**КУРСОВАЯ РАБОТА**

**по дисциплине “Объектно-ориентированное программирование”.**

**Автор: студент СибГУТИ гр. ИП-913,**

**Степин Александр Вячеславович (2 курс)**

**ОГЛАВЛЕНИЕ**

[**1. Поста**](#_Toc515570163)**новка задачи (вариант №8) 03**

**a. Описание задания**  **03**

**b. Вариант выполнения задания**  **03**

**c. Описание иерархии объектов и методов объектов**  **04**

**2**[**.**](#_Toc515570163) **Описание алгоритма программы**  **05**

**a. Алгоритм прораммы**  **05**

**b. Вывод**  **06**

**3**[**.**](#_Toc515570163) **Скриншоты работы программы**  **06**

**a. Скриншоты в полноэкранном режиме**  **06**

**b. Скриншоты в оконном режиме**  **08**

**4**[**.**](#_Toc515570163) **Листинг кода**  **09**

**a. main.cpp**  **09**

**b. App.hpp**  **11**

**c. WinApp.hpp**  **11**

**d. WinSettings.hpp**  **12**

**e. TshapeMover.hpp**  **12**

**f. Tshape.hpp**  **14**

**g. App.cpp**  **15**

**h. WinApp.cpp**  **15**

**l. TshapeMover.cpp**  **16**

**j. Tshape.cpp**  **21**

[**1. Поста**](#_Toc515570163)**новка задачи**

***a.*** *Описание задания*.

Круговое движение с вращением составного графического объекта.

***b.*** *Вариант выполнения задания*.

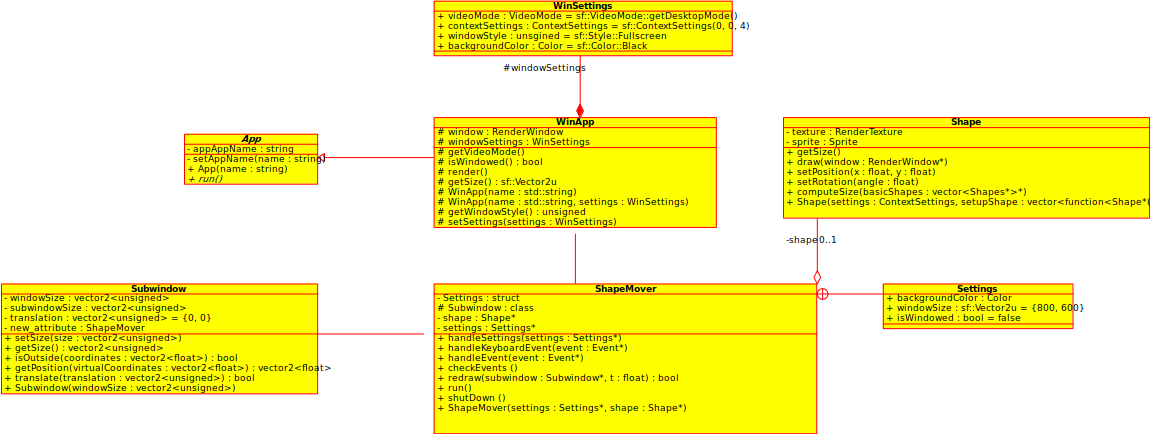
Обозначим объект, который будет двигаться и вращаться. Для реализации кругового движения (движения по окружности) используем математические (физические) формулы движения точки по окружности *x = x0 + rcos(t)*, *y = y0 + rsin(t)*, где x0, y0 – центральная точка окружности. Используем для реализации программы библиотеку SFML (<https://en.sfml-dev.org/>), которая предоставляет пользователю удобный интерфейс для работы с окном приложения и простейшими геометрическими фигурами.

Т. к. математические формулы движения по окружности подразумевают точку, а не объект с заданным размером, то сведем задачу к работе с точкой. Но для начала определим размер и форму фигуры. Фигура составная, следовательно, строится путем комбинации простейших геометрических фигур. В этом варианте задания я реализую нечто похожее на космический корабль, как показано в примере. И оно будет двигаться и вращаться. Для двигателей используем прямогульники, для “”тела” корабля – круг, для “носа” – треугольник. Для удобства при проверке выхода за границы видимой области обозначим форму фигуры как прямоугольник наименьшей площади, такой, что в него помещается вся наша составная фигура. Обозначим точку, которая находится на пересечении левой и верхней сторон нашего прямогульника. Она и будет двигаться по окружности.

Для того, чтобы наша фигура не уходила за границы окна, задаим рабочую область. Т. к. мы работаем с точкой на левом верхнем углу фигуры, импровизированного прямоугольника, то ограничичим левую и нижнюю части окна в использовании. Вычтем из реальных высоты и длины окна высоту и длину заданной фигуры, и получим размер области, с которой будем работать. Кстати, можно урезать рабочую область видимости до квадрата, у которого стороны будут равны длине наименьшей стороны ещё неурезанной рабочей области. Квадрат будет наименьшей площади, такой, что в него помещается вся окружность. И для более приятного глазу вида можно центрировать урезанную рабочую область относительно неурезанной. Таким образом, картинка будет находится по центру, а сама окружность будет смещена чуть левее и выше, как раз на размер фигуры. Для реализации работы с урезанной рабочей областью, меньшей, чем размер окна, создадим отдельный класс.

***c.*** *Описание иерархии объектов и методов объектов.*

Иерархия классов, объектов и их методов следующая (на изображении представлена uml диаграмма, а также [в файле](../../umbrello/Shape%20mover.svg) “Shape mover.svg”):



Ред. 1. Я добавил классы настроек для класса окна и класса приложения. Настройки проложения передаются в конструктор по ссылке и приложение хранит ссылку. Методы объекта окна используют конструктор копирования и, в случае, если приложение не задаёт настройки окна, обект настроек в любом случае создаётся при создании объекта окна, т.е. обект окна имеет заданные настройки в любом случае. Это сделано для того, чтобы перегрузить конструктор класса окна и дать возомжность пользователю не создавать настройки вручную. А класс приложения имеет единственный конструктор, который обязательным параметром имеет ссылку на объект настроек.

Ред 2. Теперь проект есть на гитхабе, и больше не будет ночных марафонов по приведению приложения в рабочее состояние. На компьютере есть одна старая версия проекта, но в отчёте её не будет. А будет ссылка на гитхаб, вот она: <https://github.com/alexvsezanyato/shape-mover/tree/master/>. На ветке мастер самый первый коммит, ветка main не используется (по крайней мере на момент создания отчёта).

Класс App – “примитивный” абстрактный класс, имеет в себе атрибуты и поля, общие для всех приложений.

Класс WinApp расширяет класс App, описывая поля и методы, характерные для оконного приложения. Является абстрактым классом, требует реализации.

Класс TshapeMover – реализация окнного приложения, где описаны все поля методы, необходимые для движения фигуры по окружности, её поворота (форма фигуры в виде прямоульника нужна для того чтобы создать спрайт sf::Sprite и уже с ним работать; одна из сторон этого прямоугольника паралельна касательной к окружности).

Существуют также “промежуточные классы”: классы настроек WinSettings, TshapeMover::Settings, TshapeMover::Subwindow. Классы WinSettings и TshapeMover::Settings нужны лишь для зранения настроек WinApp и TshapeMover соотвественно. С ними удобно работать. Класс TshapeMover::Subwindow выполняет более весомую функцию: предоставляет интерфейс для работы с вирутальным окном, которое может быть меньше реального окна и смещено относительно верней левой части окна на удобное для пользователя значение.

Таким образом, мы имеем класс приложения, класс фигуры и промежуточные классы. Для работы приложения помимо класса фигуры используются вспомогательный класс виртуального окна и базовый класс оконного приложения, который, в свою очередь, имеет базовый абстрактный класс приложения, который должен быть реализован, как например, класс оконного приложения. Но процесс работы класс оконного приложения не описывает, а делегирует данную задачу классу-потомку, вследствие чего и сам является абстрактным классом. Класс, описывающий процесс работы программы, использует все вохможности как базового класса, так и класса фигуры, и, по сути, является основой программы, где и задействуются все пользовательские данные, как, например, описание фигуры, размер окна и т. д.

**2**[**.**](#_Toc515570163) **Описание алгоритма программы**

***a.*** *Алгоритм программы*

Учитывая иерархию классов, объектов и их методов, алгоритм работы программы следующий:

Приложение принимает на вход простейшие геометрические фигуры, которые объединяет в одну составную, вычисляет размер прямоугольной области, такой, что её площадь наименьшая и каждая из простейших фигур находится в этой области и не выходит за её границы.

За объект, движущийся по окружности, принимается точка, находящаяся в вехнем левом углу получанной прямоугльной области. За объект вращающийся и находящийся паралельно касательной к окружности принимается верхняя сторона пряоугльной области.

Программа вращает и передвигает фигуру в круговом движении до тех пор, пока не будет закрыта.

***b.*** *Выводы по разделу #2*

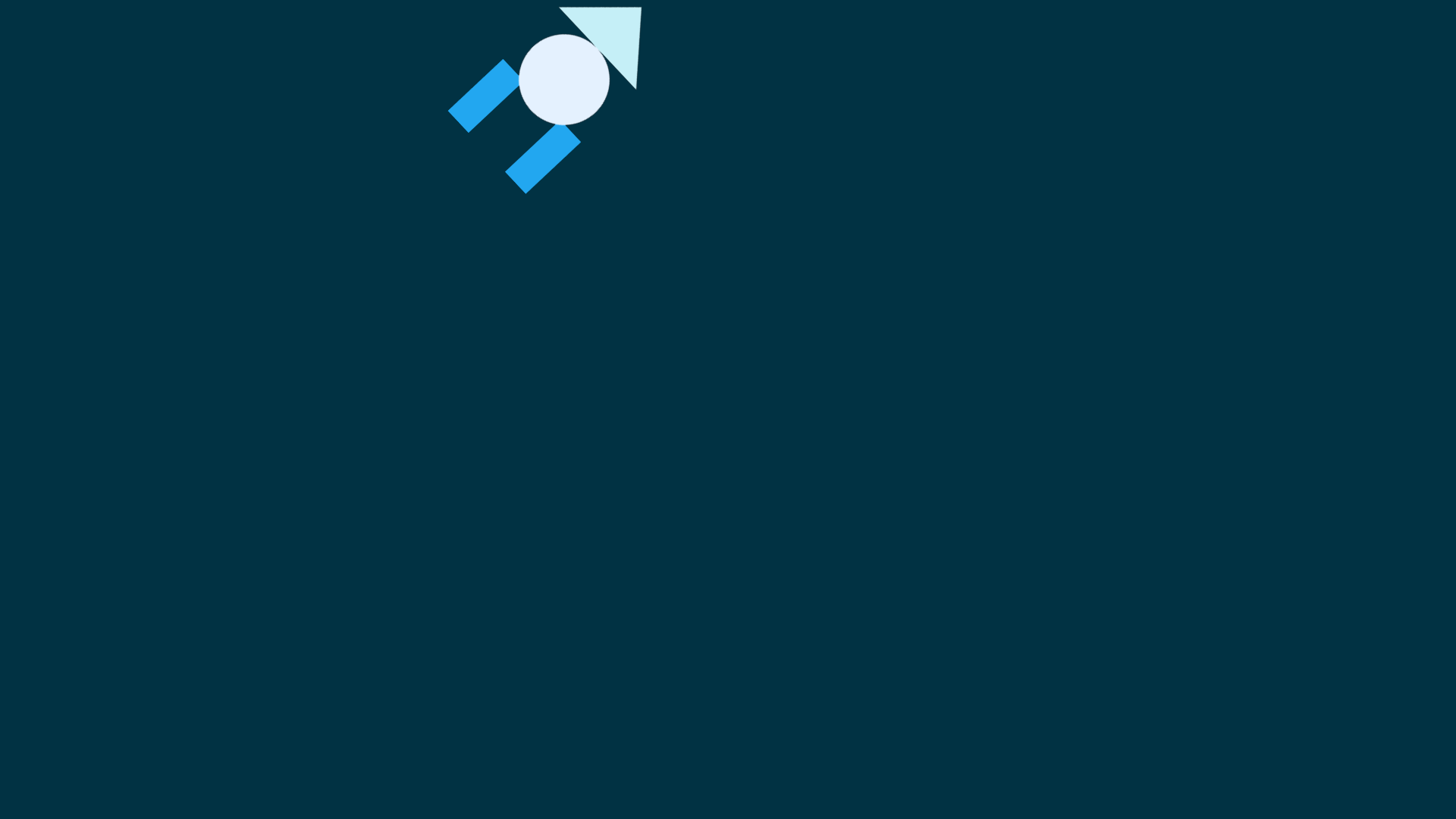
Так как каждый описанный выше этап работы программы имеет вполне определённое поведение при различных входных данных (за исключением багов, недочётов и уязвимостей которые будут выявляться в ходе эксплуатации программы при условии поддержки и развития проиложенния разработчиком, то есть мной (-: ), можем считать процесс работы данной программы алгоритмом.

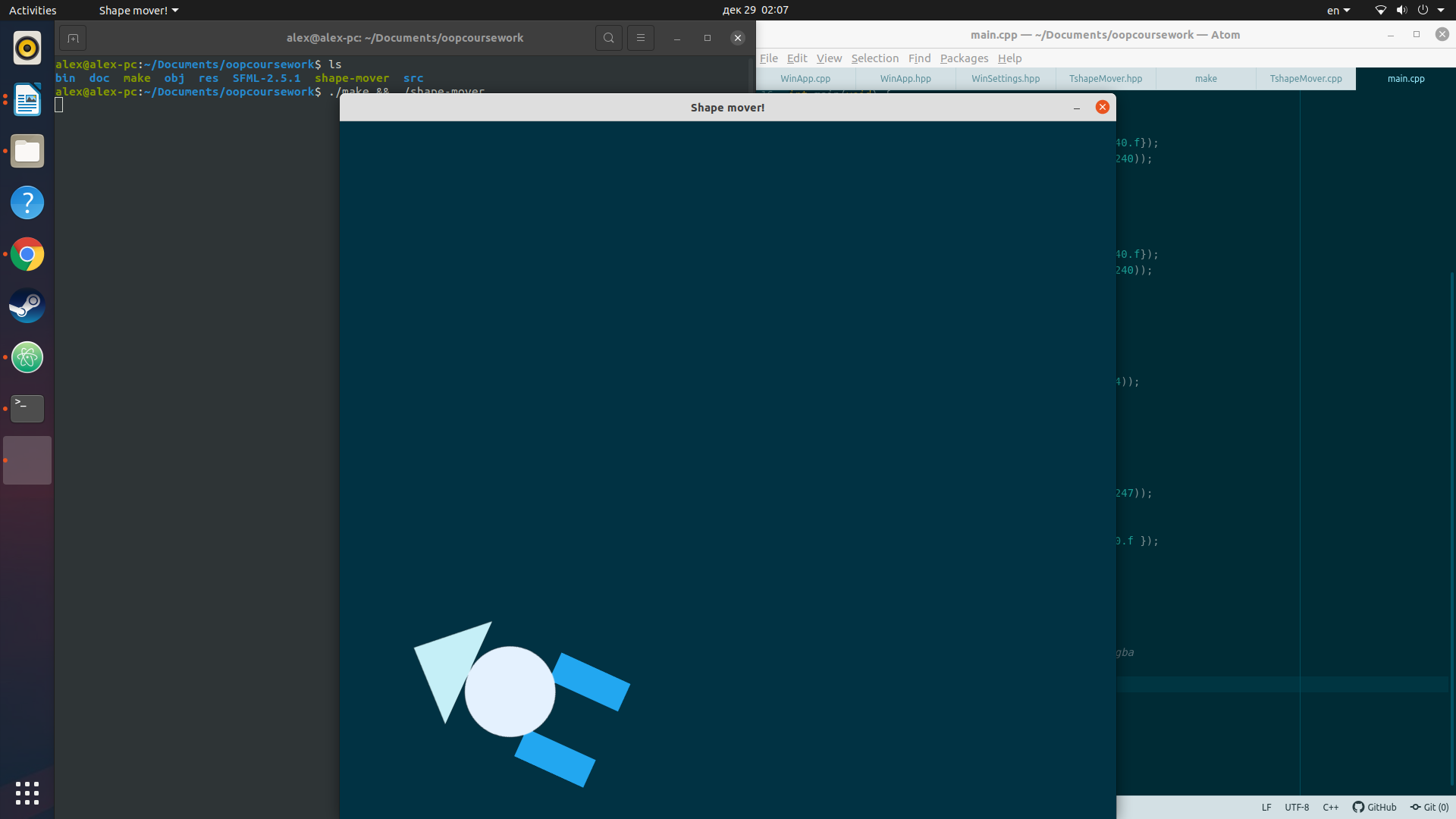
**3**[**.**](#_Toc515570163) **Скриншоты работы программы**

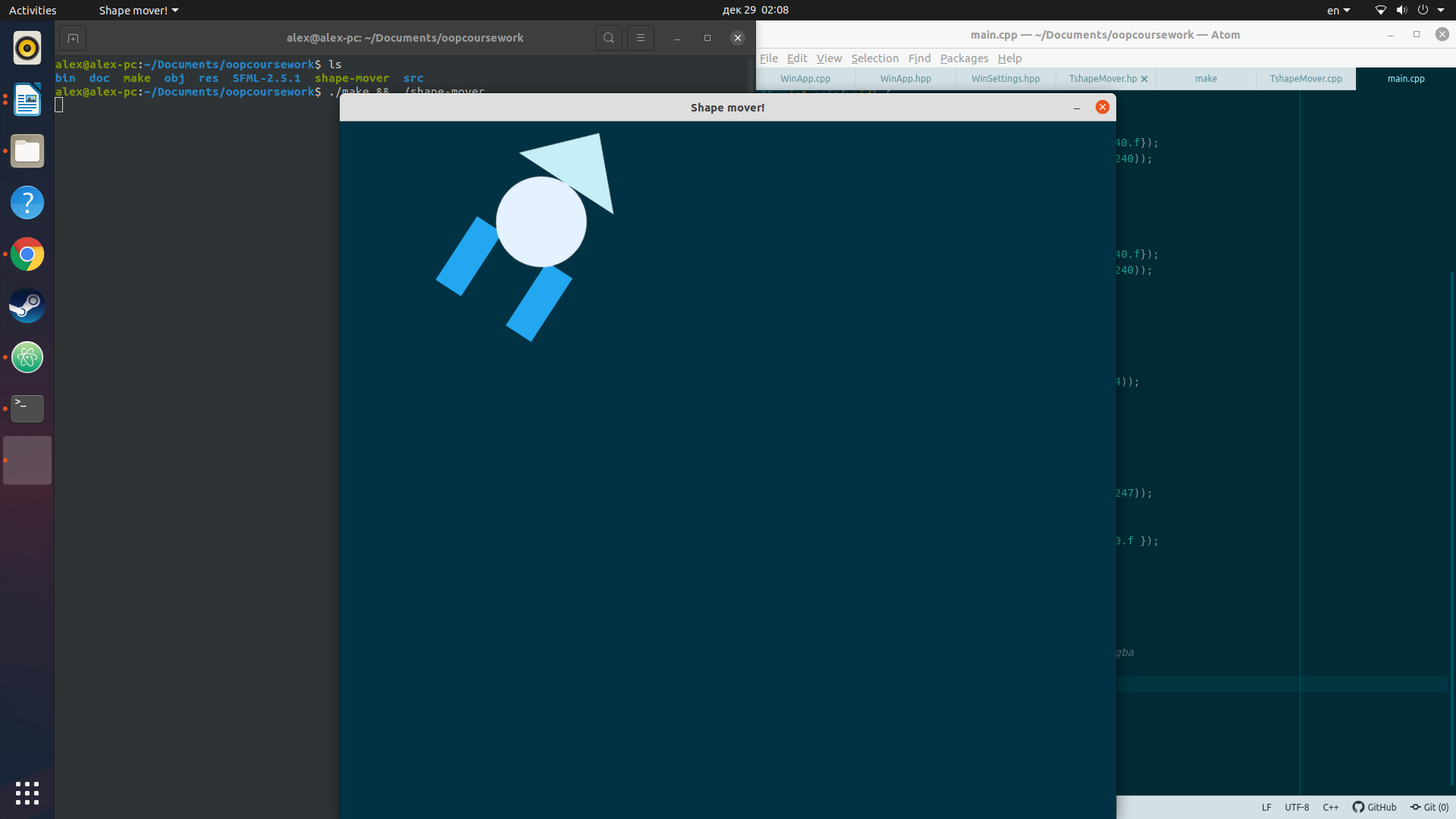
***a.*** *Полноэкранные скриншоты*

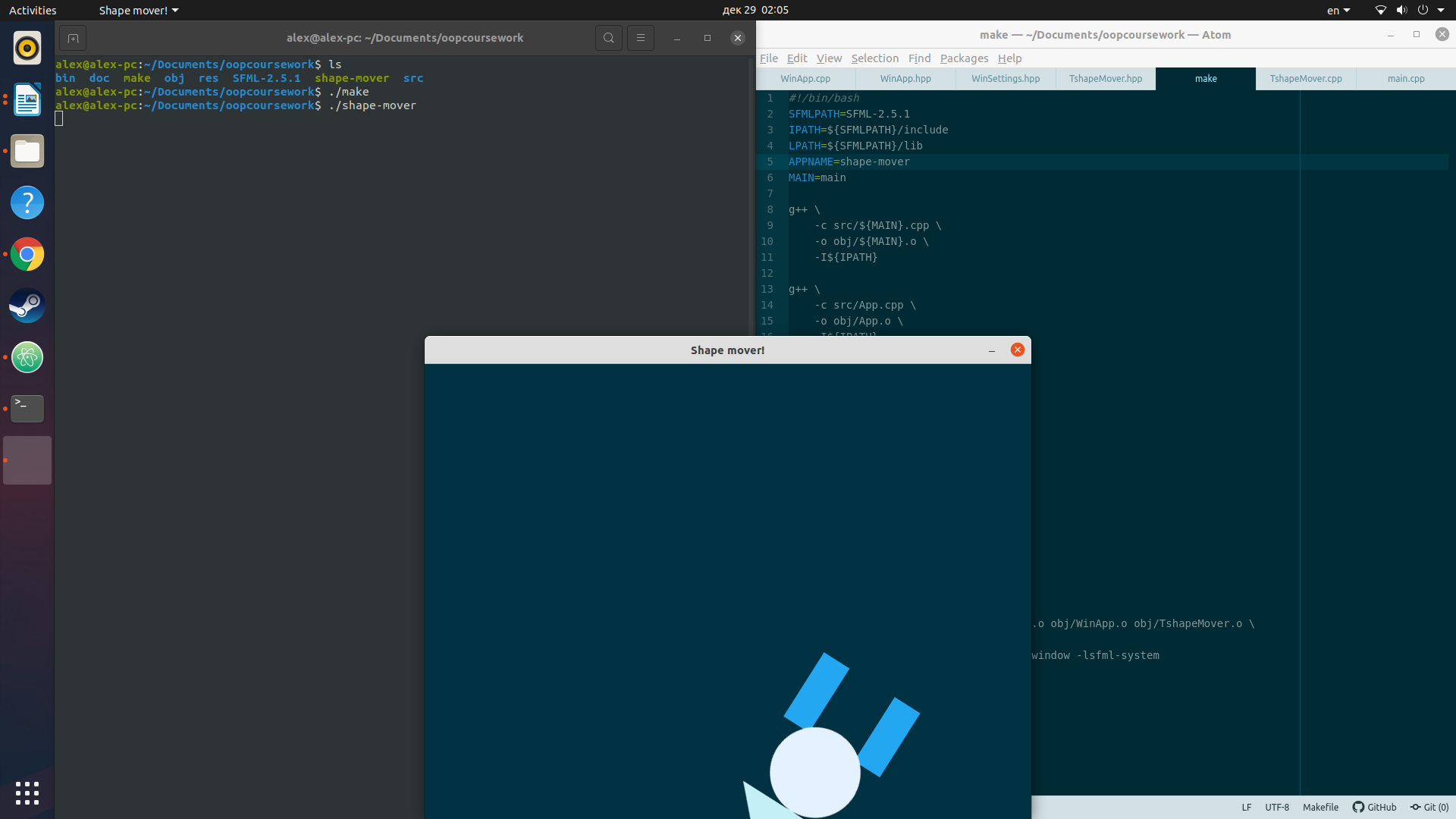


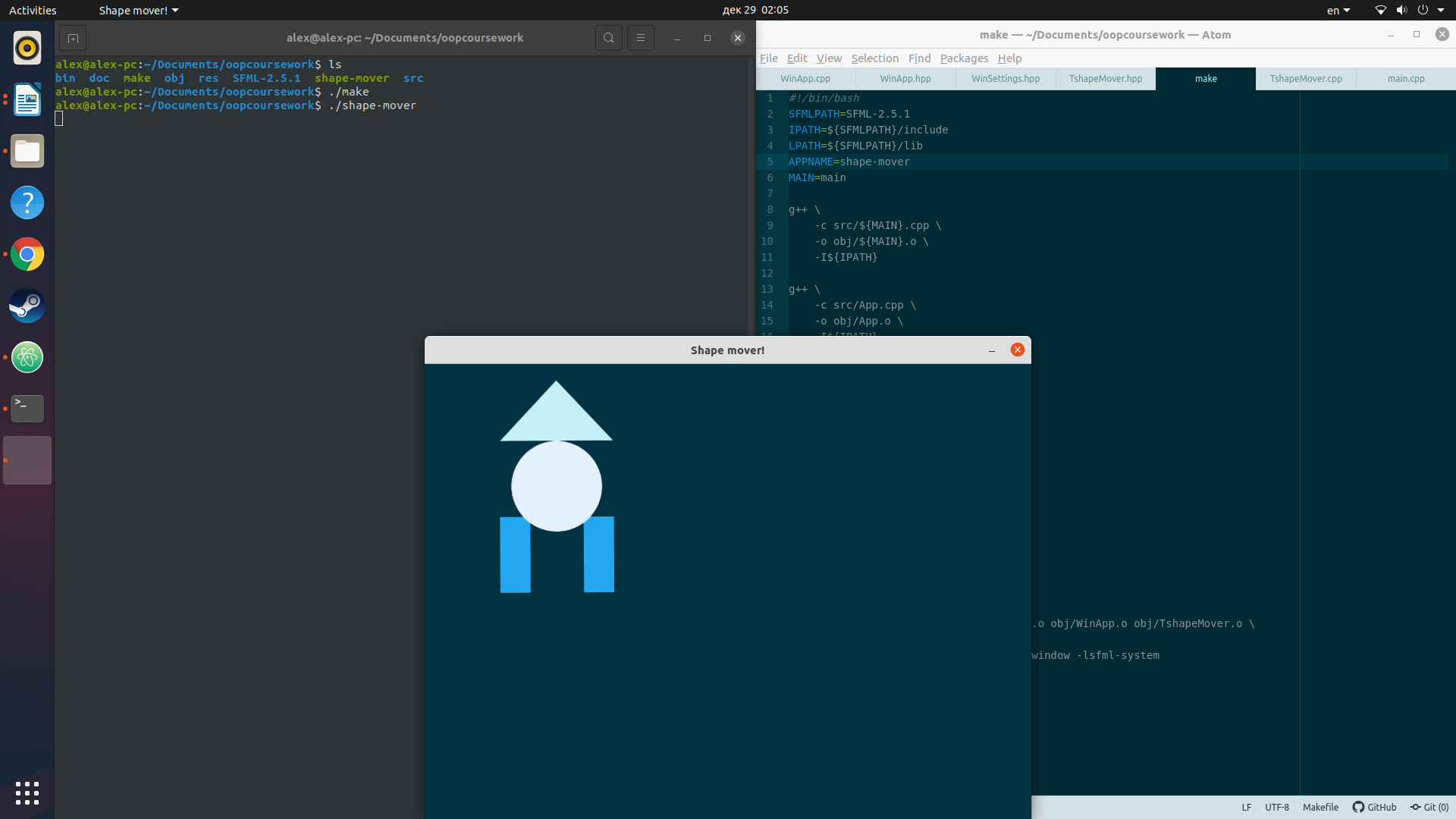












**4**[**.**](#_Toc515570163) **Листинг кода**

Подсветка синтаксиса реализована с помощью онлайн сервиса подсветки кода. С нумерацией строк посердством функций сервиса ломаются отступы (полностью исчезают). С нумерацией посредством функций LibreOffice лтступы тоже ломаются! Они есть, но они сломаны. Поэтому первый файл будет с нумерацией, остальные могут быть без.

***a.*** *main.cpp*

1. #include <SFML/Graphics/Color.hpp>
2. #include <SFML/Graphics/RectangleShape.hpp>
3. #include <SFML/Graphics/ConvexShape.hpp>
4. #include <SFML/Graphics/CircleShape.hpp>
5. #include "Tshape.hpp"
6. #include "TshapeMover.hpp"
7. #include <iostream>
8. using sf::Color;
9. // shapes available
10. using Rectangle = sf::RectangleShape;
11. using Circle = sf::CircleShape;
12. using Convex = sf::ConvexShape;
13. int main(void) {
14. Tshape shape({
15. [] {
16. auto rectangle = new Rectangle({100.f, 40.f});
17. rectangle->setFillColor(Color(34, 167, 240));
18. rectangle->setPosition(0.f, 0.f);
19. return rectangle;
20. },
21. [] {
22. auto rectangle = new Rectangle({100.f, 40.f});
23. rectangle->setFillColor(Color(34, 167, 240));
24. rectangle->setPosition(0.f, 110.f);
25. return rectangle;
26. },
27. [] {
28. auto circle = new Circle(60.f, 300);
29. circle->setFillColor(Color(228, 241, 254));
30. circle->setPosition(80.f, 15.f);
31. return circle;
32. },
33. [] {
34. auto triangle = new Convex(3);
35. triangle->setFillColor(Color(197, 239, 247));
36. triangle->setPoint(0, { 0.f, 0.f });
37. triangle->setPoint(1, { 0.f, 150.f });
38. triangle->setPoint(2, { 80.f, 0.5f \* 150.f });
39. triangle->setPosition(200.f, 0.f);
40. return triangle;
41. },
42. });
43. TshapeMover::Settings set;
44. set.backgroundColor = Color(1, 50, 67, 1); // rgba
45. set.isWindowed = true;
46. set.windowSize = {800, 600};
47. TshapeMover app(&set, &shape);
48. app.run();
49. return 0;
50. }

***b.*** *App.hpp*

#pragma once #include <string>

class App {

public:

std::string appName;

void setAppName(std::string name);

App() = delete;

protected:

App(std::string name);

public:

virtual void run() = 0;

std::string getAppName();

};

***c.*** *WinApp.hpp*

#pragma once

#include <string>

#include <SFML/System/Vector2.hpp>

#include <SFML/Window/ContextSettings.hpp>

#include <SFML/Window/VideoMode.hpp>

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Window/WindowStyle.hpp>

#include "App.hpp"

#include "WinSettings.hpp"

class WinApp : public App {

private:

WinApp() = delete;

protected:

WinSettings windowSettings;

sf::RenderWindow window;

sf::VideoMode getVideoMode();

unsigned getWindowStyle();

bool isWindowed();

void setSettings(WinSettings);

void render();

sf::Vector2u getSize();

WinApp(std::string name, WinSettings);

WinApp(std::string name);

};

***d.***  *WinSettings.hpp*

#pragma once

#include <SFML/Window/ContextSettings.hpp>

#include <SFML/Window/VideoMode.hpp>

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Window/WindowStyle.hpp>

struct WinSettings {

public:

sf::VideoMode videoMode = sf::VideoMode::getDesktopMode();

sf::ContextSettings contextSettings = sf::ContextSettings(0, 0, 4);

unsigned windowStyle = sf::Style::Fullscreen;

sf::Color backgroundColor = sf::Color::Black;

};

***e.***  *TshapeMover.hpp*

#pragma once

#include "WinApp.hpp"

#include "WinSettings.hpp"

#include "Tshape.hpp"

#include <cmath>

#include <SFML/System/Vector2.hpp>

#include <SFML/Window/Keyboard.hpp>

#include <SFML/Window/ContextSettings.hpp>

#include <SFML/Window/Event.hpp>

#include <SFML/Graphics/Color.hpp>

//

// ::TshapeMover

class TshapeMover : public WinApp {

public:

class Subwindow;

struct Settings;

private:

Tshape\* shape;

// bool isMoving = true;

TshapeMover();

Settings\* settings;

void setupSettings();

void setupSettings(Settings\*);

public:

void handleKeyboardEvent(sf::Event\* event);

void handleEvent(sf::Event\* event);

void handleSettings(Settings\*);

void checkEvents();

bool redraw(Subwindow\* subwindow, float t);

void run() override;

void shutDown(void);

TshapeMover(Settings\* settings, Tshape\* shape);

};

//

// ::TshapeMover::Settings

struct TshapeMover::Settings {

public:

sf::Color backgroundColor = sf::Color::Black;

sf::Vector2u windowSize = {800, 600};

bool isWindowed = false;

};

//

// ::TshapeMover::Subwindow

class TshapeMover::Subwindow {

private:

sf::Vector2u subwindowSize;

sf::Vector2u windowSize;

sf::Vector2u translation = {0, 0};

public:

Subwindow(sf::Vector2u windowSize);

bool translate(sf::Vector2u translation);

sf::Vector2f getPosition(sf::Vector2f coords);

bool isOutside(sf::Vector2f coords);

bool setSize(sf::Vector2u size);

sf::Vector2u getSize(void);

};

***f.***  *Tshape.hpp*

#pragma once

#include <vector>

#include <utility>

#include <functional>

#include <SFML/System/Vector2.hpp>

#include <SFML/Window/ContextSettings.hpp>

#include <SFML/Graphics/RenderWindow.hpp>

#include <SFML/Graphics/RenderTexture.hpp>

#include <SFML/Graphics/Sprite.hpp>

#include <SFML/Graphics/Shape.hpp>

class Tshape {

using TcreateShape = std::function <sf::Shape\*()>;

using BasicShape = sf::Shape;

private:

sf::RenderTexture texture;

sf::Sprite sprite;

sf::Vector2u size = {0, 0};

std::vector<BasicShape\*> basicShapes;

sf::Vector2u computeSize();

public:

sf::Vector2u getSize();

void draw(sf::RenderWindow\* w);

void setPosition(float x, float y);

void setRotation(float angle);

void build(sf::ContextSettings);

Tshape(std::vector<TcreateShape>);

~Tshape();

};

***g.***  *App.cpp*

#include "App.hpp"

void App::setAppName(std::string name) { this->appName = name; }

std::string App::getAppName() { return this->appName; }

App::App(std::string name) { this->setAppName(name); }

***h.*** *WinApp.cpp*

#include "WinApp.hpp"

#include <iostream>

sf::VideoMode WinApp::getVideoMode() { return windowSettings.videoMode; }

unsigned WinApp::getWindowStyle() { return windowSettings.windowStyle; }

void WinApp::render() {

this->window.create(

windowSettings.videoMode,

this->getAppName(),

windowSettings.windowStyle,

windowSettings.contextSettings

);

}

sf::Vector2u WinApp::getSize() {

sf::VideoMode\* videoMode = &windowSettings.videoMode;

auto width = videoMode->width;

auto height = videoMode->height;

return {width, height};

}

void WinApp::setSettings(WinSettings settings) {

this->windowSettings = settings;

return;

}

bool WinApp::isWindowed() {

auto style = windowSettings.windowStyle;

using sf::Style::Fullscreen;

if (style != Fullscreen) return true;

else return false;

}

WinApp::WinApp(std::string name, WinSettings settings) :

App(name),

windowSettings(settings)

{ ; }

WinApp::WinApp(std::string name) :

App(name)

{ ; }

***l.*** *TshapeMover.cpp*

#include "TshapeMover.hpp"

//

// ::TshapeMover::subwindow

bool TshapeMover::Subwindow::translate(sf::Vector2u translation) {

if (translation.x + subwindowSize.x > windowSize.x) return false;

if (translation.y + subwindowSize.y > windowSize.y) return false;

this->translation = translation;

return true;

}

sf::Vector2f TshapeMover::Subwindow::getPosition(sf::Vector2f coords) {

float x, y;

x = coords.x + (float) translation.x;

y = coords.y + (float) translation.y;

return {x, y};

}

bool TshapeMover::Subwindow::isOutside(sf::Vector2f coords) {

// case 1

sf::Vector2u max;

max.x = subwindowSize.x + translation.x;

max.y = subwindowSize.y + translation.y;

if ((int) coords.x > max.x) return true;

if ((int) coords.y > max.y) return true;

// case 2

sf::Vector2u min;

min.x = translation.x;

min.y = translation.y;

if ((int) coords.x < min.x) return true;

if ((int) coords.y < min.y) return true;

// if not outside

return false;

}

bool TshapeMover::Subwindow::setSize(sf::Vector2u size) {

sf::Vector2u\* windowSize = &this->windowSize;

// if new size is larger than allowed

if (size.x > windowSize->x) return false;

if (size.y > windowSize->y) return false;

// if new size is valid

this->subwindowSize = size;

return true;

}

sf::Vector2u TshapeMover::Subwindow::getSize(void) {

sf::Vector2u\* subwindowSize = &this->subwindowSize;

unsigned width = subwindowSize->x;

unsigned height = subwindowSize->y;

return {width, height};

}

TshapeMover::Subwindow::Subwindow(sf::Vector2u windowSize) {

// constructor be waiting of the real window size

this->windowSize = windowSize;

// as default, this constructor sets its own size = the real one

this->subwindowSize = this->windowSize;

}

//

// ::TshapeMover

void TshapeMover::shutDown(void) {

sf::Window\* w = &this->window;

w->close();

return;

}

void TshapeMover::handleKeyboardEvent(sf::Event\* event) {

switch ((\*event).key.code) {

case sf::Keyboard::Escape: {

this->shutDown();

break;

}

}

}

void TshapeMover::handleEvent(sf::Event\* event) {

switch (event->type) {

case sf::Event::Closed: {

this->shutDown();

break;

}

case sf::Event::KeyPressed: {

this->handleKeyboardEvent(event);

break;

}

}

}

void TshapeMover::checkEvents(void) {

sf::Event event;

sf::Window\* w = &this->window;

while (w->pollEvent(event)) this->handleEvent(&event);

return;

}

bool TshapeMover::redraw(Subwindow\* subwindow, float t) {

// x must = y at the moment

unsigned size = subwindow->getSize().x;

// radius

float r;

r = size \* 0.5f;

// center (x0, y0)

float x0, y0;

x0 = subwindow->getPosition({r, r}).x;

y0 = subwindow->getPosition({r, r}).y;

// current position

float x, y;

x = x0 + r \* cos(t);

y = y0 + r \* sin(t);

sf::RenderWindow\* w = &this->window;

Tshape\* shape = this->shape;

w->clear(windowSettings.backgroundColor);

shape->setRotation(t \* 57.2958f + 90.f);

shape->setPosition(x, y);

shape->draw(w);

w->display();

return true;

}

void TshapeMover::run() {

// prepare & render a window

// there're default settings,

// so from now on, you shouldn't care about setting this

// but you can if you want

this->setupSettings(this->settings);

this->render();

// shape has basic sfml shapes now,

// it must create a sprite

// a sprite has its own context settings,

// so you must pass the settings to it

// don't forget about building your shape, please!

shape->build(windowSettings.contextSettings);

sf::RenderWindow\* w = &this->window;

sf::Vector2u windowSize = this->getSize();

Subwindow subwindow(windowSize);

unsigned width, height;

width = windowSize.x;

height = windowSize.y;

// take shape size into account

// sf::Vector2u shapeSize = this->shape->getSize();

// unsigned width, height;

// width = windowSize.x - shapeSize.x;

// height = windowSize.y - shapeSize.y;

// square size for circle movement

unsigned squaredSize;

if (width > height) squaredSize = height;

else squaredSize = width;

subwindow.setSize({ squaredSize, squaredSize });

// center subwindow

unsigned translation = (windowSize.x - squaredSize) \* 0.5f; // - shapeSize.x \* 0.5f;

subwindow.translate({ translation, 0 });

sf::Clock clock;

// warning! the speed does not change now

// it wont do any speed change

// your default speed value

float speed = 0.5f;

// if you wanna change the speed

float maxSpeed = 2.f;

// the angle changes by the step

float step = .003f;

while (w->isOpen()) {

for (float i = -3.14f; i <= 3.14f; ) {

// system checks

if (!w->isOpen()) break;

checkEvents();

// time check

sf::Time time = clock.getElapsedTime();

if (time.asMicroseconds() < 5000.f) continue;

// next step, redraw

this->redraw(&subwindow, i);

i += step;

clock.restart();

}

}

}

void TshapeMover::handleSettings(TshapeMover::Settings\* settings) {

WinSettings\* set = &this->windowSettings;

set->backgroundColor = settings->backgroundColor;

if (settings->isWindowed) set->windowStyle = sf::Style::Close;

else set->windowStyle = sf::Style::Fullscreen;

if (!settings->isWindowed) return;

sf::Vector2u size = settings->windowSize;

set->videoMode.width = size.x;

set->videoMode.height = size.y;

return;

}

void TshapeMover::setupSettings(TshapeMover::Settings\* settings) {

TshapeMover::setupSettings();

handleSettings(settings);

return;

}

void TshapeMover::setupSettings() {

WinSettings\* ws = &this->windowSettings;

ws->contextSettings = sf::ContextSettings(0, 0, 16);

return;

}

TshapeMover::TshapeMover(TshapeMover::Settings\* settings, Tshape\* shape) :

WinApp("Shape mover!"),

shape(shape),

settings(settings)

{ ; }

***j.*** *Tshape.cpp*

#include "Tshape.hpp"

#include <iostream>

sf::Vector2u Tshape::computeSize(void) {

sf::Vector2u size{0, 0};

// ..

for (auto i : this->basicShapes) {

for (int j = i->getPointCount(); j >= 0; --j) {

sf::Vector2f point = i->getPoint(j);

sf::Vector2f position = i->getPosition();

float width = point.x + position.x;

float height = point.y + position.y;

if (size.x < width) size.x = point.x + position.x;

if (size.y < height) size.y = point.y + position.y;

}

}

return size;

}

void Tshape::draw(sf::RenderWindow\* w) {

w->draw(this->sprite);

return;

}

void Tshape::setPosition(float x, float y) {

sf::Sprite\* sprite = &this->sprite;

sprite->setPosition(x, y);

return;

}

void Tshape::setRotation(float angle) {

sf::Sprite\* sprite = &this->sprite;

sprite->setRotation(angle);

return;

}

sf::Vector2u Tshape::getSize() {

sf::Vector2u textureSize = this->texture.getSize();

unsigned width = textureSize.x;

unsigned height = textureSize.y;

return {width, height};

}

void Tshape::build(sf::ContextSettings contextSettings) {

// texture size = combined shape size

sf::Vector2u\* size = &this->size;

this->texture.create(size->x, size->y, contextSettings);

// ..

this->texture.clear(sf::Color::Transparent);

for (auto i : this->basicShapes) this->texture.draw(\*i);

this->texture.display();

this->texture.setSmooth(true);

this->sprite.setTexture(this->texture.getTexture());

return;

}

Tshape::Tshape(std::vector<TcreateShape> createShapes) {

// from now on the class contains basic shapes

std::vector<sf::Shape\*> shapes = this->basicShapes;

for (auto createShape : createShapes) shapes.push\_back(createShape());

// check basic shapes count

// if (!shapes->size()) return;

// compute combined shape size

this->basicShapes = shapes;

this->size = this->computeSize();

}

Tshape::~Tshape() {

for (auto i : basicShapes) delete i;

return;

}