элемента. Это хороший подход, если это код генерируется, но если его вводит человек, то возникает ряд проблем. Одна из проблем, это переиспользуемость кода, например, кнопки имеют одинаковые стили, но для каждой из них нужно будет копировать один и тот-же кусок кода, что в итоге приведет к печальным последствиям. Еще одна проблема, это «захламленность» кода, когда в коде «перемешиваются» создание элементов и их стилизация, сложно сразу понять что и где.

Второй вариант, это использование стилевых файлов, файлов с прописанными стилями для каждого из элементов. Здесь самым известным представителем является язык стилей CSS. Его, и было принято решение использовать, как стилевой. Однако стандарт CSS охватывает невероятное количество возможностей, которые нам не нужны, так что было принято решение взять только нужное, для текущего проекта. Были взяты следующие свойства:

1. "background-color" — цвет фона;
2. "border-color" — цвет обводки;
3. "color" — цвет текста;
4. "background-image" — фоновое изображение;
5. "background-position-x" — сдвиг изображения по X;
6. "background-position-y" — сдвиг изображения по Y;
7. "background-size" — размер изображения по ширине;
8. "font-size" — размер шрифта;
9. "line-height" — междустрочный интервал;
10. "text-align" — выравнивание текста по горизонтали;
11. "vertical-align" — выравнивание текста по вертикали;
12. "margin-top" — сдвиг текста сверху;
13. "margin-bottom" — сдвиг текста снизу;
14. "margin-left" — сдвиг текста слева;
15. "margin-right" — сдвиг текста справа;
16. "border-radius" — радиус закругления обводки;
17. "border-top" — обводка сверху;
18. "border-top-size" — ширина обводки сверху;
19. "border-top-color" — цвет обводки сверху;
20. "border-top-type" — тип обводки сверху;
21. "border-bottom" — обводка снизу;
22. "border-bottom-size" — ширина обводки снизу;
23. "border-bottom-color" — цвет обводки снизу;
24. "border-bottom-type" — тип обводки снизу;
25. "border-left" — обводка слева;
26. "border-left-size" — ширина обводки слева;
27. "border-left-color" — цвет обводки слева;
28. "border-left-type" — тип обводки слева;
29. "border-right" — обводка справа;
30. "border-right-size" — ширина обводки справа;
31. "border-right-color" — цвет обводки справа;
32. "border-right-type" — тип обводки справа.

А так же были взяты 2 псевдокласса, Hover и Active. Hover отвечает за стили, когда на элемент наведена мышь, а Active, когда мышь зажата на элементе.

Парсер построен на простом конечном автомате. Скорость загрузки файлов и парсинга – хорошие.

Следующем этапом было разработка основного класса Component пригодного для построения на нем расширяемой системы. Было решено, что Component должен поддерживать любое из вышеперечисленных свойств, для максимальной гибкости. Класс Component реализует не только внешний вид, но и гибкую настройку логику. Так любому объекту класса Component можно привязать до 7 событий:

1. "click" — событие происходящее при клике на компонент;
2. "hover" — событие происходящее при наведении на компонент;
3. "mousemotion" — событие происходящее при перемещении курсора мыши по компоненту;
4. "onmousedown" — событие происходящее при нажатии на компонент
5. "onmouseup" — событие происходящее при отпускании кнопки мыши на компоненте;
6. "onmouseover" — событие происходящее при входе курсора мыши в компонент;
7. "onmouseout" — событие происходящее при выходе курсора мыши из компонента.

Класс Component реализует в себе и иерархическую часть, так любой объект хранит всех своих детей и имеет доступ к каждому из них. Вместе с этим, каждый объект хранит в себе и родителя, что позволяет, при, надобности, попасть куда угодно из любого компонента.

Следующим шагом после разработки класса Component, стал вопрос разработки окон и интеграция с ними всей логики Component. Одним из важнейших требований, стало то, что любой компонент можно было получить с помощью одной функции. Однако встал вопрос хранения всех компонентов в окне. И для решения этой задачи было принято решение использовать строковые идентификаторы и ассоциативный массив с ключом идентификатором. Данный подход упрощает работу с компонентами и доступ к ним осуществляется по идентификатору. Так же было принято решение, что не может быть двух элементов в окне с одинаковым идентификатором, иначе возникает коллизия. Любой объект с повторяющимся идентификатором не добавляется, а в консоль будет выведено предупреждение.

Каждый окно содержит в себе особый вид компонента под название Навигатор. В окне, доступ к нему осуществляется либо через переменную navigator, либо через переменную $$. Данный компонент является главным в окне, и все компоненты окна являются его потомками.

После создания класса окна, встал вопрос, а как сделать работу с библиотекой максимально удобной. Было принято решение, что при подключении библиотеки, пользователь сразу может пользоваться экземпляром приложения в функции main без создания своего. Ему было подобрано краткое лаконичное имя «$». А так же для запуска всего приложения была добавлена функция run, которую необходимо вызвать перед return 0; для запуска приложения. До запуска приложения, пользователь имеет возможность добавлять новые окна с помощью функции addWindow, которая принимает указатель на новое окно. Создание классов окна, это отдельная тема.

Рассмотрим пример минимального приложения

**Файл main.cpp**

#include "kit/kit.h"

#include "MyWindowClass/MyWindow.h"

using namespace Lib;

int main(int argc, char\*\* argv)

{

$.addWindow(new MyWindow("new window", { 100, 100, 1000, 500 }));

$.run();

return 0;

}

**Файл MyWindow.h**

#pragma once

#include "../kit/window/window.h"

using namespace Lib;

class MyWindow : public Window

{

public:

MyWindow(string title, SimpleRect size)

: Window(title, size) {

setup();

};

public:

void setup() {

}

};

Данный пример откроет одно пустое окно размерами 1000 на 500 пикселей.

Как видно, из примера, класс MyWindow унаследован от базового класса Window. При наследовании, достаточно перегрузить только конструктор и функцию setup(). Функция Setup используется для настройки компонентов данного класса окна.

Каждое окно имеет следующий интерфейс:

* Component\* addElement(Component\* component) — добавление нового компонента в окно;
* Component\* getElementById(string id) const — получение объекта по его идентификатору;
* Components getElementsByClassName(string className) const — получение набора элементов, имеющие переданный в параметр функции класс;
* CSS::css\_block\* addStyle(string className, CSS::css\_block style) — добавление новой зависимости класса идентификатора и блока CSS стилей;
* void show() — функция показывающая окно;
* void hide() — функция скрывающая окно;
* bool isShow() — функция возвращающая текущее состояние, отображается ли окно или нет;
* void collapse() — функция сворачивающая окно;
* void close() — функция закрывающая окно;
* void setDraggableArea(SimpleRect area) — функция для установки заголовка за который это окно может быть перетаскиваемо (используется для окон без рамки);
* void include(string path) — основная функция для включения стилей в окне;

Рассмотрим создание собственного компонента на базе класса Component. Для этого достаточно перегрузить конструктор и создать функцию setup() для настройки.

#pragma once

#include "../kit/component/component.h"

using namespace Lib;

class Button : public Component

{

private:

string text;

public:

Button(string id, Rect size, string classes, string text)

: Component(id, size, classes) {

this->text = text;

setup();

};

public:

void setup()

{

setText(text);

};

};

Данный класс кнопки инкапсулирует в себе часть логики, ту же кнопку можно создать просто создав объект класса Component и вызвав у него функцию setText(). Но в данном случае мы получаем более удобную версию.

Компоненты созданные на базе базового класса так же могут настроить своих потомков самостоятельно. То есть компонент будет не единичным объектом, а некоторой связной системой из нескольких блоков. При этом данный компонент можно будет добавлять несколько раз, что уменьшает копирование кода.

# Заключение

В результате проделанной работы получилась небольшая библиотека, которая полностью удовлетворяет заданию и даже расширяет его. Библиотека корректно работает, не вылетает и не создает утечек памяти.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Б.И. Березин. Начальный курс С и С++. – М.:Издательство Диалог-МИФИ, 2005 г. – 248 с.
2. Р. Лафоре. Объектно-ориентированное программирование в С++. 4-е издание. – Спб.: Издательство ПИТЕР, 2004 г. – 902 с.
3. Б. Страуструп. Язык программирования С++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2011 г. – 1136 с.
4. Лафоре, Р. Объектно-ориентированное программирование в C++: Пер. с англ./ Р. Лафоре; Пер. А. Кузнецов, Пер. М. Назаров, Пер. В. Шрага. - 4-е изд. - СПб.: Питер, 2003. - 923 с.
5. Официальный сайт графической библиотеки SDL [Электронный ресурс] 2019. URL: <https://www.libsdl.org> (Дата обращения 13.12.2019)