ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №7

за курсом «Об’єктно Орієнтованого Програмування»

студента групи ПЗ-21-2

Гудимова Дениса Сергійовича

кафедра математичного забезпечення ЕОМ, ДНУ

2022/2023 навч.р.

1. Постановка задачі

Взявши за основу класи геометричних фігур і графічний інтерфейс, які були розроблені в курсовій роботі, створіть програму, яка працює в багатопоточному режимі і виконує наступне:

1. Раз у певний інтервал часу на поле у випадкову точку додається випадкова фігура. При цьому існує певний максимум фігур, які перебувають на полі одночасно. Наприклад, не більше 10 фігур.

2. Кожна фігура обирає собі ціль з інших фігур, до якої починає рухатися.

3. При зіткненні фігура і її ціль видаляються. При цьому враховуються лише зіткнення фігури з її ціллю, а зіткнення з іншими фігурами ігноруються.

Розподіл на потоки здійснюється наступним чином:

1. Один потік відповідає за додавання фігур.

2. Другий потік відповідає за перевірку наявності цілей у кожної фігури і присвоєння нових цілей для тих фігур, у яких попередня ціль перестала існувати.

3. Третій потік переміщує фігури до їхніх цілей і здійснює перевірку, чи зіткнулися фігури.

4. Четвертий потік відповідає за коректне видалення фігур, які зіткнулись.

При виконанні роботи продемонструвати можливість передачі функцій в потік у вигляді лямда-виразів.

2. Опис розв’язку

Для того, щоб реалізувати поставлену задачу, було додано до класу Controller нові методи згідно завдання та до класу Figure було додано поле target типу Figure.

Клас Controller зберігає методи для керування вектором фігур, що додаються на екран: add\_figures() , move\_figures\_and\_intersection\_check(), assign\_new\_targets() та remove\_figure(). У main для відповідних методів створюється потоки передаючі їч через лямбда та викликає методи класу.

У циклі створюється одна випадкова фігура та випадковим місцем розташування. Потік add\_thread викликається у цьому циклі.

Метод assign\_new\_targets() шукає об’єкти на екрані та випадковим чином призначає ціллю один із об’єктів, що є на екрані. Потік assign\_target\_thread викликає цей метод у main.

Метод move\_figures\_and\_intersection\_check() рухає об’єкти на екрані, у яких є ціль. Потік move\_thread викликає цей метод у метод у main.

Метод remove\_figure() видаляє об’єкти при зіткненні з ціллю, у яких є цілі. Потік remove\_intersection\_thread викликає цей метод у main.

У main виконується малювання об’єктів, що є у векторі figures (клас Controller). У методі main додано потік, що виконує керування об’єктами та малювання на екрані і паралельно цикл, в якому відбувається обробка подій користувача.

3. Вихідний текст програми розв’язку задачі

**Controller.cpp**

int Controller::random\_int(int min, int max) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dist(min, max);

return dist(gen);

}

void Controller::add\_figures() {

std::cout << "method add\_figures START\n";

figures\_mutex.lock();

std::cout << "LOOP START\n";

if (figures.size() < MAX\_FIGURES) {

Figure\* new\_figure\_first = prototype[random\_int(0, 4)]->clone();

new\_figure\_first->set\_position(random\_int(0, 550), random\_int(0, 550));

if (figures.size() < MAX\_FIGURES) {

figures.push\_back(new\_figure\_first);

}

std::cout << "Added new figure. Total figures: " << figures.size() << std::endl;

}

std::cout << "LOOP END\n";

figures\_mutex.unlock();

std::cout << "method add\_figures END\n";

}

void Controller::move\_figures\_and\_intersection\_check(){

std::cout << "method move\_figures\_and\_intersection\_check START\n";

if (figures.empty()){

std::cout << "CHECK figures.empty()\n";

return;

}

figures\_mutex.lock();

std::cout << "LOOP START\n";

for (auto& figure : figures) {

if (figure->get\_target() != nullptr) {

std::cout << "CHECK figure->get\_target() != nullptr\n";

auto target\_left = figure->get\_target()->get\_global\_bounds().left;

auto target\_top = figure->get\_target()->get\_global\_bounds().top;

auto target\_width = figure->get\_target()->get\_global\_bounds().width;

auto target\_height = figure->get\_target()->get\_global\_bounds().height;

figure->auto\_move(Vector2f(target\_left + target\_width / 2, target\_top + target\_height / 2));

figure->check\_intersection();

}

}

std::cout << "LOOP END\n";

figures\_mutex.unlock();

std::cout << "method move\_figures\_and\_intersection\_check END\n";

}

void Controller::assign\_new\_targets(){

std::cout << "method assign\_new\_targets START\n";

figures\_mutex.lock();

for (auto& figure : figures)

{

std::cout << "LOOP START\n";

if (figure->get\_target() == nullptr) {

std::cout << "CHECK figure->get\_target() == nullptr\n";

figure->set\_target(get\_random\_target(figure));

std::cout << "Set Target: "<< figure->get\_target()<<" name: " << figure->get\_target() << "\n";

}

std::cout << "LOOP END\n";

}

figures\_mutex.unlock();

std::cout << "method assign\_new\_targets END\n";

}

Figure\* Controller::get\_random\_target(Figure\* figure) {

if (figures.empty()) {

return nullptr;

}

auto target = figures[random\_int(0, figures.size() - 1)];

if (figure == target) {

return nullptr;

}

return target;

}

void Controller::remove\_figure() {

std::cout << "method remove\_figure START\n";

figures\_mutex.lock();

if (figures.empty()) {

figures\_mutex.unlock();

return;

}

for (auto & figure : figures){

std::cout << "LOOP START\n";

if (figure == nullptr) {

figures\_mutex.unlock();

return;

}

if (figure->check\_intersection()) {

std::cout << "CHECK igure->check\_intersection()\n";

std::cout << "Removed figure: " << figure->get\_figure\_name() <<"at memory: "<< figure << std::endl;

std::cout << "Removed figure TARGET: " << figure->get\_target()->get\_figure\_name() << "at memory: "<< figure->get\_target() << std::endl;

auto it = find(figures.begin(), figures.end(), figure->get\_target());

if (it != figures.end()) {

\*it = nullptr;

delete\* it;

figures.erase(it);

}

it = find(figures.begin(), figures.end(), figure);

if (it != figures.end()) {

\*it=nullptr;

delete\* it;

figures.erase(it);

}

}

std::cout << "LOOP END\n";

}

figures\_mutex.unlock();

std::cout << "method remove\_figure START\n";

}

**Main.cpp**

sf::Clock clock2;

float timer;

while (window.isOpen())

{

controller->event\_handler(window);

//controller->do\_thread();

window.clear(Color(239, 228, 176));

sf::Time elapsed2 = clock2.getElapsedTime();

timer = elapsed2.asMilliseconds();

if (timer > 1000)

{

std::thread add\_thread([&]() {

controller->add\_figures();

});

add\_thread.join();

}

std::thread move\_thread([&]() {controller->move\_figures\_and\_intersection\_check(); });

move\_thread.join();

std::thread remove\_intersection\_thread([&]() {controller->remove\_figure();});

remove\_intersection\_thread.join();

std::thread assign\_target\_thread([&]() {controller->assign\_new\_targets(); });

assign\_target\_thread.join();

for (auto& figure : controller->get\_figures())

{

if (figure != nullptr) {

figure->draw(window);

}

}

window.display();

}

4. Опис інтерфейсу (керівництво користувача)

Інтерфейс не передбачений у лабораторній

5. Опис тестових прикладів

Запускаємо програму та бачимо чистий екран (рис. 5.1).

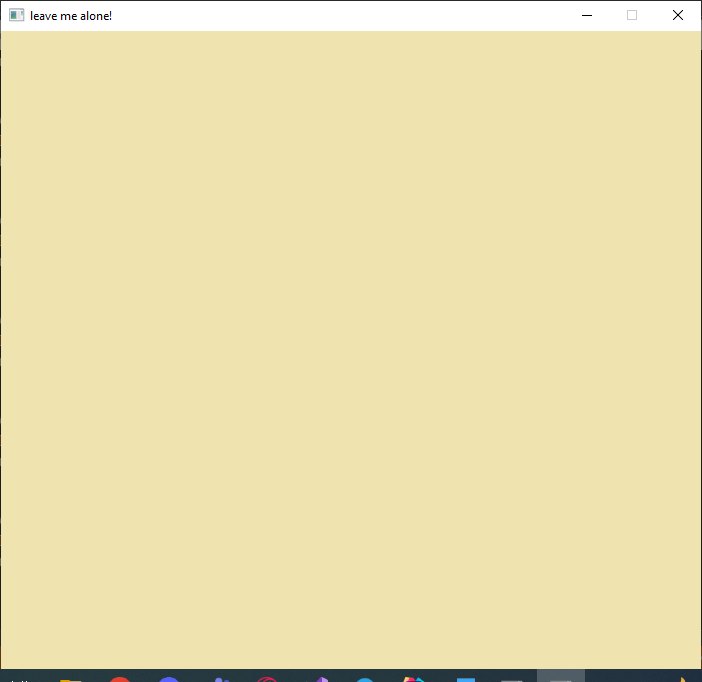


Рисунок 5.1 – Початковий екран

Через деякий проміжок часу додаватимуться нові фігури (рис. 5.2).

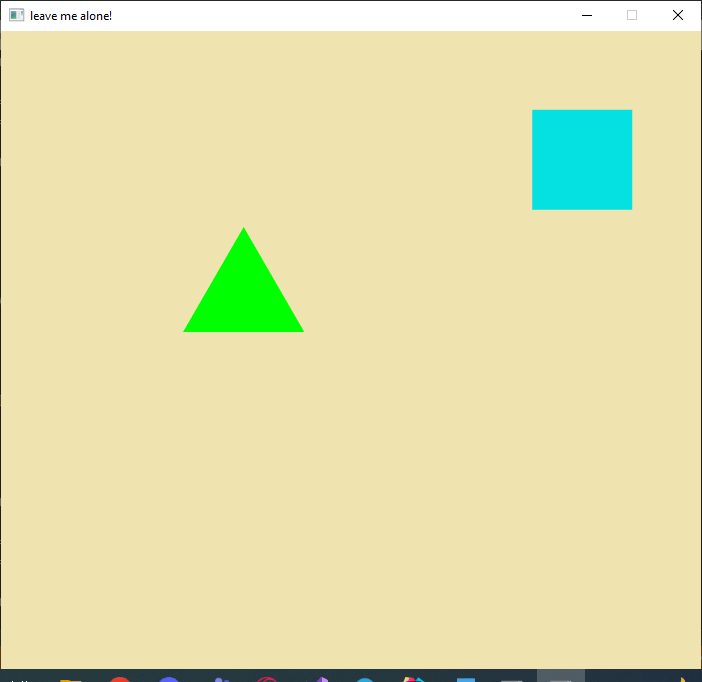


Рисунок 5.2 – Поява фігур на екрані

Результат переміщення фігур до своїх цілей (рис. 5.3) та після видалення (рис 5.4)

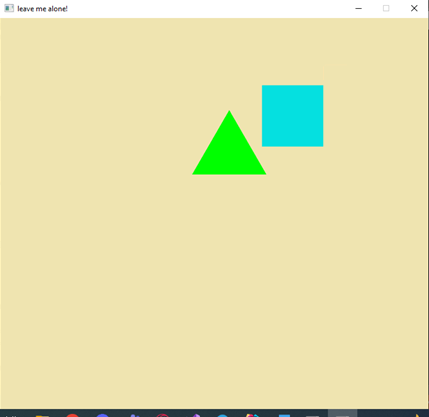


Рисунок 5.3 – Переміщення фігур

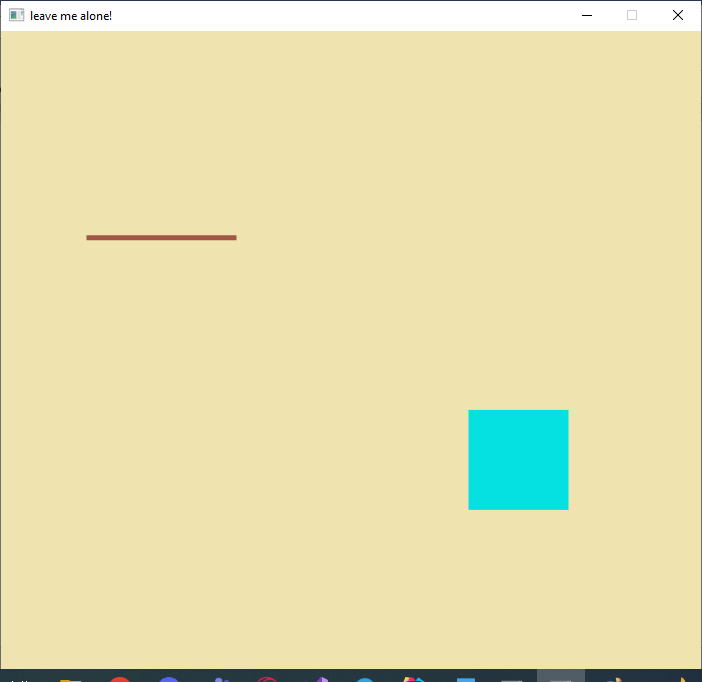


Рисунок 5.4 – Видалення фігур, що зіткнулись, поява інших фігур на екрані

На екрані створюється 10 фігур (рис 5.5).

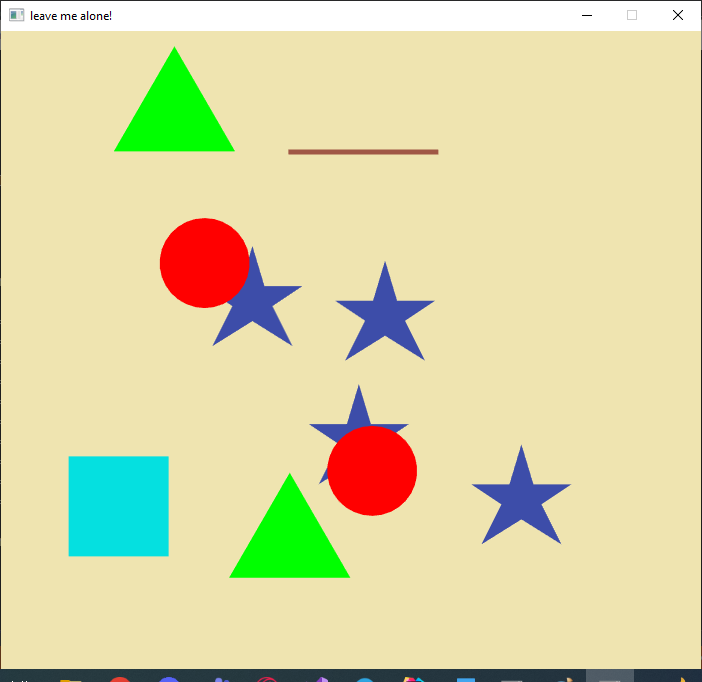


Рисунок 5.5 – Створення 10 фігур на екрані

Результат після видалення та переміщення фігур до своїх цілей (рис 5.6).

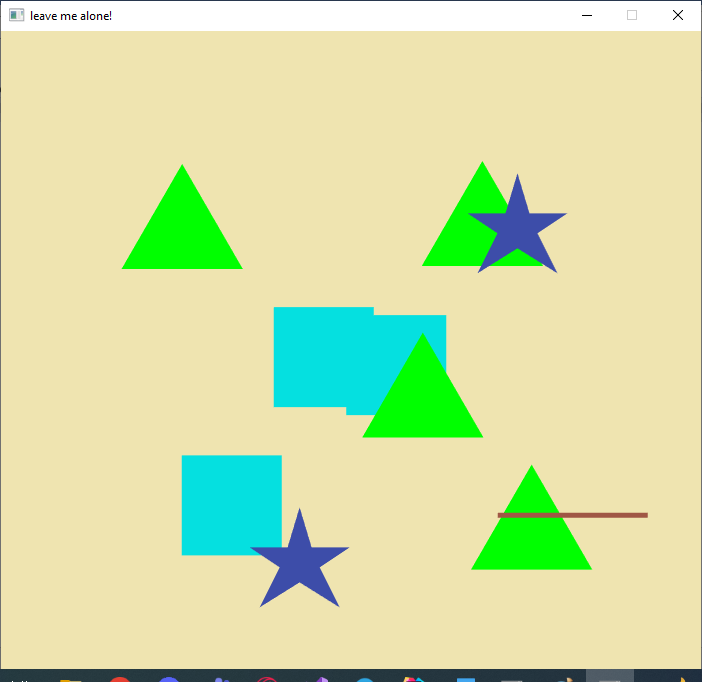


Рисунок 5.6 – Переміщення, видалення фігур, що зіткнулись, поява інших фігур на екрані

6. Аналіз помилок (опис усунення зауважень)

…