**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ**

**ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Звіт**

**з дисципліни Технологіх Розподілених Систем та Паралельних Обчислень**

**Лабораторна робота №2**

Виконав: студент 3 курсу, групи ІПЗ-4.04 спеціальності

121 Інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бухта М.М.

Перевірив\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Развалінов В.Ю.

**Одеса  2024**

**Опис завдання:**1. Створіть програму імітації руху більярдної кульки. Динаміка руху кульки задана класом Ball (див. лістинг). Розробіть графічний інтерфейс до програми для спостереження руху кульки.

2. Модифікуйте програму так, щоб кожна нова створювана кулька відтворювала свій рух в новому потоці. Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм.

3. Модифікуйте програму так, щоб кульки червоного створювались з вищим пріоритетом, ніж кульки синього кольору. Для створення кульок червоного та синього кольору створіть відповідні кнопки в графічному інтерфейсі користувача. Спостерігайте рух червоних та синіх кульок при збільшенні загальної кількості кульок. Поясніть результат спостереження. **Код програми:**

**main.cpp**

#include <QGuiApplication>

#include <QQmlApplicationEngine>

#include <QDebug>

#include <QQmlContext>

#include <QThreadPool>

#include <thread>

#include "ContextController.hpp"

#include "GameController.hpp"

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

{

QGuiApplication app(argc, argv);

quint32 max\_thread\_count = std::thread::hardware\_concurrency();

qInfo() << "Max threads count =" << max\_thread\_count;

QThreadPool::globalInstance()->setMaxThreadCount(max\_thread\_count);

QQmlApplicationEngine engine;

GameController game\_controller(&engine);

ContextController context\_controller(&engine);

auto root\_context = engine.rootContext();

root\_context->setContextProperty("game\_controller", &game\_controller);

root\_context->setContextProperty("context\_controller", &context\_controller);

QObject::connect(

&engine,

&QQmlApplicationEngine::objectCreationFailed,

&app,

[]() { QCoreApplication::exit(-1); },

Qt::QueuedConnection);

engine.loadFromModule("MultiballGame", "Main");

return app.exec();

}

**GameController.hpp**

#ifndef GAMECONTROLLER\_HPP

#define GAMECONTROLLER\_HPP

#include <QColor>

#include <QMutex>

#include <QMutexLocker>

#include <QObject>

#include <QQmlApplicationEngine>

#include <QQuickWindow>

#include <QSharedPointer>

#include <QVector>

#include "BallController.hpp"

class GameController : public QObject

{

Q\_OBJECT

public:

explicit GameController(QQmlApplicationEngine \*parent);

public slots:

void start();

void create\_ball(QColor color, qint32 coord\_x, qint32 coord\_y, QPoint direction = {});

private slots:

void update\_balls();

void ball\_position\_updated(QPoint new\_pos, BallController \*sender);

private:

void init();

// return new direction for the item. In case if is not out of bound, return {0, 0};

QPoint calculate\_direction(QPoint point, QSize size);

// return new pos;

QPoint fix\_out\_of\_bound\_pos(QPoint pos, QSize size);

private:

QQmlApplicationEngine &\_engine;

QVector<QSharedPointer<BallController>> \_balls;

QMutex \_balls\_mutex;

QPointer<QQuickWindow> \_main\_window;

};

#endif // GAMECONTROLLER\_HPP

**GameController.cpp**

#include "GameController.hpp"

#include "BallController.hpp"

#include <QDebug>

#include <QPointer>

#include <QThreadPool>

#include <QTimer>

GameController::GameController(QQmlApplicationEngine \*parent)

: QObject{parent}

, \_engine(\*parent)

{

qInfo() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] called";

}

void GameController::start() {

qInfo() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] called";

init();

QPointer<QTimer> timer = new QTimer(this);

connect(timer.data(), &QTimer::timeout, this, &GameController::update\_balls);

timer->start(1000 / 60); // 60 fps;

}

void GameController::create\_ball(QColor color, qint32 coord\_x, qint32 coord\_y, QPoint direction)

{

qInfo() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] color = " << color << "coord = {" << coord\_x << ";"

<< coord\_y << "}";

QSharedPointer<BallController> ball;

if (direction == QPoint{0, 0}) {

ball.reset(new BallController(\_engine, {coord\_x, coord\_y}, this));

} else {

ball.reset(new BallController(\_engine, {coord\_x, coord\_y}, direction, this));

}

connect(ball.get(),

&BallController::position\_updated,

this,

&GameController::ball\_position\_updated,

Qt::QueuedConnection);

QMutexLocker locker(&\_balls\_mutex);

\_balls.push\_back(ball);

ball->init(color, \_main\_window);

ball->set\_name\_number(\_balls.size() - 1);

qDebug() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] new ball on address " << \_balls.last();

}

void GameController::update\_balls()

{

QMutexLocker locker(&\_balls\_mutex);

for (auto ball : \_balls) {

QThreadPool::globalInstance()->start([ball]() { ball->run(); });

}

}

void GameController::ball\_position\_updated(QPoint new\_pos, BallController \*sender)

{

auto ball = qobject\_cast<BallController \*>(QObject::sender());

qInfo() << \_\_PRETTY\_FUNCTION\_\_ << "new pos:" << new\_pos << "; sender = " << ball;

if (ball) {

auto fixed\_pos = fix\_out\_of\_bound\_pos(new\_pos, ball->size());

ball->move(fixed\_pos);

auto new\_direction = calculate\_direction(new\_pos, ball->size());

if (new\_direction != QPoint{0, 0}) {

ball->set\_dx(new\_direction.x());

ball->set\_dy(new\_direction.y());

}

ball->draw();

} else {

qWarning() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] Unknown sender on address" << ball;

}

}

void GameController::init()

{

qInfo() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] called";

\_main\_window = qobject\_cast<QQuickWindow \*>(\_engine.rootObjects().first());

if (!\_main\_window) {

qFatal() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] Cannot open main window. Stop app";

}

}

QPoint GameController::calculate\_direction(QPoint point, QSize size)

{

auto window\_size = \_main\_window->size();

QPoint new\_direction{0, 0};

if (point.x() <= 0) {

new\_direction.setX(1);

} else if (point.x() + size.width() >= window\_size.width()) {

new\_direction.setX(-1);

}

if (point.y() <= 0) {

new\_direction.setY(1);

} else if (point.y() + size.height() >= window\_size.height()) {

new\_direction.setY(-1);

}

return new\_direction;

}

QPoint GameController::fix\_out\_of\_bound\_pos(QPoint pos, QSize size)

{

QPoint new\_pos{pos};

auto window\_size = \_main\_window->size();

if (pos.x() < 0) {

new\_pos.setX(0);

} else if (pos.x() + size.width() >= window\_size.width()) {

new\_pos.setX(window\_size.width() - 1 - size.width());

}

if (pos.y() < 0) {

new\_pos.setY(0);

} else if (pos.y() + size.height() >= window\_size.height()) {

new\_pos.setY(window\_size.height() - 1 - size.height());

}

return new\_pos;

}

**ContextController.hpp**

#ifndef CONTEXTCONTROLLER\_HPP

#define CONTEXTCONTROLLER\_HPP

#include <QMap>

#include <QObject>

#include <QQmlApplicationEngine>

#include <QQuickItem>

#include <QQuickView>

#include <QString>

#include <memory>

class ContextController : public QObject

{

Q\_OBJECT

public:

enum class GAME\_PAGE { GAME\_MENU = 0, GAME\_VIEW = 1 };

public:

explicit ContextController(QQmlApplicationEngine \*parent = nullptr);

signals:

public slots:

void on\_qml\_loaded();

void request\_to\_open\_page(GAME\_PAGE page);

private:

void init\_page\_loader();

private:

QQmlApplicationEngine &\_engine;

std::unique\_ptr<QQuickView> \_page\_controller;

QPointer<QObject> \_page\_loader;

QHash<GAME\_PAGE, QString> \_page\_associate;

};

#endif // CONTEXTCONTROLLER\_HPP

**ContextController.cpp**

#include "ContextController.hpp"

#include <QDebug>

#include <QQmlContext>

#include <QSharedPointer>

#include <QThread>

#include <QTimer>

#include <QUrl>

ContextController::ContextController(QQmlApplicationEngine \*parent)

: QObject{parent}

, \_engine(\*parent)

{

\_page\_associate = {{GAME\_PAGE::GAME\_MENU, "qrc:/GameMenuPage.qml"},

{GAME\_PAGE::GAME\_VIEW, "qrc:/GameViewPage.qml"}};

\_page\_controller = std::make\_unique<QQuickView>();

}

void ContextController::on\_qml\_loaded()

{

init\_page\_loader();

request\_to\_open\_page(GAME\_PAGE::GAME\_MENU);

}

void ContextController::request\_to\_open\_page(GAME\_PAGE page)

{

qInfo() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] page = " << \_page\_associate[page] << " ("

<< static\_cast<qint32>(page) << ") requested";

if (\_page\_loader) {

qInfo() << "Page loader object found.";

QMetaObject::invokeMethod(\_page\_loader, "load\_page", Q\_ARG(QVariant, \_page\_associate[page]));

} else {

qWarning() << "PageManager object not found!";

}

}

void ContextController::init\_page\_loader()

{

if (!\_page\_loader) {

qInfo() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] init page stack view object";

auto rootObjects = \_engine.rootObjects();

if (rootObjects.isEmpty()) {

qWarning() << "No root objects found!";

} else {

\_page\_loader = rootObjects.first()->findChild<QObject \*>("page\_loader");

}

if (!\_page\_loader) {

qWarning() << "Page loader object not found!";

}

}

}

**BallController.hpp**

#ifndef BALLCONTROLLER\_HPP

#define BALLCONTROLLER\_HPP

#include <QObject>

#include <QPoint>

#include <QQmlApplicationEngine>

#include <QQuickItem>

#include <QQuickWindow>

#include <QRunnable>

class BallController : public QObject, public QRunnable

{

Q\_OBJECT

public:

explicit BallController(QQmlApplicationEngine &engine,

QPoint start\_pos,

QObject \*parent = nullptr);

explicit BallController(QQmlApplicationEngine &engine,

QPoint start\_pos,

QPoint direction,

QObject \*parent = nullptr);

~BallController();

void run() override;

void init(QColor &color, QPointer<QQuickWindow> window);

void move();

void move(QPoint new\_pos);

QSize size();

void set\_dx(qint8 val);

void set\_dy(qint8 val);

void set\_name\_number(qint32 val);

public slots:

void draw();

signals:

void position\_updated(QPoint new\_pos, BallController \*sender);

private:

void generate\_random\_direction();

private:

static const qint32 \_MOVE\_STEP\_SIZE;

QQmlApplicationEngine &\_engine;

QPointer<QQuickItem> \_ball;

QPoint \_ball\_coords;

qint8 \_dx;

qint8 \_dy;

};

#endif // BALLCONTROLLER\_HPP

**BallController.cpp**

#include "BallController.hpp"

#include <QDebug>

#include <QQmlComponent>

#include <QQuickWindow>

#include <QRandomGenerator>

const qint32 BallController::\_MOVE\_STEP\_SIZE = 10;

BallController::BallController(QQmlApplicationEngine &engine, QPoint start\_pos, QObject \*parent)

: \_engine{engine}

, \_ball\_coords{start\_pos}

, QObject{parent}

{

generate\_random\_direction();

}

BallController::BallController(QQmlApplicationEngine &engine,

QPoint start\_pos,

QPoint direction,

QObject \*parent)

: \_engine{engine}

, \_ball\_coords{start\_pos}

, QObject{parent}

{

\_dx = static\_cast<qint8>(direction.x());

\_dy = static\_cast<qint8>(direction.y());

}

BallController::~BallController()

{

qInfo() << \_\_PRETTY\_FUNCTION\_\_ << "destructor called";

\_ball->deleteLater();

}

void BallController::init(QColor &color, QPointer<QQuickWindow> window)

{

QQmlComponent component(&\_engine, QUrl(QStringLiteral("qrc:/BilliardBall.qml")));

if (component.isError()) {

qWarning() << "[" << \_\_PRETTY\_FUNCTION\_\_

<< "] Error loading QML component:" << component.errors();

return;

}

\_ball = qobject\_cast<QQuickItem \*>(component.create());

if (!\_ball) {

qWarning() << "[" << \_\_PRETTY\_FUNCTION\_\_ << "] Failed to create QML object.";

return;

}

\_ball->setParentItem(window->contentItem());

\_ball->setX(\_ball\_coords.x());

\_ball->setY(\_ball\_coords.y());

\_ball->setProperty("ball\_color", color);

}

void BallController::run()

{

qInfo() << \_\_PRETTY\_FUNCTION\_\_ << "called";

move();

}

void BallController::move()

{

QPoint direction{\_MOVE\_STEP\_SIZE \* \_dx, \_MOVE\_STEP\_SIZE \* \_dy};

\_ball\_coords += direction;

emit position\_updated(\_ball\_coords, this);

}

void BallController::move(QPoint new\_pos)

{

\_ball\_coords = new\_pos;

// emit position\_updated(\_ball\_coords, this);

}

QSize BallController::size()

{

return \_ball->size().toSize();

}

void BallController::set\_dx(qint8 val)

{

if (val)

\_dx = val;

}

void BallController::set\_dy(qint8 val)

{

if (val)

\_dy = val;

}

void BallController::set\_name\_number(qint32 val)

{

\_ball->setProperty("ball\_number", val);

}

void BallController::draw()

{

\_ball->setX(\_ball\_coords.x());

\_ball->setY(\_ball\_coords.y());

}

void BallController::generate\_random\_direction()

{

\_dx = QRandomGenerator::global()->bounded(-1, 1);

\_dy = QRandomGenerator::global()->bounded(-1, 1);

if (\_dx == 0) {

\_dx = 1;

}

if (\_dy == 0) {

\_dy = 1;

}

}

**Main.qml**

import QtQuick

import QtQuick.Controls

Window {

width: 640

height: 480

visible: true

title: qsTr("Hello World")

Loader {

id: page\_loader

objectName: "page\_loader"

function load\_page(url) {

page\_loader.source = url

console.log("page loader source", page\_loader.source);

}

}

Component.onCompleted: {

console.log("Requesting page change with delay.");

Qt.callLater(function() {

game\_controller.start();

context\_controller.on\_qml\_loaded();

}, 0);

}

}

**GameViewPage.qml**

import QtQuick

import QtQuick.Controls

Page {

width: 640

height: 480

visible: true

MouseArea {

id: stack\_mouse\_area

anchors.fill: parent

acceptedButtons: Qt.LeftButton | Qt.RightButton

onClicked: {

var ball\_width = 50;

var ball\_height = 50;

var ball\_color;

if (mouse.button === Qt.LeftButton) {

ball\_color = "red";

} else {

ball\_color = "blue";

}

game\_controller.create\_ball(ball\_color, (parent.width - ball\_width) / 2, (parent.height - ball\_height) / 2);

}

}

Text {

text: "Press left mouse button to spawn blue ball."

font.pixelSize: 24

x: 10

y: 200

}

Text {

text: "Press right mouse button to spawn blue ball."

font.pixelSize: 24;

x: 10

y: 400

}

Component.onCompleted: {

start\_game();

}

function start\_game() {

var ball\_width = 50;

var ball\_height = 50;

console.log("start\_game called. window {width; height} = {", width, "; ", height, "}; ball {width; height} = {", ball\_width, "; ", ball\_height, "}");

game\_controller.create\_ball("red", (parent.width - ball\_width) / 2, (parent.height - ball\_height) / 2);

}

}

**GameMenuPage.qml**

import QtQuick

import QtQuick.Controls

Page {

width: 640

height: 480

visible: true

Text {

text: "Press any key to start simulation"

font.pixelSize: 30

font.bold: true

anchors.centerIn: parent

transform: Translate {y: -20}

}

Text {

text: "Make by Bukhta Mykyta, ІПЗ-4.03"

anchors.centerIn: parent

font.pixelSize: 20

transform: Translate {y: +20}

}

MouseArea {

id: stack\_mouse\_area

anchors.fill: parent

acceptedButtons: Qt.AllButtons

onClicked: {

start\_game();

}

}

Keys.onPressed: {

start\_game();

}

function start\_game() {

console.log("pressed");

context\_controller.request\_to\_open\_page(ContextHandler.GAME\_PAGE.GAME\_VIEW);

}

}

**ContextHandler.qml**

import QtQuick

Item {

enum GAME\_PAGE {

GAME\_MENU = 0,

GAME\_VIEW = 1

}

}

**BilliardBall.qml**

import QtQuick

Item {

property int ball\_number: 0

property color ball\_color: "white"

width: 50

height: 50

Rectangle {

anchors.fill: parent

radius: width

color: ball\_color

border.width: 1

Text {

text: ball\_number

anchors.centerIn: parent

}

}

}

**resources.qrc**

<RCC>

<qresource prefix="/">

<file>Main.qml</file>

<file>BilliardBall.qml</file>

<file>GameMenuPage.qml</file>

<file>ContextHandler.qml</file>

<file>GameViewPage.qml</file>

</qresource>

<qresource prefix="/resources"/>

</RCC>

**Результат виконання програми:**

Рисунок 1 – стартовий екран

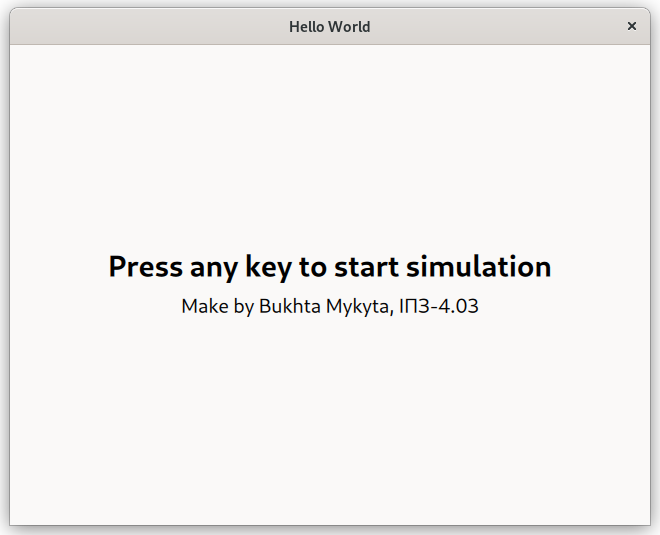


Рисунок 2 – скрін симуляції

