|  |
| --- |
|  |
| 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.).

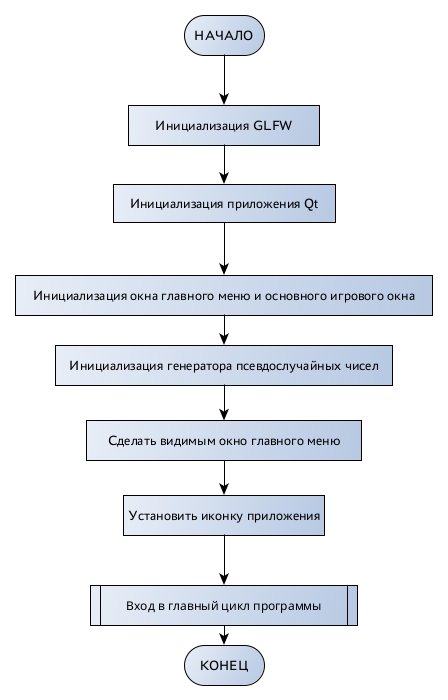


Рисунок 1 – Блок-схема главной функции main

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.

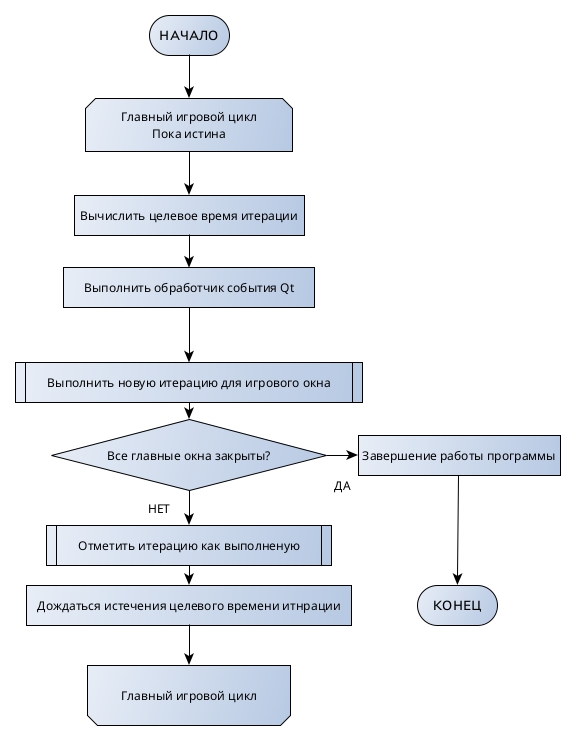


Рисунок 2 – Блок-схема главного игрового цикла

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)

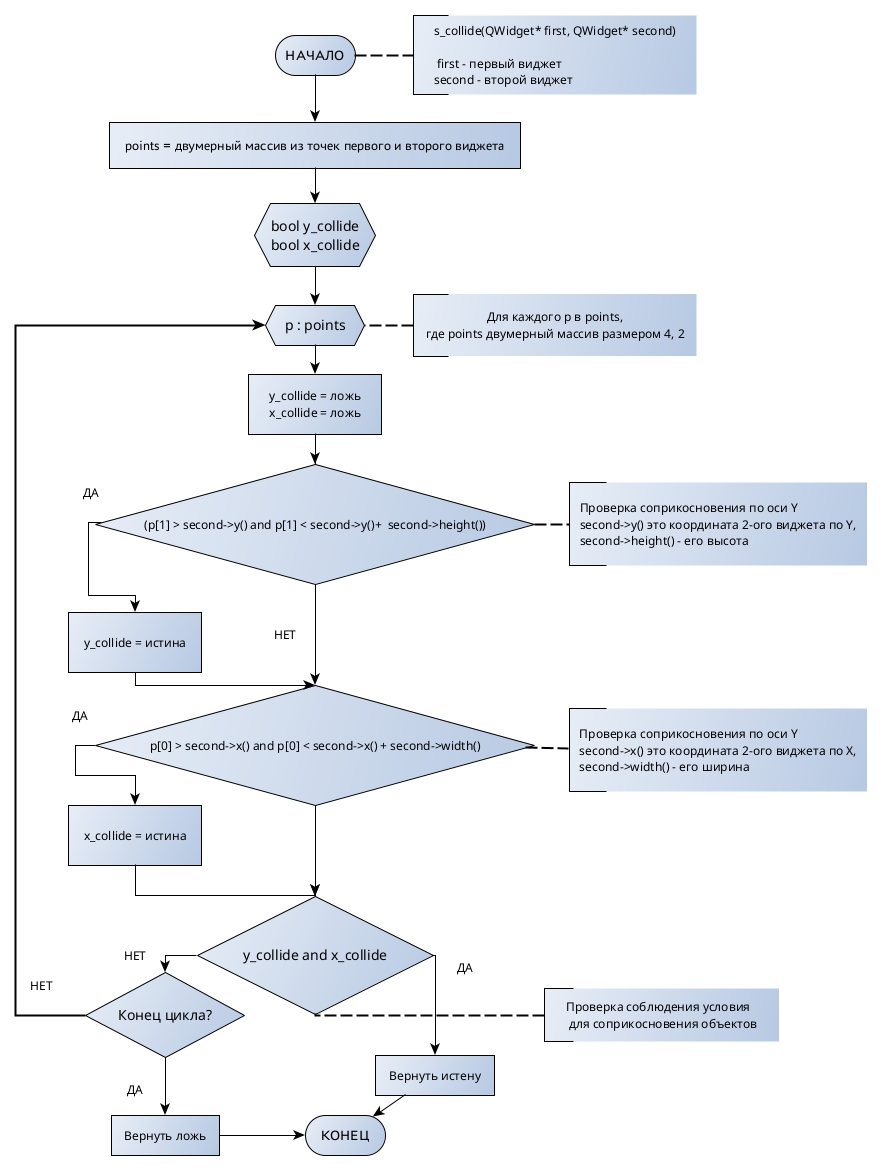


Рисунок 5 – Блок-схема алгоритма проверки столкновений двух виджетов

**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).

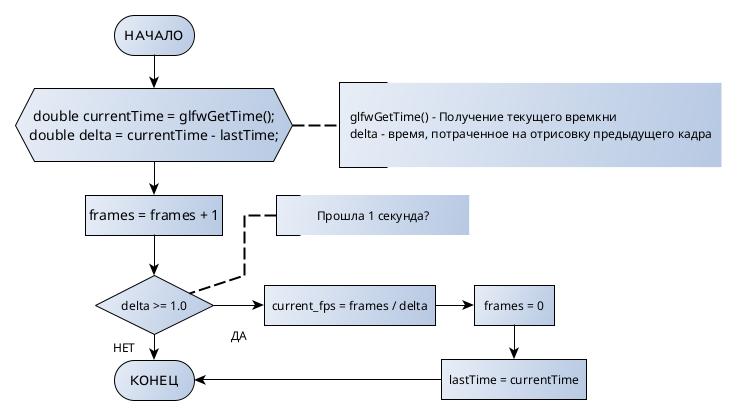


Рисунок 5 – Блок-схема алгоритма процедуры расчёта частоты кадров в секунду.

**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. У этого класса будут следующие публичные методы:

* void rest\_speed()
* void freeze\_ball()
* void set\_id(int id)
* void multiply\_ball\_speed(float m)
* int get\_id()
* float get\_x()
* float get\_y()
* void set\_y(float y)
* void set\_x(float x)
* void init(int x = 256, int y = 256)
* void change\_movement\_vector(int x, int y)
* void move()
* void process\_ball\_collisions(float platform\_x, float platform\_y, float platform\_w, float platform\_h, std::vector<std::vector<TargetBlock\*>\*>\* targets, QWidget\* GameSpace, std::vector<Bonus\*>\* bonuses, QWidget\* parent)
* void do\_balls\_collision(Ball\* other\_ball)
* void process\_ball\_collisions\_with\_other\_balls(std::vector<Ball\*>\* balls)
* void spawn\_on\_random\_good\_place(std::vector<Ball\*>\* balls)

Реализуем класс для кирпичиков, которые игроку нужно разрушить. У него будут следующие приватные переменные.

* unsigned int level = 1;
* unsigned int hp = 1;
* int column = 0, row = 0;
* bool bonus = false;

И следующие публичные методы:

* bool is\_bonus()
* void set\_bonus(bool status)
* bool hit\_block()
* void update\_color()
* bool is\_dead()
* void set\_level(int &\_level)
* void set\_hp(int \_hp)
* int get\_col()
* int get\_row()
* void set\_row(int \_row)
* void kill\_target()

Реализуем класс для бонусов. Ему потребуются следующие приватные методы:

* void set\_random\_type()
* void set\_random\_negative\_type()
* void update\_image()

И следующие публичные переменные и методы:

* bool to\_delete = false;
* float pos\_y;
* unsigned int bonus\_type;
* int get\_x()
* void init(unsigned int \_type = BONUS\_TYPE\_RANDOM)
* void move()

**3.7 Реализация классов окон.**

В для игры понадобится 4 игровых окна, для которых потребуется создать классы: окно с информацией о программе, главное меню, меню выбора уровня и главное игровое окно. В файле ./GUI/AboutForm.hpp реалтзован интерфейс окна с информацией.

Окно выбора уровня реализовано в файле ./GUI/LevelSelectMenu.hpp . Для этого окна потребуется реализовать следующие методы:

* onLevelButtonClicked(int button\_number) – вызывается при нажатии на кнопку с номером button\_number.
* void onBackButtonPushed() кнопка возврата в главное меню
* void setupUi(QWidget \*Level\_select) и void retranslateUi(QWidget \*Level\_select) – инициализация интерфейса.
* void init(GameWindow\* g\_win, QMainWindow\* main\_window) – инициализация
* void onPlayButtonPushed() – вызывается при нажатии на кнопку «играть»

Окно с главным меню реализовано в файле ./GUI/MainMenu.hpp . Для этого окна потребуется реализовать несколько методов, а также переопределить несколько базовых :

* void onExitButtonPush() - вызывается когда нажата кнопка выхода из игры
* void onLevelSelectButtonPushed() – вызывается когда нажата кнопка перехода в меню выбора уровня.
* void init() - инициализация
* void onNewGameButtonPushed() – вызывается при нажатии на кнопку начала новой игры или «продолжить».
* void onCustomLevelPlayPushed() - вызывается при нажатии на кнопку запуска пользовательского уровня.
* void setupUi() – инициализация графического интерфейса.
* void retranslateUi(Ui\_MainWindow \*MainWindow) – инициализация графического интерфейса.
* bool eventFilter(QObject \*object, QEvent \*event) – переопределение стандартного фильтра событий.
* void create\_new\_demo\_game() – запуск 1 уровня.

Окно с главным игровым окном реализовано в файле ./GUI/GameWindow.hpp . За него будет отвечать класс GameWindow. Для этого окна потребуется следующие методы:

* void setupUi(QWidget \*AboutForm) и void retranslateUi(QWidget \*AboutForm) – инициализация интерфейса.
* bool eventFilter(QObject \*object, QEvent \*event) – переопределённый фильтр событий.
* void move\_platform() – сдвинуть платформу по текущему направлению.
* void onResize() – метод, который вызывается при изменении размера окна.
* void wipe\_targets\_data() – очистка данных кирпичиков.
* bool is\_lose() – проверка проигрыша.
* bool is\_win() – проверка победы.
* void check\_win\_or\_lose() проверка победы или поражения и соответствующие действия при их обнаружении.
* void load\_level\_data(std::string &file\_path, int \_level\_number = 0, bool \_is\_bonus\_level = false) – загрузка уровня по полученным параметрам (приватный метод).
* void apply\_bonus\_triple\_ball(bool \_recursion = false) – обработка бонуса утроения шарика.
* void play\_level(unsigned int level\_id, bool bonus = false) – подготовка информации и запуск уровня в случае успешной проверки полученных данных.
* void play\_level\_by\_path(std::string \_filepath) – быстрый запуск уровня по пути к файлу.
* void update\_score() – обновить отображаемый счёт.
* void show() – переопределённый метод перевода окна из скрытого в видимое.
* void resizeEvent(QResizeEvent\* event) override – переопределённый метод обработки события изменения размеров окна, вызывает onResize.
* void hide() – переопределённый метод скрытия окна.
* void init(auto\* window) – инициализация окна.
* void new\_game\_iteration() – обработка новой итерации игры.
* bool check\_bonus\_collisions(Bonus\* bonus) – проверка столкновения бонуса и платформы.
* void rest\_game() – сброс текущей игры, вызывает wipe\_targets\_data.