Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных Систем и Сетей

Кафедра информатики

Дисциплина: Программирование

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе на тему

**Создание игры “Arcanoid” в среде разработки QT C++**

|  |  |
| --- | --- |
| Студент гр.053503 | Н.П.Демьянцев |
| Руководитель: | И.А.Удовин |

Почтовый адрес: 220040

г. Минск, ул. Некрасова,

д.33, кв.53 тел.+375-33-603-96-91

e-mail:Nikita\_dem@list.ru

2021

**ЗАДАНИЕ**

по курсовой работе *студенту Демьянцеву Н.П.*

1. Тема курсовой работы «Создание игры “Arcanoid” в среде разработки QT C++».
2. Дата защиты курсовой работы «\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.
3. Исходные данные для курсовой работы
   1. Стандарт C++.
   2. Документация Qt
4. Содержание пояснительной записки 4.1. Введение.
   1. Интерфейс Qt Creator .
   2. Структура проекта
   3. Реализация меню
   4. Игровая область
      1. Метод работы с графическими компонентами
      2. Реализация проверки столкновений
      3. Отдельные процессы объектов
   5. Игровой процесс
   6. Заключение.
   7. Список используемых источников.
   8. Приложения
5. Консультант по курсовой работе Удовин И.А.
6. Дата выдачи задания «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Руководитель курсовой работы *И.А.Удовин*

Задание принял для исполнения *Н.П.Демьянцев*

**4.1 Введение**

Идея данной курсовой работы заключается в практическом применении возможностей среды разработки Qt для создания простого экземпляра известной игры “Arcanoid”, и отработке практических навыков, полученных во время выполнения лабораторных работ втечение семестра.

Для создания игры был выбран вариант использования Qt по нескольким причинам:

1. Простой и интуитивно понятный интерфейс
2. Продвинутая в сравнении с CBuilder система классов и библиотек, поставляемая самой системой разработки
3. Визуальное исполнение

Язык, используемый для создания проекта – С++, поскольку данный язык программирования и позволяет использовать все необходимые особенности парадигмы программирования Объектно Ориентированного Программирования, и достаточно гибок в обращении, но и предполагает умелое обращение с памятью, что требует от программиста навыков, позволяющих не создавать “висячих указателей”, или других ошибок.

Никаких дополнительных графичеких библиотек использовать необходимости не возникло, поскольку в Qt присутствуют все необходимые графические компоненты.

Автор данной курсовой работы считает также хорошей альтернативой встроенным библиотекам Qt подключаемую библиотеку SFML, засчёт удобных и высокоуровневых вызможностей работы с графическими компонентами.

**4.2 Интерфейс Qt Creator**

Среда разработки Qt предполагает возможность создания как консольных (использующих встроенную консоль операционной системы, напр. Windows command prompt) программ, так и тех, что используют визуальные компоненты. (см рис.1)

Работа непосредственно с кодом проекта выполняется в настраиваемой области просмотра и редактирования кода (см рис. 2), где существует возможность выбора цвета, шрифта и размера текста (см табл. №1).

Работа с окнами и виджетами внутри окон происходит в интегрированном окне предпросмотра генерируемых форм (см рис. 3), или же вручную – всё также в области редактирования кода (см рис. 3)

Табл №1. – Варианты настройки среды разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Цвет текста | Шрифт текста | Размер текста | Предустановленные стили |
| Qt::Red, Qt::Blue, Qt::Green, Qt::Yellow, Qt::Black, Qt::White, и др. | Все, поставляемые операционной системой, а также те, что доступны для загрузки из сети Internet | От 10 до 72 пт. | Qt Creator Dark,  Dark, Default, Default classic, grayscale, InkPot, IntelliJ Idea, Modnokai nightshift v2, Solarized Darkm, Solarized Light |
| Все возможные для указания с помощью таблицы RGB |

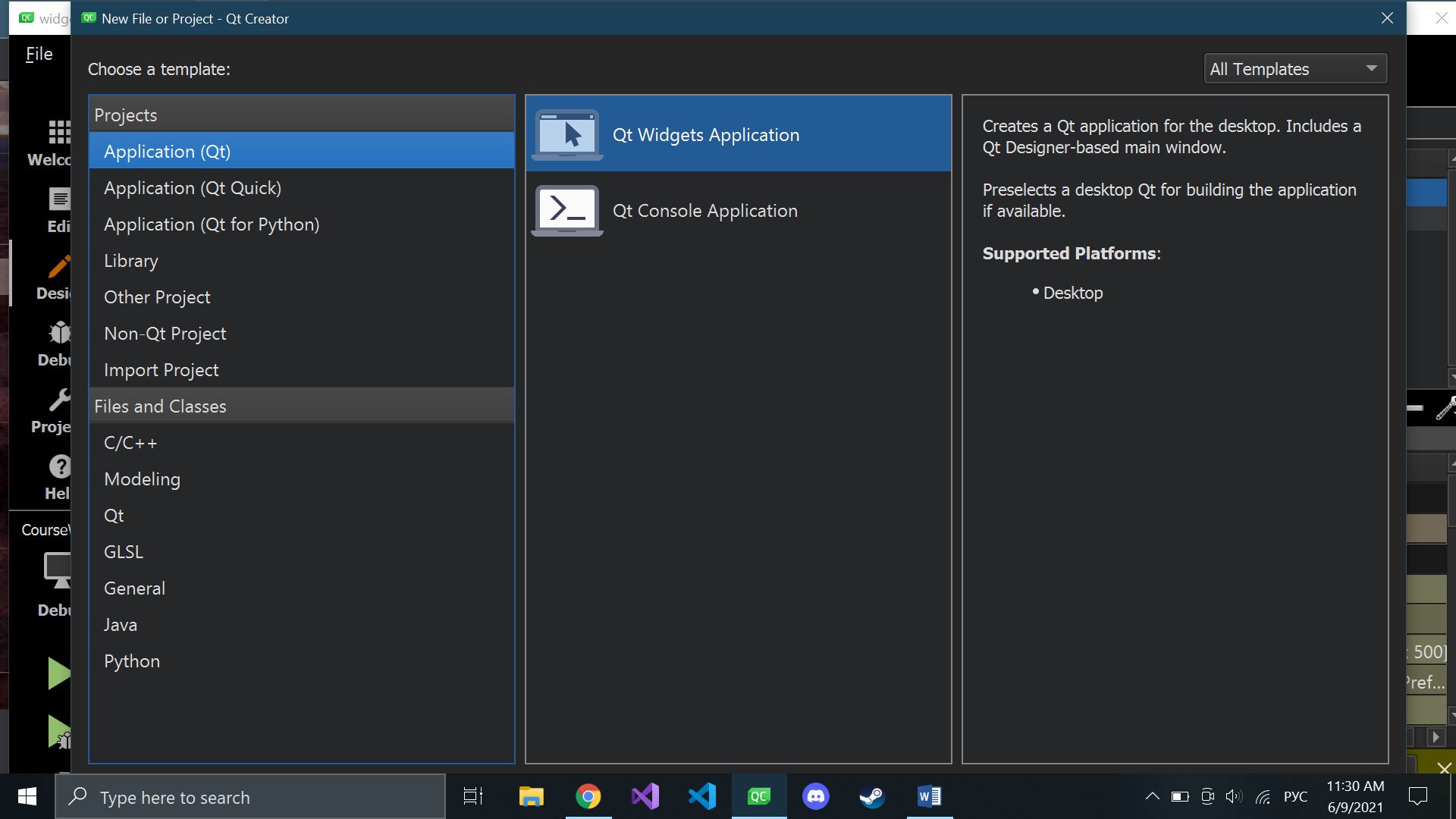


Рис. 1

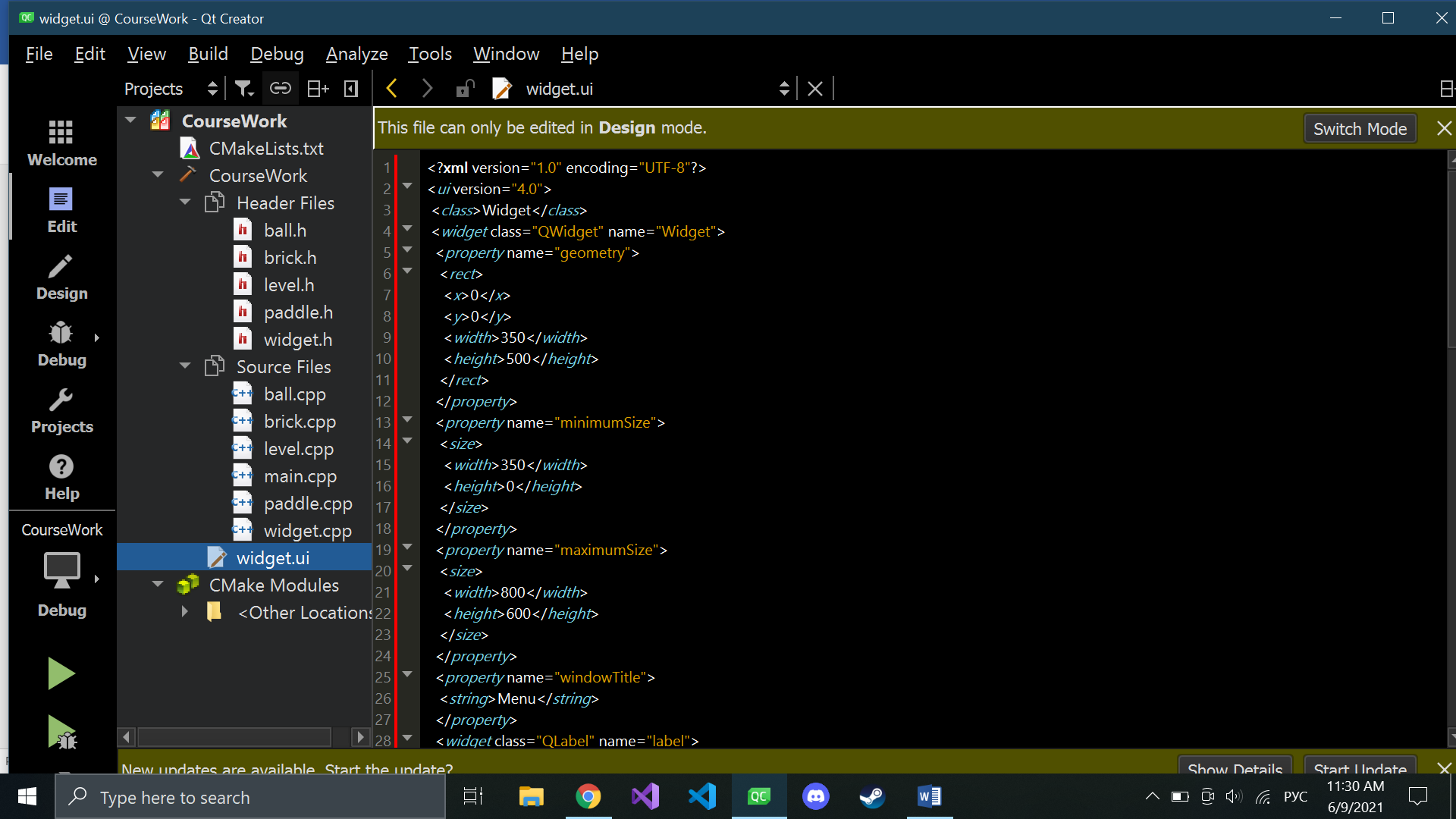


Рис. 2

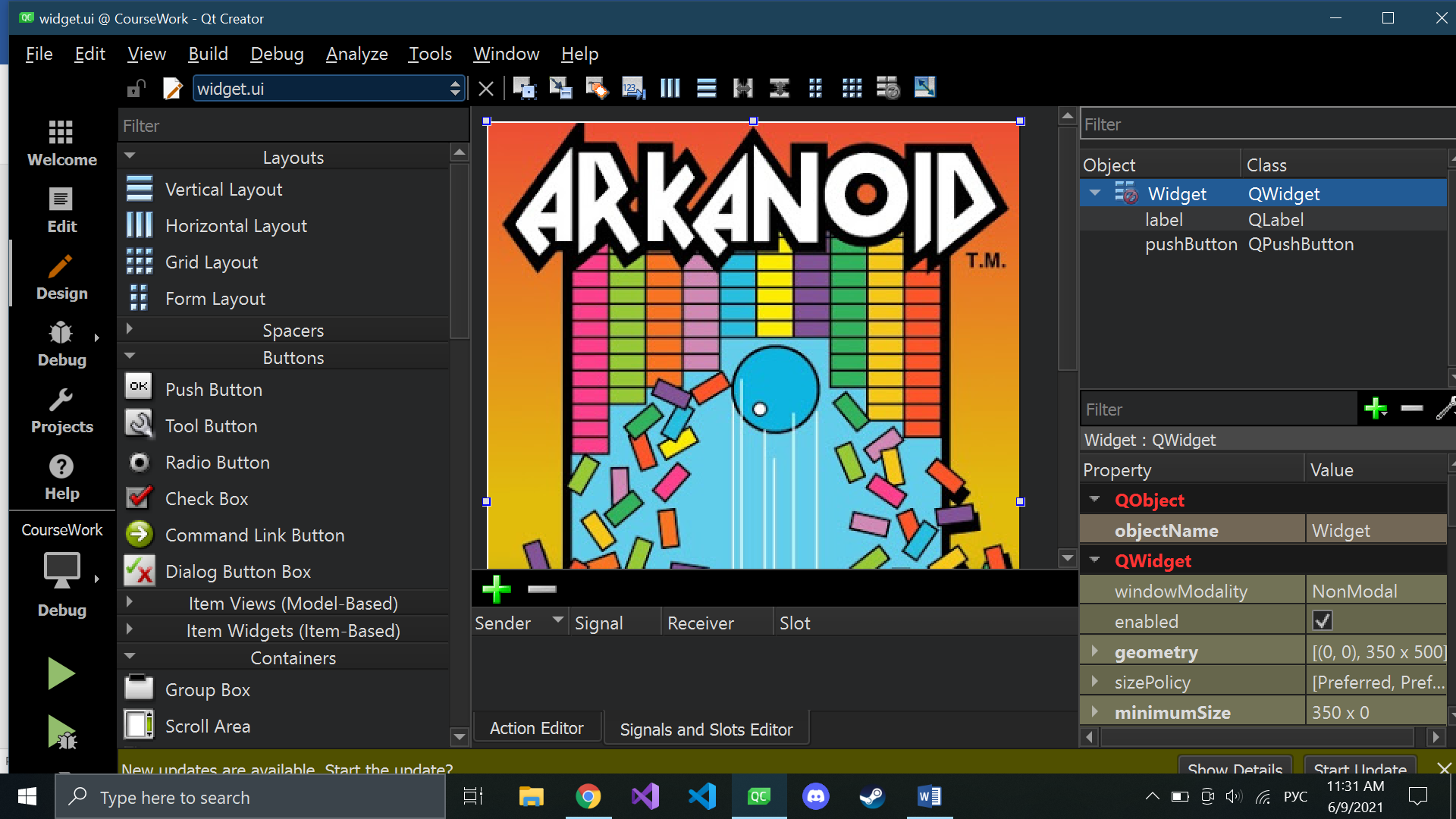


Рис. 3

**4.3 Структура проекта**

Весь проект “Arcanoid” можно поделить на 3 части – работа меню (события нажатия на кнопки), работа графической составляющей и работа логической составляющей классов и проекта в целом.

* Меню представлено встроенной в среду разработки формой – окном .ui. При нажатии на кнопку start каждый раз можно будет начать игру заново. Нажатие на крест закроет и остановит работу программы
* Графическая составляющая игры основана на работе встроенного класса Qt – QpaintEvent
* Логика классов описана согласно стандартному программированию на языке C++ при работе с классами.

Меню выглядит следующим образом (см рис.4)



Рис. 4

Основные моменты работы меню см Приложение 1

**4.5 Игровая область**

**4.5.1 Метод работы с графическими компонентами**

Как уже было названо выше, работа с графическими компонентами в Qt основана на работе встроенного класса QPaintEvent, а именно: автор данной курсовой работы режит использовать метод перерисовывания поля, отвечающего за игровую область на каждый игровой тик (10 мс) используя таймер – QTimer.

Фрагменты кода, отвечающие за работу QPaintEvent:

void Level::*paintEvent*(QPaintEvent \*e)

{

Q\_UNUSED(e);

QPainter painter(*this*);

*if* (gameOver) {

finishGame(*false*);

} *else* *if*(gameWon) {

finishGame(*true*);

}

*else* {

drawObjects(*&painter*);

}

}

И

void Level::drawObjects(QPainter \*painter)

{

painter->drawImage(ball->getRect(), ball->getImage());

painter->drawImage(paddle->getRect(), paddle->getImage());

*for* (int i=0; i<brickCount; i++) {

*if* (!bricks[i]->isDestroyed()) {

painter->drawImage(bricks[i]->getRect(), bricks[i]->getImage());

}

}

}

Где finishGame() отвечает за вызов окна о конце игры и остановки отрисовки, а все функции в drawObjects() отвечают за отрисовку каждого объекта отдельно

**4.5 Игровая Область**

**4.5.2 Реализация проверки столкновений**

Данный раздел описывают одну из самых сложных особенностей работы с графическими объектами – проверка столкновений между объектами (Шар с ракеткой и Шар с кирпичём)

Решить проблему столкновений помогают встроенные функции QPaintEvent – взятие границ прямоугольника, отвечающего за объект – bottom(), up(), right(), left().

Фрагменты кода, использующие данные механики:

void Level::checkCollision() {

*if* (ball->getRect().bottom() > botEdge) {

stopGame();

}

*for* (int i=0, j=0; i<brickCount; i++) {

*if* (bricks[i]->isDestroyed()) {

j++;

}

*if* (j == brickCount) {

victory();

}

}

*if* ((ball->getRect()).intersects(paddle->getRect())) {

int paddleLeftEdge = paddle->getRect().left();

int ballLeftEdge = ball->getRect().left();

int caseOne = paddleLeftEdge + 8;

int caseTwo = paddleLeftEdge + 16;

int caseThree = paddleLeftEdge + 24;

int caseFour = paddleLeftEdge + 32;

*if* (ballLeftEdge < caseOne) {

ball->setXDir(-1);

ball->setYDir(-1);

}

*if* (ballLeftEdge >= caseOne && ballLeftEdge < caseTwo) {

ball->setXDir(-1);

ball->setYDir(-1\*ball->getYDir());

}

*if* (ballLeftEdge >= caseTwo && ballLeftEdge < caseThree) {

ball->setXDir(0);

ball->setYDir(-1);

}

*if* (ballLeftEdge >= caseThree && ballLeftEdge < caseFour) {

ball->setXDir(1);

ball->setYDir(-1\*ball->getYDir());

}

*if* (ballLeftEdge > caseFour) {

ball->setXDir(1);

ball->setYDir(-1);

}

}

*for* (int i=0; i<brickCount; i++) {

*if* ((ball->getRect()).intersects(bricks[i]->getRect())) {

int ballLeft = ball->getRect().left();

int ballHeight = ball->getRect().height();

int ballWidth = ball->getRect().width();

int ballTop = ball->getRect().top();

QPoint pointRight(ballLeft + ballWidth + 1, ballTop);

QPoint pointLeft(ballLeft - 1, ballTop);

QPoint pointTop(ballLeft, ballTop -1);

QPoint pointBottom(ballLeft, ballTop + ballHeight + 1);

*if* (!bricks[i]->isDestroyed()) {

*if*(bricks[i]->getRect().contains(pointRight)) {

ball->setXDir(-1);

}

*else* *if*(bricks[i]->getRect().contains(pointLeft)) {

ball->setXDir(1);

}

*if*(bricks[i]->getRect().contains(pointTop)) {

ball->setYDir(1);

}

*else* *if*(bricks[i]->getRect().contains(pointBottom)) {

ball->setYDir(-1);

}

bricks[i]->setDestroyed(*true*);

}

}

}

}

Вся представленная функция отвечает за проверку столкновений и представляет основной логический процесс, происходящий каждый игровой тик, где также выполняется проверка на столкновение с кирпичём, изменение направления движения шара и уничтожение кирпича.

**4.5 Игровая область**

**4.5.3 Отдельные процессы объектов**

Данный отдел описывает работу некоторых моментов, реализованных в работе таких классов, как Ball, Paddle, Brick и игрового поля – Level.

* Работа класса Ball главным образом представлена функцией автоматического движения каждый игровой тик:

void Ball::autoMove() {

rect.translate(xdir, ydir);

*if* (rect.left() == 0) {

xdir = 1;

}

*if* (rect.right() == rightEdge) {

xdir = -1;

}

*if* (rect.top() == 0) {

ydir = 1;

}}

* Основная логика в работе Brick представлена конструктором объекта Brick, где происходит выбор случайной текстуры

Brick::Brick(int x, int y)

{

int r = QRandomGenerator::global()->generate() % 8;

image.load(brickImages[r]);

destroyed = *false*;

rect = image.rect();

rect.translate(x, y);

}

* Работа Paddle представлена также в конструкторе

void Paddle::move() {

int x = rect.x() + dx;

int y = rect.top();

rect.moveTo(x, y);}

А также реализовано взаимодествие с игроком, которое представлено внутри класса Level:

void Level::*keyReleaseEvent*(QKeyEvent \*e) {

int dx = 0;

*switch* (e->key()) {

*case* Qt::*Key\_Left*:

dx = 0;

paddle->setDx(dx);

*break*;

*case* Qt::*Key\_Right*:

dx = 0;

paddle->setDx(dx);

*break*;

}

}

void Level::*keyPressEvent*(QKeyEvent \*e) {

int dx = 0;

*switch* (e->key()) {

*case* Qt::*Key\_Left*:

dx = -2;

paddle->setDx(dx);

*break*;

*case* Qt::*Key\_Right*:

dx = 2;

paddle->setDx(dx);

*break*;

*case* Qt::*Key\_P*:

pauseGame();

*break*;

*case* Qt::*Key\_Space*:

startGame();

*break*;

*case* Qt::*Key\_Escape*:

qApp->exit();

*break*;

*default*:

QWidget::keyPressEvent(*e*);

}}

Данные функции отвечают за обработку нажатых клавишь во время игры, главным образом влияя на Paddle

* В функционировании класса Level осталось несколько неописанных функций, отвечающих за игровой процесс:

void Level::startGame() {

*if* (!gameStarted) {

ball->resetState();

paddle->resetState();

*for* (int i=0; i<brickCount; i++) {

bricks[i]->setDestroyed(*false*);

}

gameOver = *false*;

gameWon = *false*;

gameStarted = *true*;

timerId = startTimer(delay);

}

}

void Level::pauseGame() {

*if* (paused) {

timerId = startTimer(delay);

paused = *false*;

} *else* {

paused = *true*;

killTimer(timerId);

}

}

void Level::stopGame() {

killTimer(timerId);

gameOver = *true*;

gameStarted = *false*;

}

void Level::victory() {

killTimer(timerId);

gameWon = *true*;

gameStarted = *false*;}

Эти функции начинают игру, ставят на паузу, а также заканчивают с победой или поражением, в зависимости от результатов, которых достиг игрок по ходу игрового процесса.

* Также далее представлены реализации .h файлов(хедеров), где объявлены некоторые не указанные выше константы или функции:
* Шар, с его константами стартовой позиции и указания границы экрана справа:

*class* Ball {

*public*:

Ball();

~Ball();

*public*:

void resetState();

void autoMove();

void setXDir(int);

void setYDir(int);

int getXDir();

int getYDir();

QRect getRect();

QImage & getImage();

*private*:

int xdir;

int ydir;

QImage image;

QRect rect;

*static* *const* int startPosX = 230;

*static* *const* int startPosY= 355;

*static* *const* int rightEdge = 600;};

* Класс кирпича и переменная, отвечающая за проверку на существование и встроенные QImage и QRect

*class* Brick {

*public*:

Brick(int, int);

~Brick();

*public*:

bool isDestroyed();

void setDestroyed(bool);

QRect getRect();

void setRect(QRect);

QImage & getImage();

*private*:

QImage image;

QRect rect;

bool destroyed;};

* Класс ракетки, его стартовая позиция и функция, отвечающая за размещение в начале игры – resetState

*class* Paddle {

*public*:

Paddle();

~Paddle();

*public*:

void resetState();

void move();

void setDx(int);

QRect getRect();

QImage & getImage();

*private*:

QImage image;

QRect rect;

int dx;

*static* *const* int startPosX = 400;

*static* *const* int startPosY = 600;};

* Класс игрового поля Level и оставшиеся неописанные фрагменты кода

#include <QWidget>

#include <QKeyEvent>

#include <QMessageBox>

#include "ball.h"

#include "brick.h"

#include "paddle.h"

*class* Level : *public* QWidget {

*public*:

Level(QWidget \*parent = 0);

~*Level*();

*protected*:

void *paintEvent*(QPaintEvent \*);

void *timerEvent*(QTimerEvent \*);

void *keyPressEvent*(QKeyEvent \*);

void *keyReleaseEvent*(QKeyEvent \*);

void drawObjects(QPainter \*);

void finishGame(bool);

void moveObjects();

void startGame();

void pauseGame();

void stopGame();

void victory();

void checkCollision();

*private*:

int x;

int timerId;

*static* *const* int brickCount = 72;

*static* *const* int delay = 8;

*static* *const* int botEdge = 650;

Ball \*ball;

Paddle \*paddle;

Brick \*bricks[brickCount];

bool gameOver;

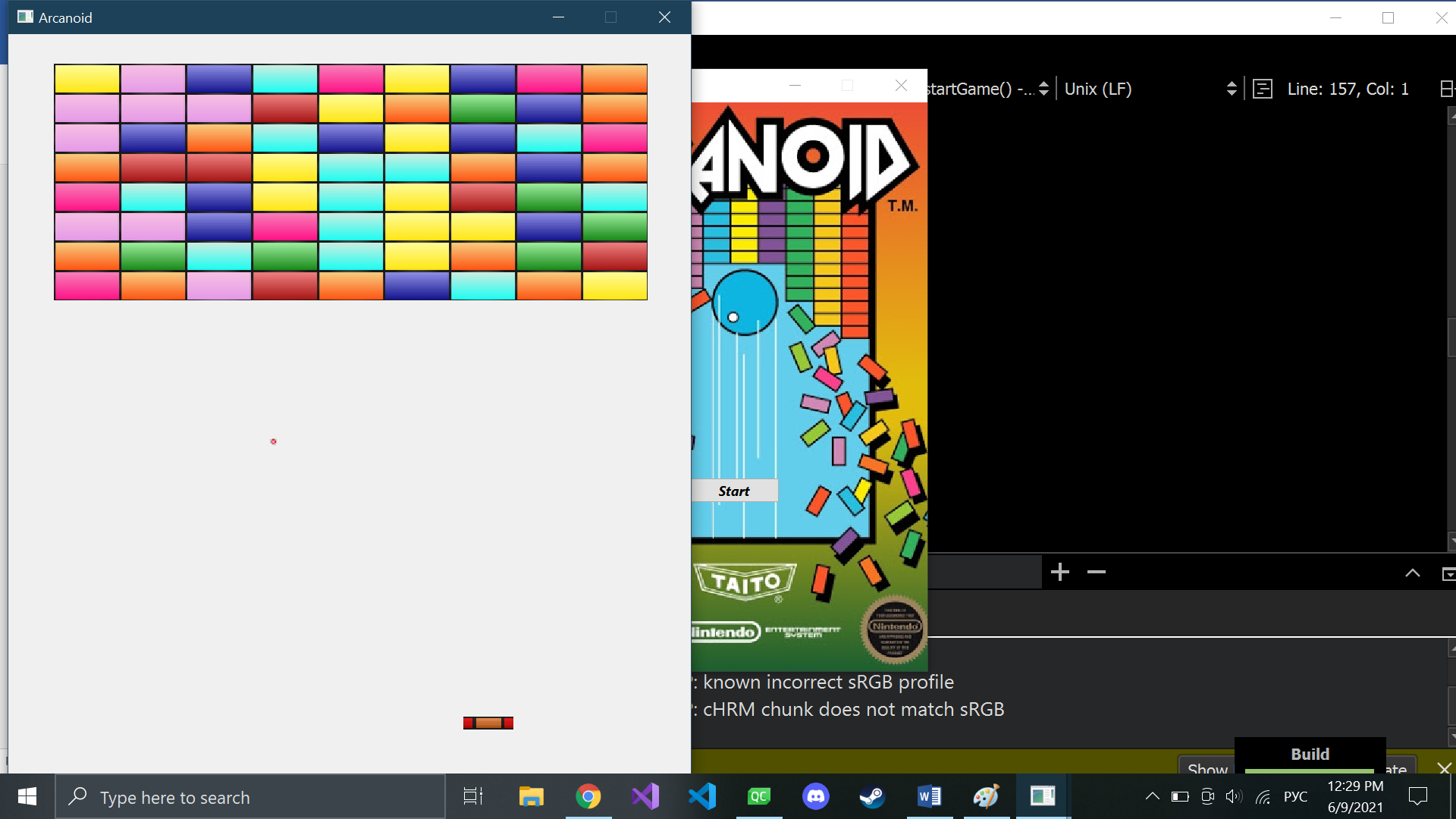
bool gameWon;

bool gameStarted;

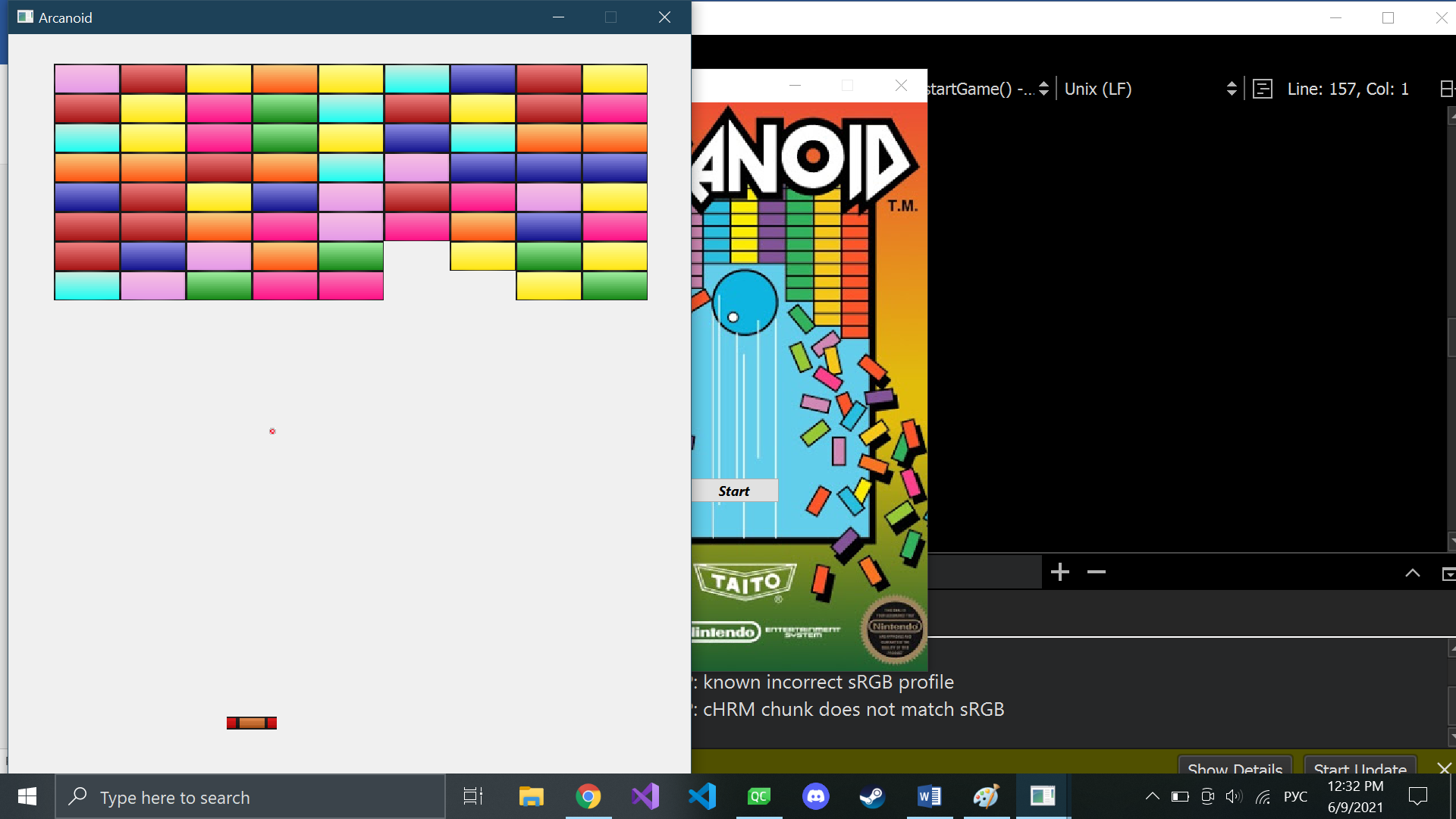
bool paused;

};

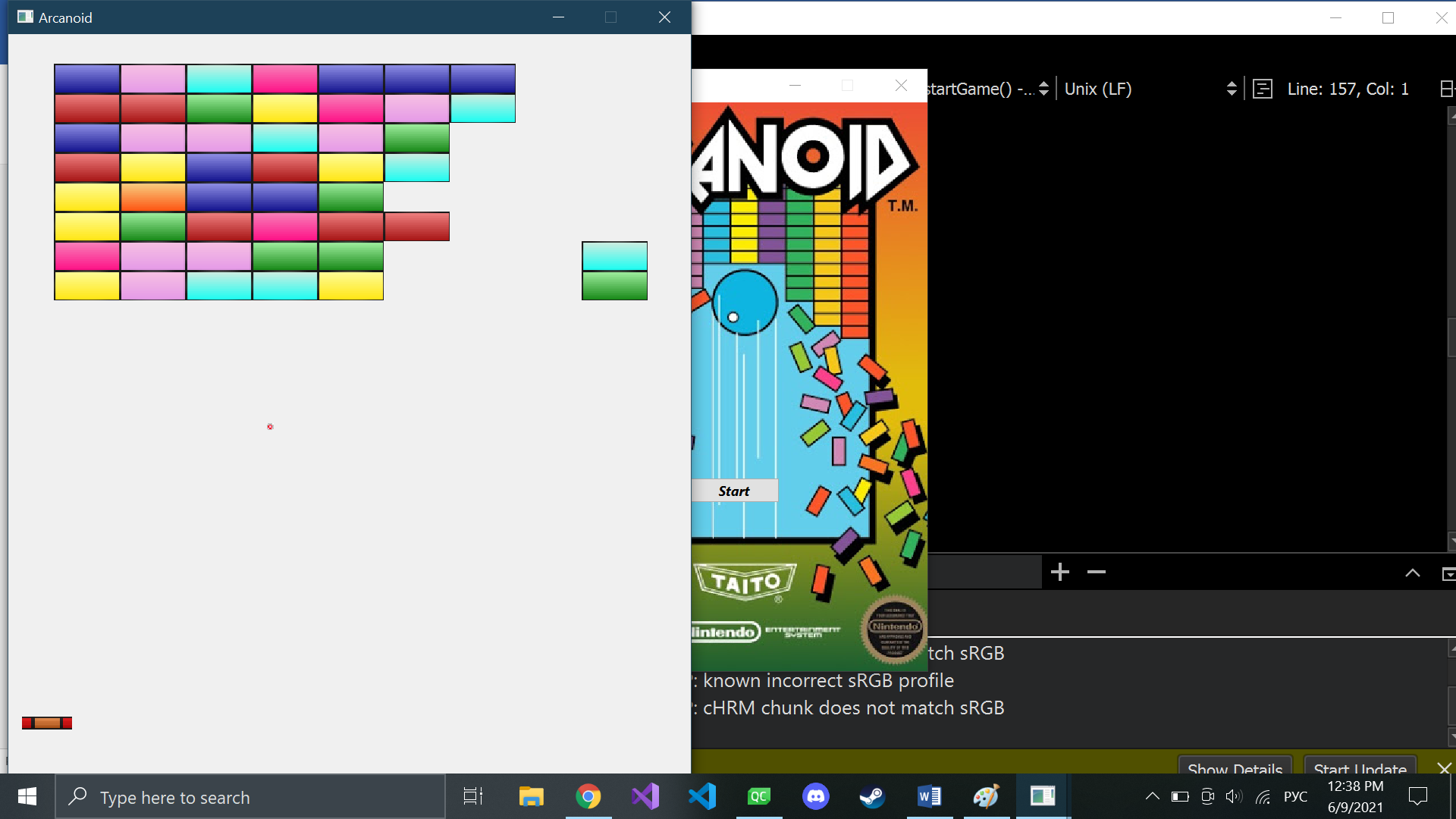
**4.6 Игровой процесс**

В данном разделе будет представлен и описан игровой процесс и результаты, которые увидит игрок после победы или поражения. 

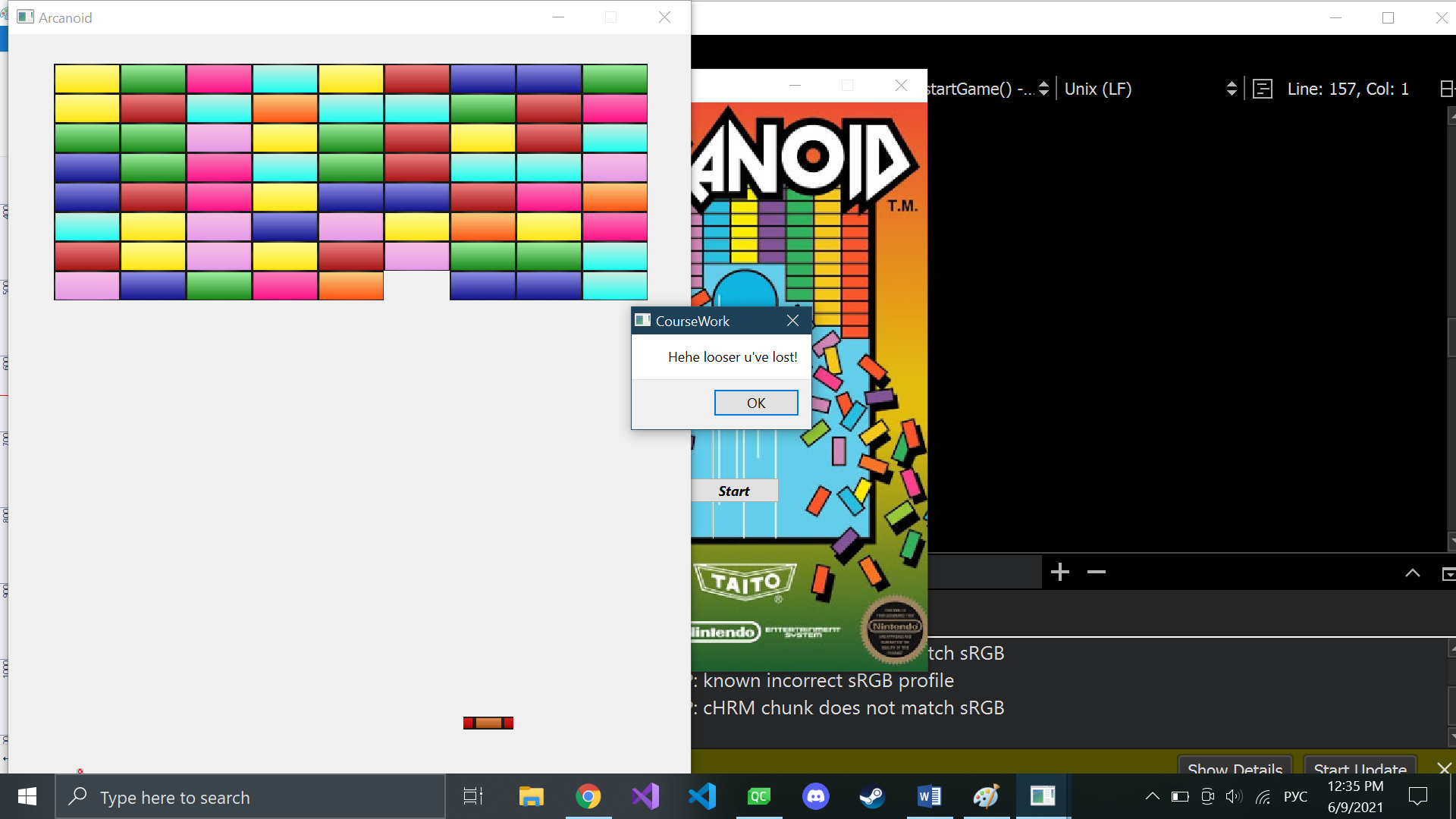
Вид игрового окна после нажатия Start, для начала игры нажать пробел



Игровая область в процессе игры, уничтожено 3 кирпича



Процесс игры, уничтожена практически половина кирпичей



Сообщение игры о том, что игрок проиграл

**4.7 Заключение**

В заключение автор данной курсовой работы хотел бы указать, что выполнение поставленного задание требовало необходимых указанных во введении навыков и позволило в достаточной мере практиковать как следование основным принципам ООП – Наследование, Абстракция, Инкапсуляция, Полиморфизм(меньше), так и использование встроенных возможностей среды разработки Qt, а также общая стандартизированная работа с классами и их методами.

Из выводов, которые можно было сделать во время работы, представляются:

* Среда разработки Qt отлично подходит для написания программы с использованием графических компонентов
* При работе с Qt у стундентов выйдет положительная практика по применению ООП

**4.8 Список источников**

1. **Документация Qt:** [**https://doc.qt.io/**](https://doc.qt.io/)
2. **Парадигмы ООП:** [**https://stefaniuk.website/all/paradigmy-oop**](https://stefaniuk.website/all/paradigmy-oop)

**4.9 Приложения**

**Приложение 1 – Работа виджета, отвечающего за меню**

**Widget.h**

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include "level.h"

QT\_BEGIN\_NAMESPACE

*namespace* Ui { *class* Widget; }

QT\_END\_NAMESPACE

*class* Widget : *public* QWidget

{

Q\_OBJECT

*public*:

Widget(QWidget \*parent = *nullptr*);

~*Widget*();

*private* slots:

void on\_pushButton\_clicked();

*private*:

Ui::Widget \*ui;

};

#endif *//* *WIDGET\_H*

*Widget.cpp:*

#include "widget.h"

#include "./ui\_widget.h"

Widget::Widget(QWidget \*parent)

: QWidget(*parent*)

, ui(*new* Ui::Widget)

{

ui->setupUi(*this*);

setMinimumSize( 350, 500 );

setMaximumSize( 350, 500 );

}

Widget::~*Widget*()

{

*delete* ui;

}

void Widget::on\_pushButton\_clicked()

{

Level \*window = *new* Level();

window->setFixedSize(600,800);

window->setWindowTitle("Arcanoid");

window->show();

}