Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 7: «Проектирование структуры классов»

|  |  |
| --- | --- |
| Группа: | М8О-108Б-18, №6 |
| Студент: | Васильева Василиса Евгеньевна |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 09.12.2019 |

Москва, 2019

1. **Задание**

Спроектировать простейший графический векторный редактор. Требование к функционалу редактора:

• создание нового документа

• импорт документа из файла

• экспорт документа в файл

• создание графического примитива (согласно варианту задания)

• удаление графического примитива

• отображение документа на экране (печать перечня графических объектов и их характеристик)

• реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

• Создание графических примитивов необходимо вынести в отдельный класс – Factory.

• Сделать упор на использовании полиморфизма при работе с фигурами;

• Взаимодействие с пользователем (ввод команд) реализовать в функции main;

1. **Адрес репозитория на GitHub**

https://github.com/vasilisavasileva/oop-07

1. **Код программы на С++**

Sdl.h

#ifndef D\_SDL\_H\_

#define D\_SDL\_H\_ 1

#include <string>

#include "SDL\_events.h"

#include "SDL\_render.h"

#include "SDL\_video.h"

namespace sdl {

struct sdl {

sdl();

~sdl();

};

struct renderer {

renderer(const std::string& window\_name);

~renderer();

// set color for subsequent operations

void set\_color(uint8\_t r, uint8\_t g, uint8\_t b) const;

// fill screen with current color

void clear() const;

// draw segment with current color

void draw\_line(int32\_t x1, int32\_t y1, int32\_t x2, int32\_t y2) const;

// every command draws to a temporary buffer

// this function swaps temporary buffer containing new frame with current frame

void present() const;

private:

sdl system;

SDL\_Window\* window\_;

SDL\_Renderer\* renderer\_;

};

struct quit\_event {

quit\_event() = default;

quit\_event(const SDL\_QuitEvent& e);

private:

SDL\_QuitEvent event\_;

};

struct mouse\_button\_event {

mouse\_button\_event() = default;

mouse\_button\_event(const SDL\_MouseButtonEvent& e);

static constexpr uint32\_t down = SDL\_MOUSEBUTTONDOWN;

static constexpr uint32\_t up = SDL\_MOUSEBUTTONUP;

static constexpr uint8\_t left = SDL\_BUTTON\_LEFT;

static constexpr uint8\_t right = SDL\_BUTTON\_RIGHT;

// button up or down

uint32\_t type() const;

// left or right button

uint8\_t button() const;

// distance from left border in pixels

int32\_t x() const;

// distance from top border in pixels

int32\_t y() const;

private:

SDL\_MouseButtonEvent event\_;

};

struct event {

// try to convert generic event to some specific event

bool extract(quit\_event& event) const;

bool extract(mouse\_button\_event& event) const;

// try to get next event

static bool poll(event& e);

private:

SDL\_Event event\_;

};

} // namespace sdl

#endif // D\_SDL\_H\_

Sdl.cpp

#include "sdl.h"

#include <SDL.h>

#include "imgui.h"

#include "imgui\_sdl.h"

#include "imgui\_impl\_sdl.h"

namespace sdl {

sdl::sdl() {

SDL\_Init(SDL\_INIT\_VIDEO);

}

sdl::~sdl() {

SDL\_Quit();

}

renderer::renderer(const std::string& window\_name):

window\_(SDL\_CreateWindow(window\_name.data(), SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,

800, 600, 0)),

renderer\_(SDL\_CreateRenderer(window\_, -1, SDL\_RENDERER\_SOFTWARE)) {

ImGui::CreateContext();

ImGui\_ImplSDL2\_Init(window\_);

ImGuiSDL::Initialize(renderer\_, 800, 600);

ImGui\_ImplSDL2\_NewFrame(window\_);

ImGui::NewFrame();

}

renderer::~renderer() {

ImGuiSDL::Deinitialize();

ImGui::DestroyContext();

SDL\_DestroyRenderer(renderer\_);

SDL\_DestroyWindow(window\_);

}

void renderer::set\_color(uint8\_t r, uint8\_t g, uint8\_t b) const {

SDL\_SetRenderDrawColor(renderer\_, r, g, b, 255);

}

void renderer::clear() const {

SDL\_RenderClear(renderer\_);

}

void renderer::draw\_line(int32\_t x1, int32\_t y1, int32\_t x2, int32\_t y2) const {

SDL\_RenderDrawLine(renderer\_, x1, y1, x2, y2);

}

void renderer::present() const {

ImGui::Render();

ImGuiSDL::Render(ImGui::GetDrawData());

SDL\_RenderPresent(renderer\_);

ImGui\_ImplSDL2\_NewFrame(window\_);

ImGui::NewFrame();

}

quit\_event::quit\_event(const SDL\_QuitEvent& e): event\_(e) {}

mouse\_button\_event::mouse\_button\_event(const SDL\_MouseButtonEvent& e): event\_(e) {}

uint32\_t mouse\_button\_event::type() const {

return event\_.type;

}

uint8\_t mouse\_button\_event::button() const {

return event\_.button;

}

int32\_t mouse\_button\_event::x() const {

return event\_.x;

}

int32\_t mouse\_button\_event::y() const {

return event\_.y;

}

bool event::extract(quit\_event& event) const {

if(event\_.type == SDL\_QUIT){

event = event\_.quit;

return true;

}

return false;

}

bool event::extract(mouse\_button\_event& event) const {

if(event\_.type == SDL\_MOUSEBUTTONDOWN || event\_.type == SDL\_MOUSEBUTTONUP){

event = event\_.button;

return true;

}

return false;

}

bool event::poll(event& e) {

bool result = SDL\_PollEvent(&e.event\_);

if(result){

ImGui\_ImplSDL2\_ProcessEvent(&e.event\_);

}

return result;

}

} // namespace sdl

Main.cpp

#include <array>

#include <fstream>

#include <iostream>

#include <memory>

#include <vector>

#include<string>

#include "sdl.h"

#include "imgui.h"

struct figure {

virtual void render(const sdl::renderer& renderer) const = 0;

virtual void save(std::ostream& os) const = 0;

virtual ~figure() = default;

};

struct vertex {

int32\_t x, y;

};

struct pentagon : figure {

pentagon(const std::array<vertex, 5>& vertices): vertices\_(vertices) {}

void render(const sdl::renderer& renderer) const override {

renderer.set\_color(255, 0, 0);

for(int32\_t i = 0; i < 5; ++i){

renderer.draw\_line(vertices\_[i].x, vertices\_[i].y,

vertices\_[(i + 1) % 5].x, vertices\_[(i + 1) % 5].y);

}

}

void save(std::ostream& os) const override {

os << "pentagon\n";

for(int32\_t i = 0; i < 5; ++i){

os << vertices\_[i].x << ' ' << vertices\_[i].y << '\n';

}

}

private:

std::array<vertex, 5> vertices\_;

};

struct hexagon : figure {

hexagon(const std::array<vertex, 6>& vertices) : vertices\_(vertices) {}

void render(const sdl::renderer& renderer) const override {

renderer.set\_color(255, 0, 0);

for (int32\_t i = 0; i < 6; ++i) {

renderer.draw\_line(vertices\_[i].x, vertices\_[i].y,

vertices\_[(i + 1) % 6].x, vertices\_[(i + 1) % 6].y);

}

}

void save(std::ostream& os) const override {

os << "hexagon\n";

for (int32\_t i = 0; i < 6; ++i) {

os << vertices\_[i].x << ' ' << vertices\_[i].y << '\n';

}

}

private:

std::array<vertex, 6> vertices\_;

};

struct octagon : figure {

octagon(const std::array<vertex, 8>& vertices) : vertices\_(vertices) {}

void render(const sdl::renderer& renderer) const override {

renderer.set\_color(255, 0, 0);

for (int32\_t i = 0; i < 8; ++i) {

renderer.draw\_line(vertices\_[i].x, vertices\_[i].y,

vertices\_[(i + 1) % 8].x, vertices\_[(i + 1) % 8].y);

}

}

void save(std::ostream& os) const override {

os << "octagon\n";

for (int32\_t i = 0; i < 8; ++i) {

os << vertices\_[i].x << ' ' << vertices\_[i].y << '\n';

}

}

private:

std::array<vertex, 8> vertices\_;

};

struct loader {

private:

std::string fig\_name;

std::array<vertex, 5> vertices\_p;

std::array<vertex, 6> vertices\_h;

std::array<vertex, 8> vertices\_o;

public:

~loader() = default;

std::vector<std::unique\_ptr<figure>> load(std::ifstream& is) {

std::vector<std::unique\_ptr<figure>> figures;

while (is >> fig\_name) {

if (fig\_name == "pentagon") {

for (int i = 0; i < 5; i++) {

is >> vertices\_p[i].x >> vertices\_p[i].y;

}

figures.emplace\_back(std::make\_unique<pentagon>(vertices\_p));

}

else if (fig\_name == "hexagon") {

for (int i = 0; i < 6; i++) {

is >> vertices\_h[i].x >> vertices\_h[i].y;

}

figures.emplace\_back(std::make\_unique<hexagon>(vertices\_h));

}

else if (fig\_name == "octagon") {

for (int i = 0; i < 8; i++) {

is >> vertices\_o[i].x >> vertices\_o[i].y;

}

figures.emplace\_back(std::make\_unique<octagon>(vertices\_o));

}

}

return figures;

}

};

struct builder {

virtual std::unique\_ptr<figure> add\_vertex(const vertex& v) = 0;

virtual ~builder() = default;

};

struct pentagon\_builder : builder {

std::unique\_ptr<figure> add\_vertex(const vertex& v) {

vertices\_[n\_] = v;

n\_ += 1;

if (n\_ != 5) {

return nullptr;

}

return std::make\_unique<pentagon>(vertices\_);

}

private:

int32\_t n\_ = 0;

std::array<vertex, 5> vertices\_;

};

struct hexagon\_builder : builder {

std::unique\_ptr<figure> add\_vertex(const vertex& v) {

vertices\_[n\_] = v;

n\_ += 1;

if (n\_ != 6) {

return nullptr;

}

return std::make\_unique<hexagon>(vertices\_);

}

private:

int32\_t n\_ = 0;

std::array<vertex, 6> vertices\_;

};

struct octagon\_builder : builder {

std::unique\_ptr<figure> add\_vertex(const vertex& v) {

vertices\_[n\_] = v;

n\_ += 1;

if (n\_ != 8) {

return nullptr;

}

return std::make\_unique<octagon>(vertices\_);

}

private:

int32\_t n\_ = 0;

std::array<vertex, 8> vertices\_;

};

int main() {

sdl::renderer renderer("Editor");

bool quit = false;

std::vector<std::unique\_ptr<figure>> figures;

std::unique\_ptr<builder> active\_builder = nullptr;

const int32\_t file\_name\_length = 128;

char file\_name[file\_name\_length] = "";

int32\_t remove\_id = 0;

while (!quit) {

renderer.set\_color(0, 0, 0);

renderer.clear();

sdl::event event;

while (sdl::event::poll(event)) {

sdl::quit\_event quit\_event;

sdl::mouse\_button\_event mouse\_button\_event;

if (event.extract(quit\_event)) {

quit = true;

break;

}

else if (event.extract(mouse\_button\_event)) {

if (active\_builder && mouse\_button\_event.button() == sdl::mouse\_button\_event::left &&

mouse\_button\_event.type() == sdl::mouse\_button\_event::down) {

std::unique\_ptr<figure> figure =

active\_builder->add\_vertex(vertex{ mouse\_button\_event.x(), mouse\_button\_event.y() });

if (figure) {

figures.emplace\_back(std::move(figure));

active\_builder = nullptr;

}

}

}

}

for (const std::unique\_ptr<figure>& figure : figures) {

figure->render(renderer);

}

ImGui::Begin("Menu");

if (ImGui::Button("New canvas")) {

figures.clear();

}

ImGui::InputText("File name", file\_name, file\_name\_length - 1);

if (ImGui::Button("Save")) {

std::ofstream os(file\_name);

if (os) {

for (const std::unique\_ptr<figure>& figure : figures) {

figure->save(os);

}

}

}

ImGui::SameLine();

if (ImGui::Button("Load")) {

std::ifstream is(file\_name);

if (is) {

loader loader;

figures = loader.load(is);

}

}

if (ImGui::Button("Pentagon")) {

active\_builder = std::make\_unique<pentagon\_builder>();

}

if (ImGui::Button("Hexagon")) {

active\_builder = std::make\_unique<hexagon\_builder>();

}

if (ImGui::Button("Octagon")) {

active\_builder = std::make\_unique<octagon\_builder>();

}

ImGui::InputInt("Remove id", &remove\_id);

if (ImGui::Button("Remove")) {

if(remove\_id< figures.size())

figures.erase(figures.begin() + remove\_id);

}

ImGui::End();

renderer.present();

}

}

1. **Объяснение результатов работы программы**

В исполняемом окне фон заполняется черным и выводится меню с несколькими вариантами действий на выбор. Пользователю предоставляется кнопочки рисования фигур (пятиугольник, шестиугольник и восьмиугольник), кнопочки удаления, сохранения в файл и чтения из файла. После нажатия кнопочки рисования фигуры по точкам, в которые нажимается мышью, строится заданная фигура. После этого можно добавить в строку при “File name” путь к файлу, в который мы хотим сохранить рисунок, и сохранить его. Извлечь из файла рисунок можно кнопкой “Load”.

1. **Вывод**

Янаучилась создавать простые графические редакторы с функцией создания и удаления фигур, рисования фигур по точкам. Моя программа и ее визуальное оформление исполнения просты и понятны обычному пользователю. Мне было очень интересно познакомиться со средствами визуального оформления программы.