|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное  учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
|  |
|  |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ПРОЕКТНОЙ РАБОТЕ** | |
| «Разработка игрового приложения «...»» | |
| **по дисциплине**  **«**ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ**»** | |
| Выполнил студент группы *ИКБО-09-22* | *Кузнецов Я. А.* |
| Принял |  |

|  |  |  |
| --- | --- | --- |
| Проектная | « » 2022 г. |  |
| работа выполнена |  |  |
| «Зачтено» | « » 2022 г. |  |

Москва 2022

**СОДЕРЖАНИЕ**

[1 ПОСТАНОВКА ЗАДАЧИ 3](#_Toc6)

[2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ 4](#_Toc7)

[2.1 Восстановленная таблица истинности 4](#_Toc8)

[2.2 Реализация 5](#_Toc9)

[ВЫВОДЫ 7](#_Toc10)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 8](#_Toc11)

**1 ВВЕДЕНИЕ**

...

2 ПОСТАНОВКА ЗАДАЧИ

Разработать программный продукт со следующими возможностями:

**2.1 Геймплей**

* Игрок управляет платформой и может двигать её в право и влево.
* Шарик летает по игровому полю. Игрок должен отбивать его платформой. Если шарик коснётся края экрана за платформой он разрушится.
* Если на игровом поле не останется шариков игрок проиграл.
* На игровом поле располагаются цели, которые игроку требуется поразить шариком. Из них могут выпадать случайные усиления, которые влияют на геймплей.
* Чтобы активировать усиление игрок должен соприкоснутся с ним платформой.
* Игрок побеждает если разрушит все цели на уровне и переходит к следующему уровню.

**2.2 Игровые меню**

* Меню выбора уровней.
* Главное меню в котором игрок может:
  + Начать новую игру или вернутся к не завершенной.
  + Открыть меню выбора уровней.
  + Открыть меню с информацией о программе.
  + Выйти из игры
* Меню с информацией о программе.

**3 РЕАЛИЗАЦИЯ**

**3.1 Структура проекта**

Весь весь код в проекте разделён на заголовочные файлы в зависимости от функционала. В папке «GUI» хранится код элементов имеющих графический интерфейс. Заголовочные файлы в корне проекта не несут в себе графического интерфейса. Они отвечают за расчёты, работу с файловой системой, хранение глобальных переменных и т.п. .В папках «icons», «levels», «ui\_design\_files» хранятся файлы иконок, уровней и файлов интерфейсов соответственно.

**3.2 Файлы уровней**

Файл с данными уровня представляет из себя текстовый файл с матрицей из целых чисел произвольного размера. Модуль числа обозначает прочность платформы, а знак показывает спрятан ли в ней бонус или нет(Если есть минус то есть бонус, в противном случае его нет). Название файла с уровнем должно заканчиваться на «.level».

**3.3 Инициализация и главный цикл**

Главная функция main отвечает за инициализацию компонентов и запуск главного игрового цикла посредствам вызова процедуры enter\_main\_loop .Её логику можно описать следующей блок-схемой(см рисунок 1.).

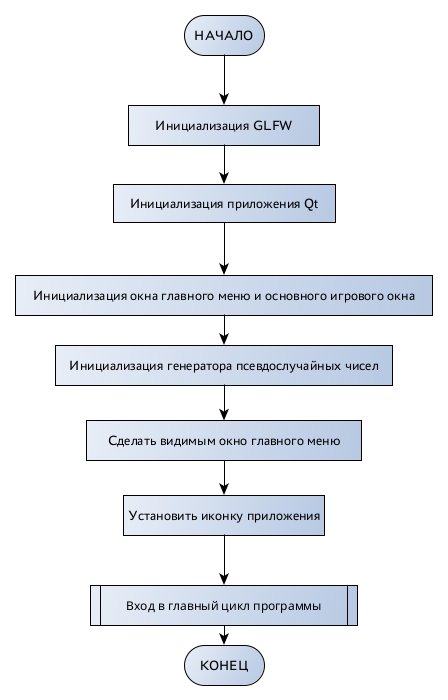


Рисунок 1 – Блок-схема главной функции main

Подробнее рассмотрим главный игровой цикл. В начале мы получаем текущее время и рассчитываем минимальное допустимое время для выполнения итерации. Это нужно для стабилизации частоты кадров в секунду и более плавного движения игровых объектов. После, вызываем обработчик событий Qt. Рассчитываем следующую итерацию игры для текущего игрового окна, посредствам вызова метода new\_game\_iteration. Проверяем, закрыты ли все основные окна и если это так завершаем работу программы. В противном случае отмечаем эту итерацию как выполненную и дожидаемся истечения целевого времени итерации. Этот цикл выполняется на протяжении всей работы программы. Этот алгоритм отражает блок-схема на рисунке 2.

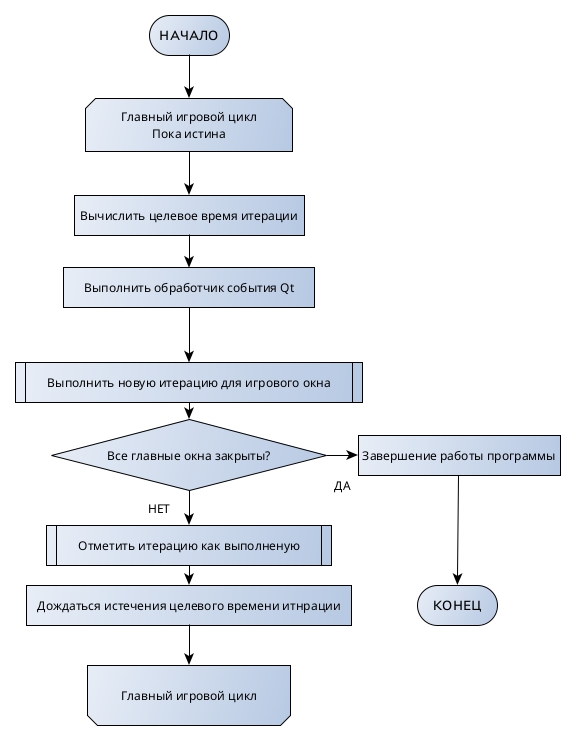


Рисунок 2 – Блок-схема главного игрового цикла

Рассмотрим код файла main.cpp, в котором реализованы вышеупомянуты функция main и процедура enter\_main\_loop. (см рис. 3, 4)

|  |
| --- |
| **void enter\_main\_loop()**  **{**    **while (true){**  **auto now = std::chrono::steady\_clock::now();**  **auto end = now + std::chrono::milliseconds(**8**);**    **QApplication::processEvents();**    **if (!window->g\_window->isHidden()){**  **window->g\_window->new\_game\_iteration();**  **}**  **if (window->g\_window->isHidden() and window->isHidden())**  **exit(**0**);**    **next\_frame();**  **std::this\_thread::sleep\_until(end);**  **}**  **}** |

Рисунок 3 – Код на языке c++ для главного игрового цикла.

|  |
| --- |
| **int main(int argc, char \*argv[]) {**  **glfwInit(); *// Инициализация OpenGL и GLFW***  **glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);**  **glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);**  **glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);**  **#ifdef \_\_APPLE\_\_**  **glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); *// Для корректной работы под macOS***  **#endif**      **QApplication a(argc, argv);**  **window = new Ui\_MainWindow();**  **std::cout << "INIT" << std::endl;**  **window->g\_window = new GameWindow();**  **window->init();**  **init\_random();**  **window->g\_window->init(window);**  **std::cout << "INIT DONE" << std::endl;**    **window->show();**    **QApplication::quitOnLastWindowClosed();**    **auto\* icon = new QIcon("../icons/Ball\_triple\_ico.png");**  **QApplication::setWindowIcon(\*icon);**  **enter\_main\_loop();**  **return 0;**  **}** |

Рисунок 4 – Код на языке c++ для функции main.

**3.4 Заголовочный файл physics.hpp и физика.**

В этом файле содержится 2 реализации функции по проверки столкновения двух объектов родственных классу QWidget. 1 реализация принимает 2 указателя на объекты, а вторая сами эти объекты. Проверка столкновений осуществляется по следующему алгоритму.(см рис. 5)

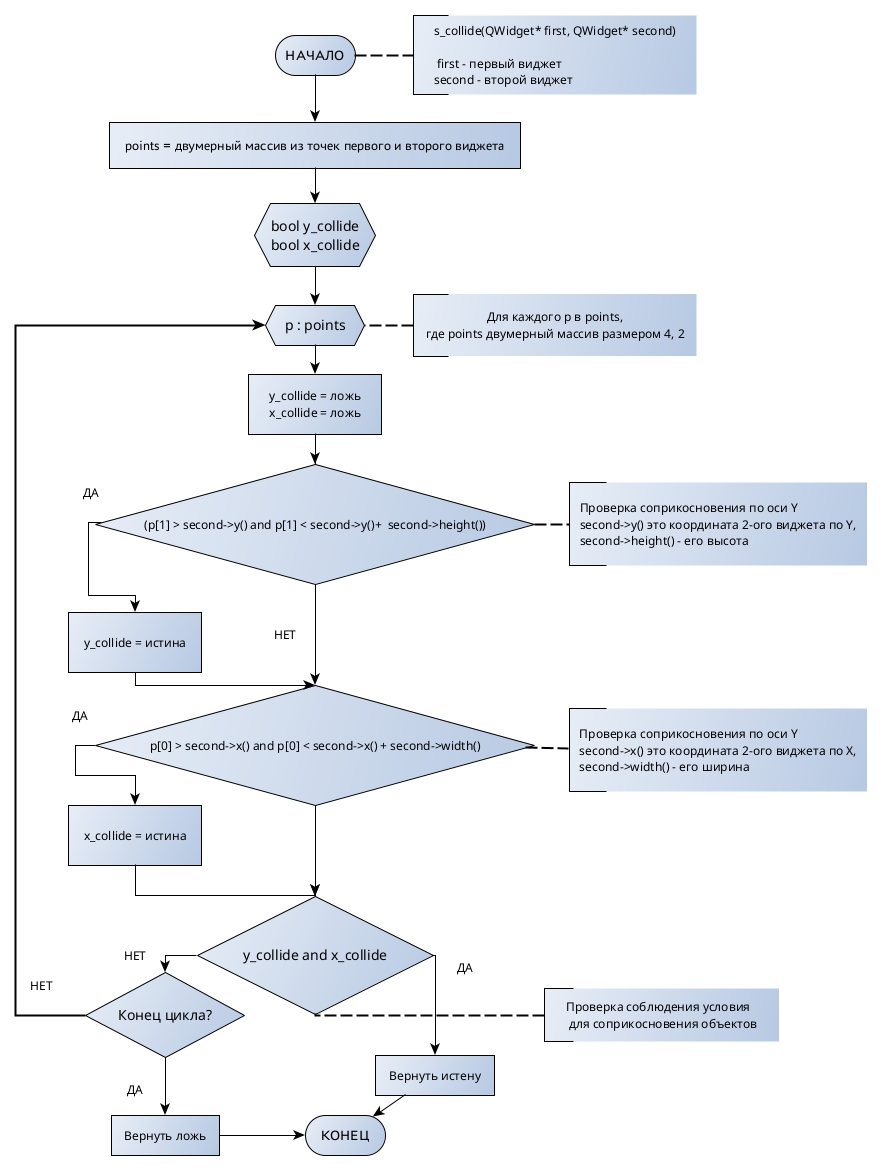


Рисунок 5 – Блок-схема алгоритма проверки столкновений двух виджетов

**3.5 Заголовочный файл random.hpp и генерация псевдослучайных чисел.**

В этом файле реализованы функции по генерации псевдослучайных чисел. Процедура init\_random задаёт ключ для генератора, в виде текущего времени. Функция randint(int a, int b) возвращает псевдослучайное число из диапазона [a, b]. Функция get\_random\_inversion возвращает -1.0 или 1.0 случайным образом. Рассмотрим их реализацию на языке c++.

|  |
| --- |
| **void** init\_random**(){**  **srand((unsigned)time(nullptr));**  **}**    **int** randint**(int a, int b){**  **int rnd = (rand()%b)+a;**  **if (rnd >= a and rnd <= b){**  **return rnd;**  **}**  **return a + rnd % (b - a +** 1**);**  **}**    **float** get\_random\_inversion**(){**  **int a = randint(**0**,** 128**);**  **if (a %** 2 **==** 0**){**  **return** 1.0f**;**  **}**  **return** -1.0f**;**  **}** |

Рисунок 6 – реализация вышеупомянутых функций на c++.

**3.5 Заголовочный файл fps\_control.hpp и контроль частоты кадров.**

В этом файле содержится процедура для вычисления частоты кадров в секунду и функция, которая возвращает текущую частоту кадров, а также глобальные переменные frames, current\_fps, lastTime. Функция get\_main\_thread\_fps возвращает значение current\_fps, если оно > 1, в противном случае 1. Рассмотрим алгоритм работы процедуры next\_frame с помощью блок-схемы(см. рис. 7).

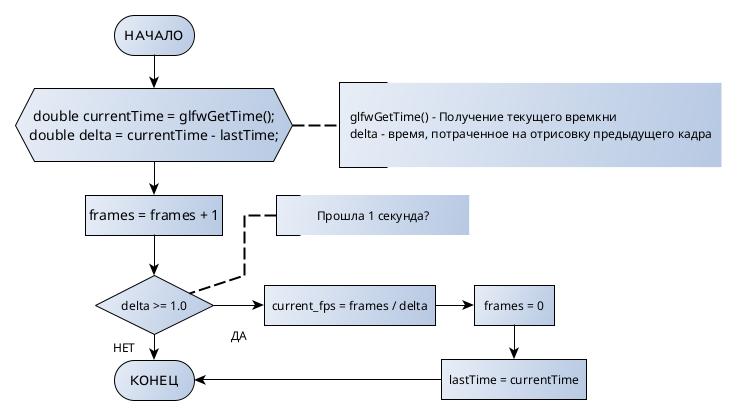


Рисунок 5 – Блок-схема алгоритма процедуры расчёта частоты кадров в секунду.

**3.6 Заголовочный файл level\_files\_handler.hpp и обработка файлов уровней.**

В данном файле реализованны функции для обработка файлов уровней. Функция is\_delimiter(char c), проверяет, является ли символ c разделителем. Функция split(std::string &str) разбивает строку str на массив чисел. get\_main\_levels\_count() и get\_bonus\_levels\_count() возврощают кол-во основных и дополнительных уровней соответственно.

**3.7 Реализация классов игровых объектов.**

Для удобства работы с игровыми объектами и хранения данных на потребуются классы блоков, которые нужно поразить игроку, бонусов, а также шарика. Все эти классы будут наследовать методы от QPushButton, чтобы ними можно было взаимодействовать как с кнопками и базовыми виджетами.

Начнём с класса Ball. Для этого реализуем с начала приватные метод и переменные: update\_qt\_pos() – метод для обновления отображаемых параметров (координаты и размер), а также координаты x, y, проекции скорости на координатные оси и id шарика. Остальные методы будут общедоступными так-как их потребуется вызывать в других частях программы.