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

РТУ МИРЭА

ОТЧЕТ О ВЫПОЛНЕНИИ

ИНДИВИДУАЛЬНОЙ ПРОЕКТНОЙ РАБОТЕ

«Разработка игрового приложения «Arcanoid»»

по дисциплине
«ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ»

Выполнил студент группы *ИКБО-09-22* *Кузнецов Я. А.*

Приняла преподаватель *Евстигнеева О. А.*

Проектная работа выполнена «__»_____2022 г. _____

Оценка «__»_____2022 г. _____

Москва 2022

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Идея написания написания этой игры возникла после предложения моего преподавателя сделать проектную работу, так как простые задачи по предмету были простые и скучные. Я написал данное приложении по следующим причинам:

1. Обучение.

Мне хотелось изучить новые технологии для создания приложений с графическим интерфейсом (фреймворк Qt6), а также аспекты языка c++ не затронутые в учебной программе.

2. Это классика.

Старые 2D игры отличный источник вдохновения. Их реализация не настолько сложна и отлично подходит чтобы попрактиковаться.

3. Игры это не только про MS Windows.

На данный момент не многие игры поддерживают Linux системы, упуская возможность порадовать их пользователей. Я собираюсь улучшить сложившуюся ситуацию, ведь в Linux системах есть масса возможностей для создания качественных высокопроизводительных игр.

4. Выгодно для меня.

Эту работу можно разместить в портфолио, чтобы впечатлять будущего работодателя.

При создании этой игры я вдохновлялся классической игрой «Арканоид», откуда и взял идею и некоторые механики.

2 ПОСТАНОВКА ЗАДАЧИ

Разработать игровое приложение со следующими особенностями и возможностями:

2.1 Особенности реализации

- Использование языка c++ в качестве ведущего для данного приложения.
- Использование фреймворка Qt6.
- Использование системы сборки CMake для возможности сборки на любой платформе.
- Возможность сыграть в уровень, созданный пользователем.
- Сторонние разделяемые библиотеки нельзя распространять с исходным кодом и скомпилированным приложением!

2.2 Игровой процесс

- Цель игры выбить шаром как можно больше блоков. Задавать скорость движения шара игрок может управляя платформой.
- Платформа двигается в право и влево с помощью клавиш A, D или стрелок вправо, влево.
- Шарик(и) перемещается по игровому полю. Игрок должен отбивать его платформой. Если шарик коснётся края экрана, за платформой, он исчезнет.
- Если на игровом поле не останется шариков, игрок проигрывает.
- На игровом поле располагаются блоки в виде кирпичиков, 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 кирпичики и переходит к следующему уровню (если это возможно).

2.3 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 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. Разделитель табуляция, пробел или «;». 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.).

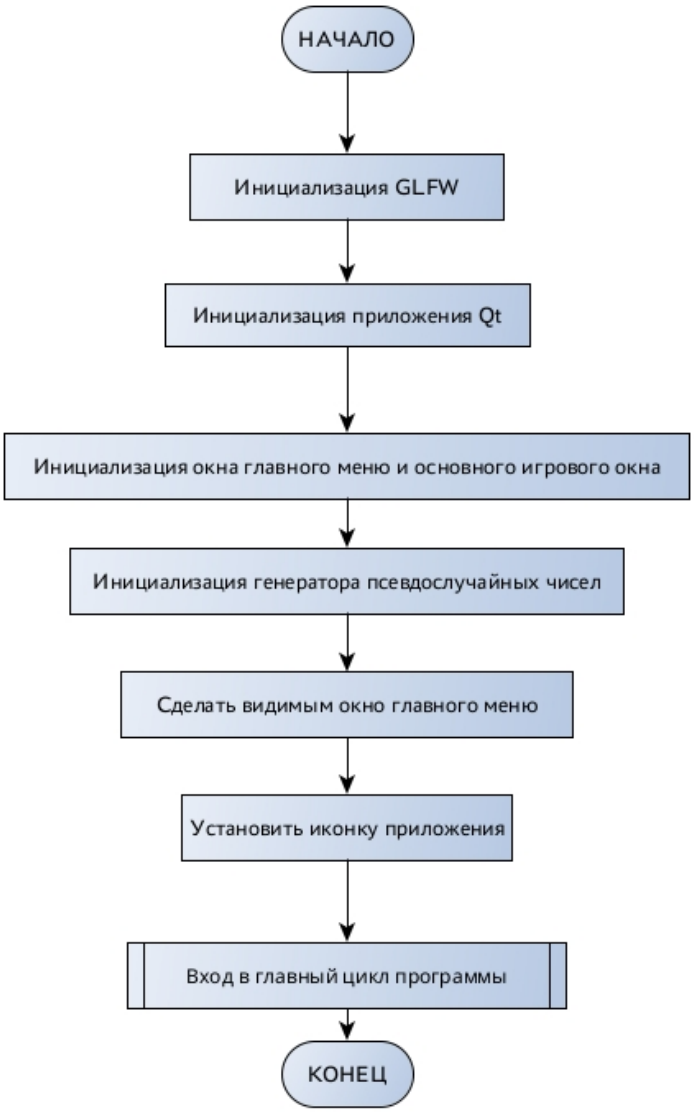

Рисунок 1 – Блок-схема главной функции main

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.

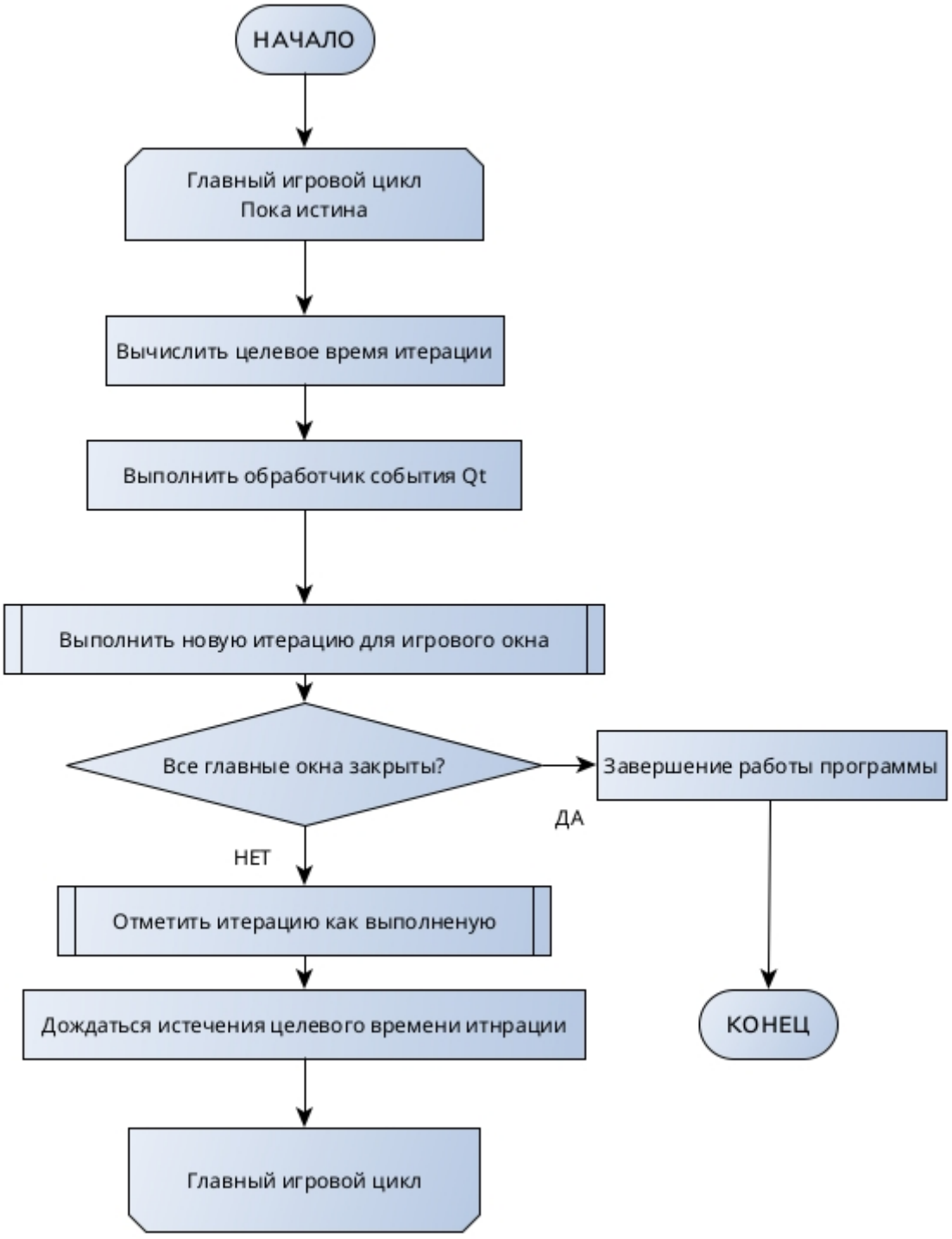

Рисунок 2 – Блок-схема главного игрового цикла

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)

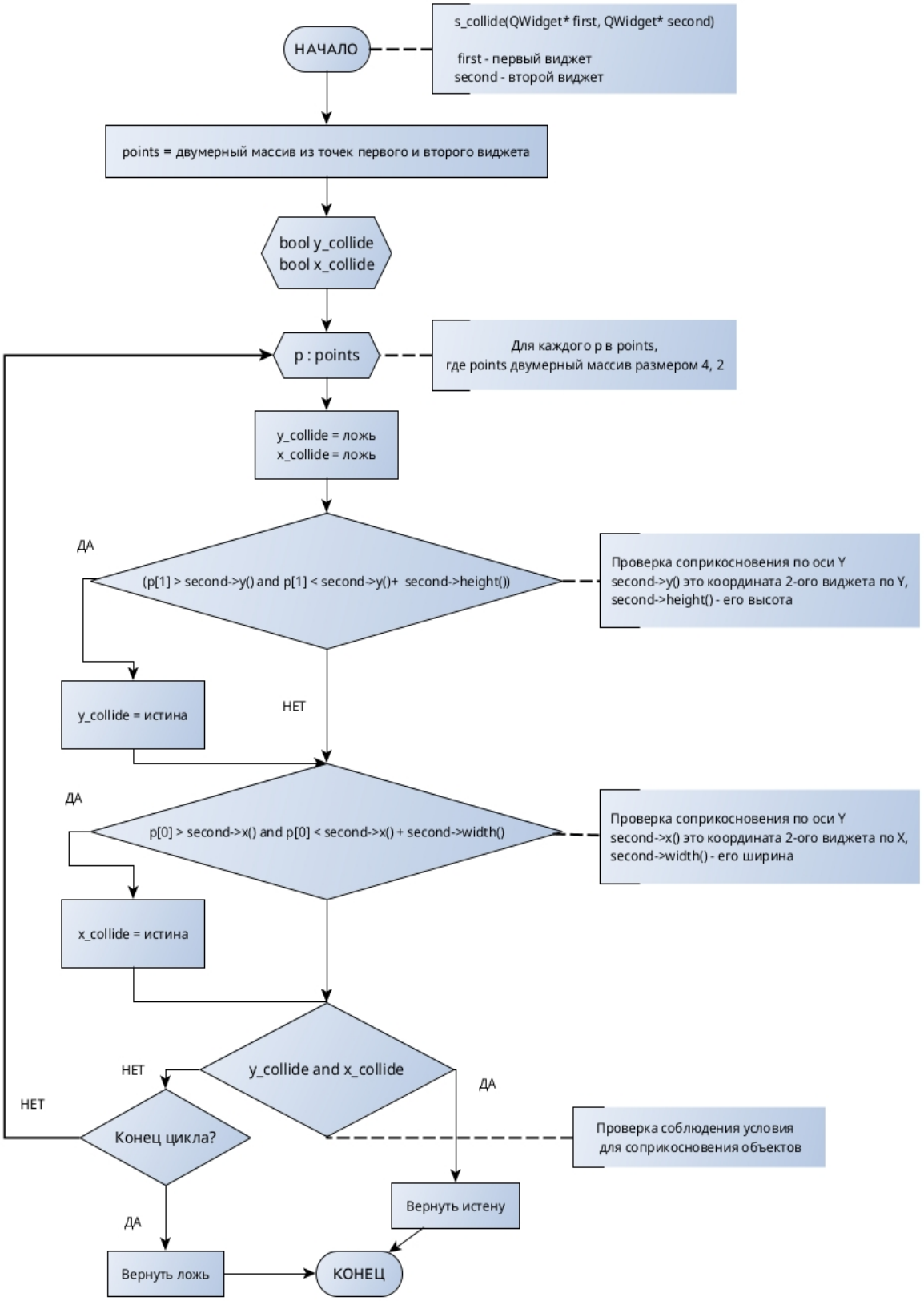

Рисунок 5 – Блок-схема алгоритма проверки столкновений двух виджетов

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).

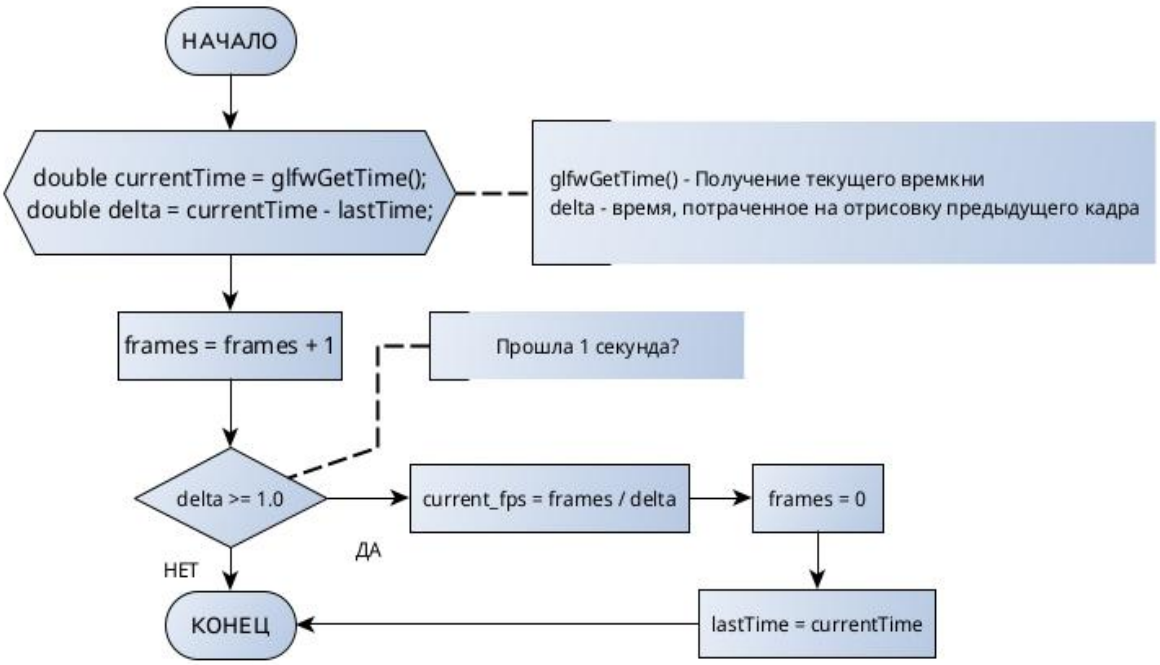

Рисунок 5 – Блок-схема алгоритма процедуры расчёта частоты кадров в секунду.

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 с разделителем. Функция split(std::string &str) разбивает строку str на массив чисел. get_main_levels_count() и get_bonus_levels_count() возвращают 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()

3.8 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.

3.9 Алгоритм обработки новой итерации игры

Реализуем данный алгоритм в виде блок-схемы (см. рис. 6).

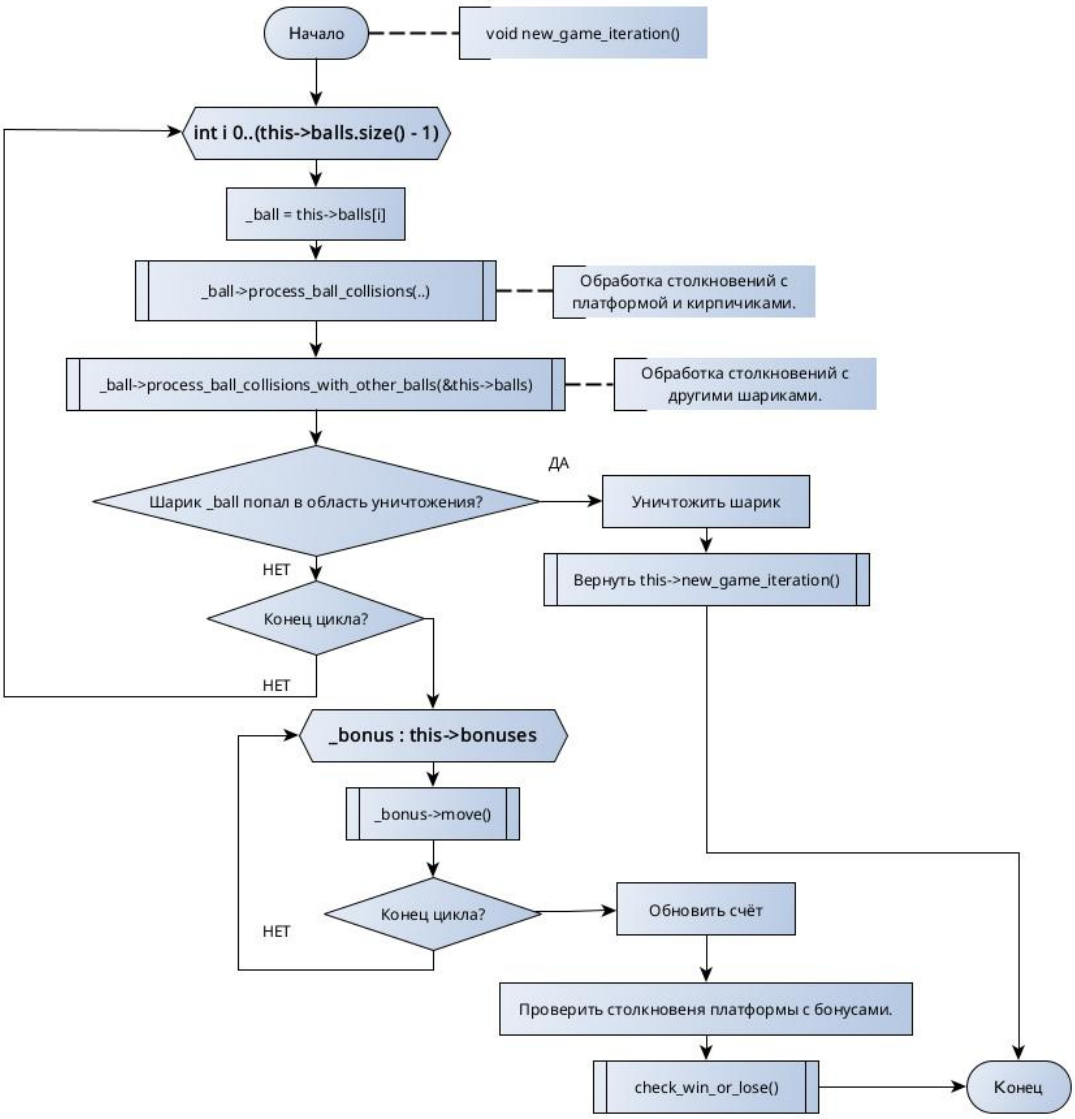

Рисунок 6 – Блок-схема алгоритма обработки новой игровой итерации

Рассмотрим блок-схемы для некоторых функций, использованных в вышеупомянутом алгоритме. Начнём с check_win_or_lose (см. рис. 7).

3.10 Организация файлов в скомпилированном проекте

Правильная организация файлов необходима для корректной работы программы. Исполняемые файлы должны хранится в папке bin, изображения в images, а уровни в levels. Рассмотрим иерархию файлов в виде графа (см. рис. 7).

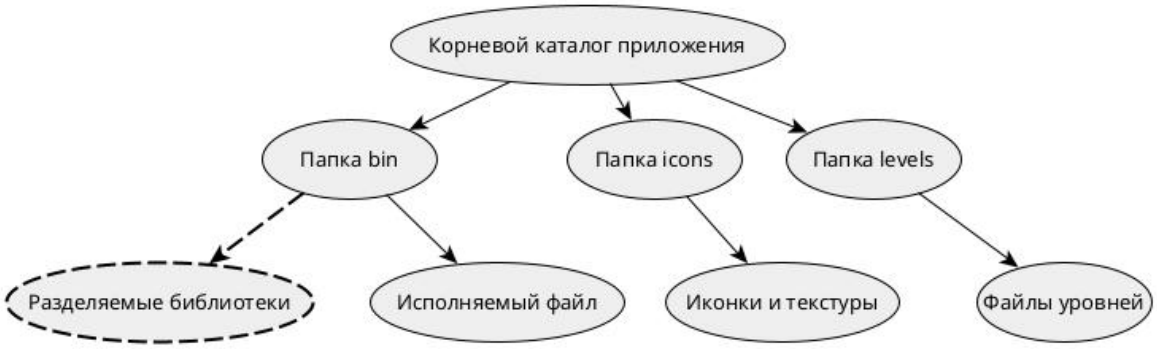

Рисунок 7 – Иерархия файлов в собранном пороете.

4 ПРИМИЧАНИЕ

У данной игры есть ряд требований к компьютеру пользователя о которых необходимо.

4.1 Разделяемые библиотеки и программы, необходимые для сборки.

Согласно требованиям все сторонние разделяемые библиотеки не должны распространятся с приложением. Поэтому ответственность за их корректную установку возлагается на пользователя.

Для пользователей Windows систем необходимо установить следующие пакеты для запуска скомпилированной программы:

- Qt6 (Скачать установщик с официального сайта qt.io)
- Mingw (Поставить галочку в онлайн установщике qt)
- GLFW (Скачать с официального сайта glfw.org)

Для сборки потребуется ещё установить:

- Пакеты для разработка под c++ для Microsoft Windows (Поставить галочку в онлайн установщике qt или скачать с официального сайта Microsoft)
- CMake (Поставить галочку в онлайн установщике qt)

Если Windows не позволяет установить необходимую библиотеку, то её нужно скомпилировать из исходного кода и полученные .dll файлы поместить в папку с исполняемыми файлами игры (bin).

Для запуска на POSIX системах (Linux, MacOS...) потребуется следующие библиотеки (пакеты не dev версий!):

- libglfw.so.3

- libQt6Widgets.so.6
- libQt6Gui.so.6
- libQt6Core.so.6

Для сборки понадобятся установить дополнительно :

- make
- g++
- CMake
- Dev версии вышеуказанных пакетов (glfw, Qt6)

Установить их можно пр помощи пакетного менеджера для конкретной системы. Также для всех пакетов понадобится установить зависимости!

4.2 Аппаратные требования

- Многоядерный процессор (Рекомендуется)
- Поддержка процессором инструкций: cx16, sse, sse2, sse3.
- Дискретная или встроенная видеокарта, способная обрабатывать OpenGL 3.3 и выше.
- amd64 совместимый процессор (Для скомпилированных исполняемых файлов, прикреплённых к проекту).

4.3 Требования к операционной системе

Из за санкций на территории РФ для ОС. Windows затруднительно получить, необходимые для запуска и сборки игры пакеты, поэтому её использование не рекомендуется! Для запуска потребуется Windows 7 или новее(Рекомендуется 10 64-биная).

Для POSIX систем главное требование, это возможность установить необходимые пакеты. Также необходимо наличие установленных последних обновлений.

4.4 Требования к Linux системам для запуска скомпилированной игры, приложенной к проекту.

К проекту приложены 2 сборки под разные дистрибутивы Linux(Для RHEL и Debian подобных систем). RHEL версия требует более новой версии glibc, поэтому её не рекомендуется запускать на дистрибутивах со старой пакетной базой. Она оптимизирована для новых RHEL подобных систем. Debian сборка предназначена для Debian подобных систем.

4.5 Особенности сборки проекта.

В POSIX системах сборка выполняется через bash скрипт build_linux.sh. В результате появляется папка Build, содержащая собранный проект.

В случае сборки под Windows или вручную через CMake потребуется вручную сформировать необходимую структуру игровых файлов, как сказано в подразделе 3.10 .

ВЫВОДЫ

Приложение реализовано и протестировано. Подготовлен отчёт о проделанной работе.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1.Информатика: Методические указания по выполнению практических работ / С. С. Смирнов, Д. А. Карпов – М., МИРЭА – Российский технологический университет, 2020. – 102 с.

2.Основы промышленного программирования / М. В. Преображенский Текст. Изображение. Устная речь : электронные : Лицей Академии Яндекса онлайн-учебник учебник. – URL: lyceum.yandex.ru/ (дата обращения:25.05.2022). – Режим доступа: только для студентов Лицея Академии Яндекса 2-ого курса.

3. Ravesli [Электронный ресурс] // Уроки по C++ : [сайт]. [2021]. URL: http://ravesli.com/uroki-cpp (дата обращения: 28.11.2022) – Режим доступа: для всех, кроме пользователей из России.