Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра Информационная безопасность автоматизированных систем

Направление подготовки 10.03.01 Информационная безопасность

Расчётно-графическая работа

по дисциплине «Языки программирования»

Тема: «Игра «Tower defense»»

Выполнил: студент 1 курса

учебной группы с-ИБС11

очной формы обучения

Катаржин Михаил Антонович

Проверил: ассистент каф. ИБС

Романчук С. П.

Саратов 2020

**Аннотация**

Выполнение расчётно-графической работы можно свести к нескольким обобщённым областям, а именно: ознакомление с новой средой разработки QtCreator; ознакомление и изучение работы с GUI в QtCreator; изучение алгоритмов нахождения кратчайших путей между двумя точками в пространстве, способном содержать между ними непреодолимые преграды; разработка классов ( возможно наследуюмых и наследующих ) для различных типов нападающих, обороняющихся и препятсвующих движению объектов; разработка карт местности игрового поля.

Выполнение данной работы позволит и обязует обучиться работе с выводом окон для приложения, интерактивных виджетов и графических элементов, взаимодействию программ с периферийными устройствами отличными от клавиатуры, а именно мышь компьютерная, логике действия алгоритмов для конкретной задачи.

**Содержание**

[Аннотация](#Аннотация) ………………………………………………………………..………………2

[Содержание](#Содержание) ……………………………………………………………………..……….3

[Введение](#Введение) ……………………………………………………………..…………………..4

[Теоретическая часть](#ТеоретическаяЧасть) ……………………………………………………………..……...5

[Практическая часть](#ПрактическаяЧасть) ……………………………………………………………..………8

[Стандартные библиотеки, используемые в программе](#Стандартные_библиотеки) ……………...………8

[Разработанные классы, используемые в программе](#Разработанные_классы) ..…………………..……10

[Алгоритмы](#Алгоритмы) ……………………………………….……………………..………11

[Заключение](#Заключение) ……………………………………………………………..………………13

[Приложения](#Приложения) ……………………………………………………………..……………...14

[Листинг 1 – класс Menu](#Листинг_1_класс_Menu) ……………………………...………………………...14

[Листинг 2.1 – интерфейс IEnemyFactory](#Листинг_2_1_интерфейс_IEnemyFactory) ………………………..…………....15

[Листинг 2.2 – классы Easy / Base / HardEnemyFactory](#Листинг_2_2_класс_EnemyFactories) ………………………16

[Листинг 3.1 – интерфейс IEnemy](#Листинг_3_1_интерфейс_IEnemy) …………………………………….............18

[Листинг 3.2 – классы Enemy, EnemyArmored, EnemyRunner](#Листинг_3_2_класс_Enemy_Armored_Runner) ………...........18

[Листинг 4 – класс Game](#Листинг_4_класс_Game) ……………………………………….......................25

[Листинг 5 – класс HUD](#Листинг_5_класс_HUD) …………………………….………….......................32

[Листинг 6 – класс Level](#Листинг_6_класс_Level) ……………………………………...........................34

[Листинг 7.1 – интерфейс ILevelParser](#Листинг_7_1_интерфейс_IlevelParser) …………………………….................38

[Листинг 7.2 – класс TextFileLevelParser](#Листинг_7_2_класс_TextFileLevelParser) ……………………………..............39

[Листинг 8 – класс Map](#Листинг_8_класс_Map) …………………………….......................................42

[Листинг 9 – класс RoadFinder](#Листинг_9_класс_RoadFinder) …………………………….............................43

[Листинг 10 – класс TowerPlace](#Листинг_10_класс_TowerPlace) ……………………………...........................50

[Листинг 11 – класс CastlePlace](#Листинг_11_класс_CastlePlace) ……………………………...........................51

[Листинг 12 – класс RoadPlace](#Листинг_12_класс_RoadPlace) ……………………………............................51

[Листинг 13 – класс DenPlace](#Листинг_13_класс_DenPlace) ………………………………..........................53

[Листинг 14.1 – интерфейс ITower](#Листинг_14_1_интерфейс_ITower) ……………………………......................54

[Листинг 14.2 – классы Tower, TowerAntiArmor, TowerRapidFire](#Листинг_14_2_класс_Tower) …..…..…55

[Листинг 15 – класс TowerArea](#Листинг_15_класс_TowerArea) .………………….........................................62

[Листинг 16 – класс TowerUI](#Листинг_16_класс_TowerUI) ………………….............................................63

[Листинг 17 – класс BuildingUI](#Листинг_17_класс_BuildingUI) …………………..........................................66

[Листинг 18 – класс UiObject](#Листинг_18_класс_UiObject) ………………….............................................69

[Листинг 19 – класс Bullet](#Листинг_19_класс_Bullet) …………………..................................................70

[Литература](#Литература) ..……………………………. ……………………….……………………..73

**Введение**

**Тема расчётно-графической работы:**

Игра «Tower defense»»

**Цель расчётно-графической работы:**

Используя среду разработки Qt Creator создать рекреационное интерактивное приложения с стратегическим и эконмическим уклоном типа Tower defense на языке С++.

**Задачи рассчётно-графической работы:**

* Изучение основных возможностей QtCreator;
* Изучение GUI в QtCreator;
* Подбор алгоритма для рассчёта перемещения вражеских юнитов от точек появления;
* Создание нескольких карт для игрового пространства;
* Создание класса «вражеского юнита» с уникальными характеристиками;
* Создание дополнительных классов вражеских юнитов с уникальными характеристиками;
* Создание класса «башни» для уменьшения целостности вражеских юнитов;
* Создание дополнительных классов башен с уникальными характеристиками;
* Создание и назначение спрайтов для «вражеских юнитов» и «башни»;
* Добавление возможности улучшения «башни» за игровую валюту;
* *Создание класса «препятсвия» на «дороге» для задержания «вражеских юнитов» до уничтожения «препятствия»;*
* Разработка меню приложения с выбром карт в окне игры;
* *Добавление возможности улучшения башен от количества убитых вражеских юнитов*;

**Видение итога выполнения работы:**

В результате выполнения работы ожидается получить рекреационное приложение двумерное с видом сверху башенной защиты с возможностю выбрать несколько различных карт, где нападали бы различные по скорости, прочности, урону и сопротивлениям врагам. Защита состояла бы из подбора места размещения и типа башен и препятствий способных к усовершенствоанию. Поражение засчитывается при уничтожении «препятсвия» - врат - преграждающего крайнюю к концу пути клетку дороги. Возможность приостановки, перезапуска и завершения игры с возвращением в меню выбора карт или завершения программы.

[**Теоретическая часть**](#Содержание)

**Выбор языка программирования: С++**

C++ - компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

Язык С++ возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьёрн Страуструп придумал ряд усовершенствований к языку C под собственные нужды. Когда в конце 1970-х годов Страуструп начал работать в Bell Labs над задачами теории очередей, он обнаружил, что попытки применения существующих в то время языков моделирования оказываются неэффективными, а применение высокоэффективных машинных языков слишком сложно из-за их ограниченной выразительности. Так, язык Симула имеет такие возможности, которые были бы очень полезны для разработки объемного программного обеспечения, но работает слишком медленно, а язык BCPL достаточно быстр, но слишком близок к языкам низкого уровня и не подходит для разработки объемного программного обеспечения. Страуструп дополнил язык C возможностями работы с классами и объектами. В результате практические задачи моделирования оказались доступными для решения как с точки зрения времени разработки (благодаря использованию Симула-подобных классов), так и с точки зрения времени вычислений (благодаря быстродействию C).

C++ — один из самых мощных и востребованных языков программирования. Ежедневно на нём пишут сотни приложений, зачастую использующих GUI. В противовес этому часто ставится скорость написания кода, которая, например, у интерпретируемых языков на порядок выше. Однако очевиден выбор, что лучше для конечного пользователя: время разработки приложения или его скорость работы?

Скорость исполнения кода — пожалуй, главный аргумент в пользу того, почему C++ был, есть и будет востребован в ИТ. В первую же очередь, огромное сообщество — плодородная почва для появления качественной литературы. По C++ есть несколько фундаментальных книг, по которым училось несколько поколений, есть новые, учитывающие все свежие изменения и актуальное ПО, есть масса интернет-ресурсов для обучения.

Компиляторы C++ есть на каждой операционной системе, большинство программ легко переносится с платформы на платформу, со средой разработки и библиотеками у вас точно не возникнет проблем. C++ — это демонстрация идеи классического программирования, когда 90% мыслей связано с кодом и лишь 10% с используемой периферией. Достаточно просто вспомнить, где он используется: микроконтроллеры, IoT, роботы, десктопные и мобильные приложения, веб, игры, системы моделирования, прогнозирования, обработки статистики и в нейронных сетях. Везде. Просто не существует такой области программирования, где C++ был бы бесполезен.

C++ имеет колоссальное сообщество программистов, которые постоянно делятся библиотеками, шаблонами и кодами, приходят на помощь новичкам и опытным коллегам.

С++ полезен в качестве фундамента для обучения. Java, JavaScript, C #, как и огромное количество других популярных языков программирования, содержит в основе принципы C++. Именно поэтому учебные программы многих вузов содержат курс «плюсплюс», который идёт сразу за получением основ на C. К примеру, принцип работы Java, одного из мощнейших языков современности, достаточно сложно понять, если не начать обучение с основ, которые впервые появились именно в C++. Кроме того, популярно мнение, что если вы сможете изучить C++, то любой другой язык не вызовет у вас никаких затруднений. Это что-то вроде того, как легко пытаться пересесть с механической коробки передач на автоматическую и как сложно проделать путь в обратном направлении.

Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие.

**Выбор среды разработки: Qt Creator**

Qt Creator (ранее известная под кодовым названием Greenhouse) — кроссплатформенная свободная IDE для разработки на С, С++ и QML. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW

Основная задача Qt Creator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому среди возможностей, присущих любой среде разработки, есть и специфичные, такие как отладка приложений на QML и отображение в отладчике данных из контейнеров Qt, встроенный дизайнер интерфейсов: как на QML, так и на QtWidgets.

В Qt Сreator реализовано автодополнение, в том числе ключевых слов, введённых в стандарте C++11 (начиная с версии 2.5), подсветка кода (её определение аналогично таковому в Kate, что позволяет создавать свои виды подсветок или использовать уже готовые). Также, начиная с версии 2.4, есть возможность задания стиля выравнивания, отступов и постановки скобок.

Qt представляет собой целый набор инструментов для быстрого и удобного проектирования GUI. Qt привлекает программистов в основном тем, что она является быстрой, удобной, гибкой и кроссплатформенной. Во многом Qt обогнала даже SFML: библиотека доступна как на Windows, Linux и Mac, так и на мобильных платформах — Windows Mobile, Android и iOS. С помощью Qt были разработаны такие известные приложения, как: KDE, Opera, Google Earth и Skype. Впервые Qt был опубликован в мае 1995 года.

За счёт использования собственного фреймворка и мощного инструментария Qt позволяет быстро и удобно создавать собственные кроссплатформенные приложения. Кроме того, команды разработчиков получают возможность работать на разных платформах, используя при этом общие инструменты для разработки и отладки.

**[Практическая часть](#Содержание)**

**Стандартные библиотеки, используемые в программе:**

**<math.h>** - определяет набор функций для выполнения общих математических операций и преобразований.

**<QApplication>** - управляет главным потоком и основными настройками приложения с GUI. Содержит главный цикл обработки сообщений, где обрабатываются и пересылаются все сообщения посланные оконной системой и другими ресурсами. Также тут реализованы инициализация, завершение приложения и управление сессией. Также в данном классе реализованы возможности расширения системы и приложения.

**<QFile>** - это устройство ввода-вывода для чтения и записи текстовых и двоичных файлов.

**<QGraphicsScene>** - предоставляет поверхность для управления большим числом графических 2D элементов. Этот класс служит как контейнер для QGraphicsItems. Он используется вместе с QGraphicsView для отображения графических объектов, таких как линии, прямоугольники, текст или даже собственные элементы на двухмерной поверхности. QGraphicsScene водит в каркас графического представления. QGraphicsScene также предоставляет функциональность, которая позволит вам эффективно определять положение элементов и какие элементы видимы внутри произвольной области сцены.

**<QList>** -является одним из универсальных классов контейнеров Qt . Он хранит элементы в списке, который обеспечивает быстрый доступ, вставку и удаление на основе индекса.

**<QObject>** - это сердцевина модели объектов Qt. Главная особенность этой модели - это очень мощный механизм для связи объектов, навываемый сигналами и слотами. QObject поддерживают информацию о дереве дочерних объектов. Когда Вы создаете QObject в качестве родителя другого объекта, то объект автоматически добавит себя в список дочерних объектов children(). Родитель принимает в собственность свои дочерние объекты, т.е. дочерние объекты будут автоматически удалены в деструкторе объекта-родителя. Все виджеты Qt наследуют QObject. С помощью функции isWidgetType() можно узнать, дейтвительлно ли объект является виджетом.

**<QPainter>** - содержит высокооптимизированные функции позволяющие выполнять большинство рисунков требуемых программами с GUI. QPainter может рисовать все от простых линий до сложных форм подобных секторам круга и хордам. Он также может рисовать выровненный текст и растровые изображения. Обычно он рисует в "естественной" системе координат, но может совершать также планарные и объёмные преобразования.

**<QPoint>** - Точка задается координатами x и y, доступ к которым можно получить с помощью функций x () и y ().Объект QPoint также может быть использован в качестве вектора: сложение и вычитание определяются как для векторов (каждый компонент добавляется отдельно). Кроме того, класс QPoint предоставляет функцию manhattanLength (), которая дает недорогую аппроксимацию длины объекта QPoint, интерпретируемого как вектор. Наконец, объекты QPoint можно передавать в потоковом режиме, а также сравнивать.

**<QPolygon>** - это QVector< QPoint>.

**<QRegularExpression>** - обеспечивает сопоставление шаблонов с использованием регулярных выражений.

**<QString>** - поддерживает символьную строку Unicode, хранит строку 16-разрядного QChar s, где каждый QChar соответствует одному кодовому блоку UTF-16. QString использует неявный общий доступ (копирование при записи), чтобы уменьшить использование памяти и избежать ненужного копирования данных. Это также помогает снизить издержки, связанные с хранением 16-разрядных символов вместо 8-разрядных символов.

**<QtDebug>** - используется всякий раз, когда разработчику необходимо записать отладочную или трассировочную информацию на устройство, файл, строку или консоль.

**<QTextStream>** - предоставляет удобный интерфейс для чтения и записи текста.

**<QTimer>** - предоставляет высокоуровневый программный интерфейс для таймеров. В многопоточных приложениях QTimer можно использовать в любом потоке, имеющем цикл событий. Чтобы запустить цикл событий из потока без графического интерфейса пользователя, используйте QThread:: exec (). Qt использует сродство потоков таймера, чтобы определить, какой поток будет излучать сигнал timeout (). Из-за этого необходимо запустить и остановить таймер в его потоке; невозможно запустить таймер из другого потока.

**<QtMultimedia>** - мультимедийный модуль Qt предоставляет богатый набор функций, который позволяет вам легко воспользоваться мультимедийными возможностями платформы, такими как воспроизведение мультимедиа и использование камер и радиоустройств.

**<QtWidgets>** - предоставляет набор элементов пользовательского интерфейса для создания классических пользовательских интерфейсов в стиле рабочего стола.

**<QWidget>** - является атомом пользовательского интерфейса: он получает мышь, клавиатуру и другие события из оконной системы, а также рисует представление самого себя на экране. Каждый виджет имеет прямоугольную форму, и они отсортированы в Z-порядке. Виджет обрезается его родителем и виджетами перед ним. Виджет, который не встроен в родительский виджет, называется окном. Как правило, окна имеют рамку и строку заголовка.

**<QVector>** - является одним из универсальных классов контейнеров Qt . Он хранит свои элементы в соседних местах памяти и обеспечивает быстрый доступ на основе индекса.

**Разработанные классы, используемые в программе:**

[**Menu**](#Листинг_1_класс_Menu) – отвечает за существование меню, цель которого – запуск игры с выбранным уровнем сложности.

[**Game**](#Листинг_4_класс_Game)– основной класс, содержащий в себе все необходимые для существоаания классы и отвечает за их кооперативную работу, за расположение основного окна, аудио-сопровождение, графическое отображение основных элементов программы и контролирует важнейшие события в процессе работы программы.

[**Map**](#Листинг_8_класс_Map) – хранит в двумерном массиве образную схему игрового поля, его размеры и размеры ячеек поля.

[**Level**](#Листинг_6_класс_Level) – хранит в себе Map, отвечает за графическую отрисовку имеющихся и новых объектов игрового поля, их создание и деструкцию.

[**TextFileLevelParser**](#Листинг_7_2_класс_TextFileLevelParser) – отвечает за считывание, преобразование и передачу в Map из имеющегосся текстового файла информации о параметрах игрового поля, ячеек образную схему содержания начальных объектов и, при необходимости и наличия в текстовом файле, полигона путевых точек.

[**HUD**](#Листинг_5_класс_HUD) – отвечает за отображение блока с основной игровой информацией.

[**CastlePlace**](#Листинг_11_класс_CastlePlace) – отвечает за главную ячейку, к которой должны стремиться вторженцы и , соответсвенно, их прибытие до которой необходимо предотвратить.

[**DenPlace**](#Листинг_13_класс_DenPlace) – отвечает за ячейки, на которых будут появляться вторженцы. При старте волны на каждой из ячеек будут порождаться новые вторженцы.

[**RoadPlace**](#Листинг_12_класс_RoadPlace) – отвечает за ячейки, по которым будут способны перемещаться вторженцы. Все ячейки имеют собственный вес и вес урона( в разработке :: на его основе вторженцы будут перестраивать свой маршрут во время движения ).

[**TowerPlace**](#Листинг_10_класс_TowerPlace) – отвечает за ячейки, на месте которых дозволено возводить защитные башни.

[**RoadFinder**](#Листинг_9_класс_RoadFinder) – отвечает за инициализацию всех RoadPlace и просчёт по среднему между A\* и Волновым алгоритму всех путей от существующего CastlePlace по доступным RoadPlace.

[**IEnemyFactory**](#Листинг_2_1_интерфейс_IEnemyFactory) – интерфейс для фабрик, производящих юнитов ( паттерн – абстрактная фабрика ) и возвращающих указатели или списки указателей на эти новые юниты.

[**EasyEnemyFactory**](#Листинг_2_2_класс_EnemyFactories), [**BaseEnemyFactory**](#Листинг_2_2_класс_EnemyFactories), [**HardEnemyFactory**](#Листинг_2_2_класс_EnemyFactories) – фабрики, отвечающие за производство юнитов по различным логикам ( паттерн – фабричный метод ) и возврат указателей или списков указателей на эти новые юниты.

[**IEnemy**](#Листинг_3_1_интерфейс_IEnemy) – общий интерфейс для всех юнитов.

[**Enemy**](#Листинг_3_2_класс_Enemy_Armored_Runner), [**EnemyArmored**](#Листинг_3_2_класс_Enemy_Armored_Runner), [**EnemyRunner**](#Листинг_3_2_класс_Enemy_Armored_Runner) – отвечает за все характеристики юнитов, просчитывание получаемого урона, логистику движения.

[**ITower**](#Листинг_14_1_интерфейс_ITower) – общий интерфейс для всех защитных башен.

[**Tower**](#Листинг_14_2_класс_Tower), [**TowerAntiArmor**](#Листинг_14_2_класс_Tower), [**TowerRapidFire**](#Листинг_14_2_класс_Tower) - отвечает за все характеристики защитных башен, процесс атаки ( поиск в пределах радиуса поражения вторженцев и нанесенеи урона ), логистику совершенствования башен и отслеживание наведения курсора на башню.

[**TowerArea**](#Листинг_15_класс_TowerArea) – отвечает за отрисовку дальности поражения башни.

[**TowerUI**](#Листинг_16_класс_TowerUI) – отвечает за отрисовку параметров выбранной башни, а так же кнопки взаимодействия с башней.

[**BuildingUI**](#Листинг_17_класс_BuildingUI) – отвечает за отображение доступных для создания ( покупки ) защитных башен с их начальными параметрами, а так же за выбор доступных для создания башен.

[**UiObject**](#Листинг_18_класс_UiObject) – отвечает за объект для взаимодействия с целью создания новой башни с отображением её растрового изображения.

[**Bullet**](#Листинг_19_класс_Bullet) – отвечает за существование, отрисовку и движение снаряда башни.

**Алгоритмы, используемые в программе:**

*Расположение объектов согласно образной схеме игрового поля проходит от начала списка до его конца:*

1 – на всех местах соответсвующих единице размещаются TowerPlace ( без данных ячеек невозможно будет оборонять );

2 – на всех местах соответсвующих двойке размещаются RoadPlace ( без данных ячеек вторженцы будут неиметь возможности перемещаться );

3 – на всех местах соответсвующих тройке размещаются RoadPlace для возможности перемещения по этим ячейкам, а над ней размещается DenPlace, указатель на которую добавляются в список всех спаунеров, по координатам которых будут появляться вторженцы ( без данных ячеек вторженцы не будут появляться );

4 – на всех местах соответсвующих четвёрке будут размещены изображения синих кристалов ( на данный момент не имеют применения );

5 – на всех местах соответсвующих тройке размещаются RoadPlace для возможности перемещения по этим ячейкам, а над ней размещается CastlePlace, указатель на которую записывается в Level, если указатель ранее уже содержался, то он заменяется новым CastlePlace ( без данной ячейки вторженцы не будут иметь возможности поеждать ).

*Рассчёт весов всех RoadPlace:*

В справочник рассматриваемых RoadPlace добавляется та, что лежит под CastlePlace, указатель на который содержится в Level; при добавлении в справочник в качестве ключа всегда используется вес ячейки.

Пока справочникрассматриваемых RoadPlace не пуст выбирается ячейка по минимальному ключу:

Последовательно выполняется проверка наличия RoadPlace по координатам:

с y() на высоту изображения ячейки больше;

с y() на высоту изображения ячейки меньше;

с x() на ширину изображения ячейки больше;

с x() на ширину изображения ячейки меньше;

при наличии RoadPlace, если её вес базовый или её вес без базового веса и веса урона превышает вес рассматриваемой ячейки, то ей присваивается вес равный сумме веса рассматриваемой, собственного базового веса и веса урона.

*Рассчёт пути юнита:*

1. Проверка наличия RoadPlace под собой
2. Если нынешние координаты совпадают с координатами следующей выбранной ячейки, т опроисходит рассчёт выбора новой следующей ячейки:
   1. Проверка наличия ячеек RoadPlace во всех направлениях на растоянии размера изображения ячейки;
   2. Сравнение и выбор наименьшей по весу ячейки в качестве слудующей.
3. Перемещение по x() и y() в сторону совпадения координат юнита с координатами следующей ячейки.
4. Если по координатам бнита обнаружена CastlePlace то вызывается win().

**[Заключение](#Содержание)**

В ходе выполнения расчётно-графической работы были достигнуты большинство посталвенных задач. Результаты полцчены не только в виде готовой для использования и распространения программы, но в сфере расширения знаний и навыков и сфере понимания и ощущения на практике многих сложностей и «подводных камней» в процессе разработки программ.

Была изучена часть функционала QT Creator, использование графических интерфейсов и взаимодействие граффических объектов и виджетов друг с другом, я изучил и развил навыки работы с многопоточностью, как и столкнулся со сложностями, завязанными на многопоточности. Из-за раскрытия в процессе выполнения работы новых возможностей или сложностей в осуществлении планированного многократно пересматривались алгоритмы действий разных объектов, логистики вычислений и логистики прявязок.

В ходе работы задача создания нескольких карт для игрового процесса я трансформировал в возможность здать любому пользользователю начальные характеристики игровой карты редактированием простого текстового файла map, что , по моему мнению, более удобно и раскрывает много путей развития направления связанного с картами (за счёт этого функционала создание дополнительных карт очень просто).

Алгоритм просчёта оптимальных путей в данной версии игры работает и просчитывает только при запуске карты, и неизменимо при дальнейших изменениях карты (постройка, улучшения и снос башен) в связи сложностей с многопоточностью и обращением разом многих объектов к одному изменяемому параметру. Данную проблему в дальнейшем возможно разрешить пересмотрев логистику хранения карты с весами и выдачу путей движения. По той же причине не были добавлены в возможности игры установка препятсвий на дороге.

Движения врагов привязаны координатно строго к ячейкам RoadPlace так как вычисление попиксельного свободного перемещения требует большего числа рассчётов, и в связи с намерением в дальнейшем изменить логистику для выбора путей было решено упростить перемещение для более лёгкой их замены при рефакторинге.

Так же при выполнении задания я ознакомился с многими паттернами и различными способами решений одинаковых задач, что раскрывает мои потенциальные возможности и увеличивает планы по улучшению разработанной к данному времени программы. Так же, помимо улучшения программы, я обнаружил и методы её оптимизации и универсализации создания и логистики многих объектов, участвующих в программе.

**[Приложения](#Содержание)**

*[Листинг 1 – класс Menu](#Содержание)*

#include <QObject>

#include <QtWidgets>

#include "enemy/IEnemyFactory.h"

*class* **Menu** : *public* QWidget

{

Q\_OBJECT

*public*:

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

*public* slots:

void **chooseEasy**();

void **chooseBase**();

void **chooseHard**();

*private*:

void **chooseLvl**(IEnemyFactory \*enemyFactory);

QPushButton easyGame;

QPushButton baseGame;

QPushButton hardGame;

QPushButton quit;

};

#include "enemy/BaseEnemyFactory.h"

#include "enemy/EasyEnemyFactory.h"

#include "enemy/HardEnemyFactory.h"

#include "game.h"

#include "level/TextFileLevelParser.h"

#include "level/XmlLevelParser.h"

#include "menu.h"

Menu::**Menu**(QWidget \*parent)

{

QVBoxLayout \* v\_box = *new* QVBoxLayout ();

*this*->setLayout(v\_box);

setWindowTitle("Tower Defense");

setFixedSize(300,300);

easyGame.setText("&Easy game");

baseGame.setText("&Base game");

hardGame.setText("&Hard game");

quit.setText("&Exit");

v\_box->addWidget(&easyGame);

v\_box->addWidget(&baseGame);

v\_box->addWidget(&hardGame);

v\_box->addWidget(&quit);

connect(&easyGame, SIGNAL(clicked()), *this*, SLOT(chooseEasy()));

connect(&baseGame, SIGNAL(clicked()), *this*, SLOT(chooseBase()));

connect(&hardGame, SIGNAL(clicked()), *this*, SLOT(chooseHard()));

connect(&quit, SIGNAL(clicked()), *this*, SLOT(close()));

Q\_UNUSED(parent);

}

void Menu::**chooseLvl**(IEnemyFactory \*enemyFactory)

{

Game \* game = *new* Game(*nullptr*, enemyFactory, *new* TextFileLevelParser(":/res/maps/map"));

*this*->close();

game->show();

}

void Menu::**chooseEasy**()

{

*this*->chooseLvl(*new* EasyEnemyFactory());

}

void Menu::**chooseBase**()

{

*this*->chooseLvl(*new* BaseEnemyFactory());

}

void Menu::**chooseHard**()

{

*this*->chooseLvl(*new* HardEnemyFactory());

}

*[Листинг 2.1 – интерфейс IEnemyFactory](#Содержание)*

#include <QGraphicsScene>

#include <QList>

#include "enemy/IEnemy.h"

#include "places/denPlace.h"

*class* **IEnemyFactory** {

*public*:

*virtual* IEnemy\* ***createEnemy***(DenPlace \* den, int wave, QGraphicsScene \* level) = 0;

*virtual* QList<IEnemy\*>\* ***createEnemies***(QList<DenPlace \*> \*listDens, int wave, QGraphicsScene \* level) = 0;

};

*[Листинг 2.2 – классы EasyEnemyFactory, BaseEnemyFactory, HardEnemyFactory](#Содержание)*

*class* **BaseEnemyFactory** : *public* IEnemyFactory

{

*public*:

**BaseEnemyFactory**();

IEnemy\* ***createEnemy***(DenPlace \* den, int wave, QGraphicsScene \* level) *override*;

QList<IEnemy\*>\* ***createEnemies***(QList<DenPlace \*> \*listDens, int wave, QGraphicsScene \* level) *override*;

*private*:

int counter = 0;

};

#include "BaseEnemyFactory.h"

#include "enemy/enemy.h"

#include "enemy/enemyArmored.h"

#include "enemy/enemyRunner.h"

IEnemy\* BaseEnemyFactory::***createEnemy***(DenPlace \* den,

int wave,

QGraphicsScene \* level

)

{

QPointF startPos = den->pos();

srand(time(0));

*switch* (rand() % 3)

{

*case* 1:

*return* *new* EnemyRunner(&startPos, wave, level);

*case* 2:

*return* *new* EnemyArmored(&startPos, wave, level);

*default*:

*return* *new* Enemy(&startPos, wave, level);

}

}

IEnemy\* EasyEnemyFactory::***createEnemy***(DenPlace \* den,

int wave,

QGraphicsScene \* level

)

{

QPointF startPos = den->pos();

srand(time(0));

*switch* (rand() % 15)

{

*case* 1:

*case* 3:

*return* *new* EnemyRunner(&startPos, wave, level);

*case* 2:

*case* 4:

*return* *new* EnemyArmored(&startPos, wave, level);

*default*:

*return* *new* Enemy(&startPos, wave, level);

}

}

IEnemy\* HardEnemyFactory::***createEnemy***(DenPlace \* den,

int wave,

QGraphicsScene \* level

)

{

QPointF startPos = den->pos();

srand(time(0));

*switch* (rand() % 10)

{

*case* 1:

*case* 3:

*case* 2:

*case* 4:

*return* *new* EnemyRunner(&startPos, wave, level);

*default*:

*return* *new* EnemyArmored(&startPos, wave, level);

}

}

QList<IEnemy\*>\* HardEnemyFactory::***createEnemies***(QList<DenPlace \*> \*listDens,

int wave,

QGraphicsScene \* level

)

{

QList<IEnemy\*> \* listEmenies = *new* QList<IEnemy\*>();

*for*(DenPlace \* currentDen : \*listDens)

{

listEmenies->push\_back(*createEnemy*(currentDen, wave, level));

}

*return* listEmenies;

}

*[Листинг 3.1 – интерфейс IEnemy](#Содержание)*

#include <QGraphicsObject>

*class* **IEnemy** : *public* QGraphicsObject {

Q\_OBJECT

*public*:

*virtual* int ***getpassedWay***() = 0;

*virtual* int ***getSpeed***() = 0;

*virtual* int ***getArmor***() = 0;

*virtual* void ***damaged***(int) = 0;

*public* slots:

*virtual* void ***move***() = 0;

*virtual* void ***stop***() = 0;

signals:

*virtual* void ***dead***(int) = 0;

*virtual* void ***win***() = 0;

};

*[Листинг 3.2 – классы Enemy, EnemyArmored, EnemyRunner](#Содержание)*

#include <math.h>

#include <QGraphicsScene>

#include <QPainter>

#include <QTimer>

#include "enemy/IEnemy.h"

#include "places/castlePlace.h"

#include "places/roadPlace.h"

*class* **Enemy**: *public* IEnemy

{

Q\_OBJECT

*public*:

**Enemy**(QPointF \*startCoordinats, int wave, QGraphicsScene \* level);

~***Enemy***();

int ***getpassedWay***();

int ***getSpeed***();

int ***getArmor***();

void ***damaged***(int);

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

QPainterPath ***shape***() *const*;

*public* slots:

void ***move***();

void ***stop***();

signals:

void ***dead***(int);

void ***win***();

*private*:

QPixmap \* sprite;

QTimer \* steps;

int passedWay;

int prize;

int max\_hp;

int current\_hp;

int speed;

int armor;

QGraphicsScene\* level;

RoadPlace \*nextRoadPlace;

};

Enemy::**Enemy**(QPointF \*startCoordinats, int wave, QGraphicsScene \* level)

{

*this*->level = level;

passedWay = 0;

sprite = *new* QPixmap(":/res/images/Enemy.png");

max\_hp = 100 \* pow(1.15, wave);

current\_hp = max\_hp;

speed = 500 \* pow(1.005, wave);

armor = 4 \* pow(1.15, wave);

prize = 5 \* pow(1.15, wave);

*this*->setPos(startCoordinats->x(),

startCoordinats->y());

QPointF tmpPoint;

QList<QGraphicsItem \*> listItems;

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y());

listItems = level->items(tmpPoint);

*for*(QGraphicsItem \* item : listItems)

{

nextRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(nextRoadPlace != *nullptr*)

*break*;

}

steps = *new* QTimer();

connect(steps, SIGNAL(timeout()), *this*, SLOT(move()));

steps->start(10000 / speed);

}

EnemyArmored::**EnemyArmored**(QPointF \*startCoordinats, int wave, QGraphicsScene \* level)

{

*this*->level = level;

passedWay = 0;

sprite = *new* QPixmap(":/res/images/EnemyArmored.png");

max\_hp = 175 \* pow(1.25, wave);

current\_hp = max\_hp;

speed = 400 \* pow(1.005, wave);

armor = 10 \* pow(1.2, wave);

prize = 8 \* pow(1.15, wave);

*this*->setPos(startCoordinats->x(),

startCoordinats->y());

QPointF tmpPoint;

QList<QGraphicsItem \*> listItems;

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y());

listItems = level->items(tmpPoint);

*for*(QGraphicsItem \* item : listItems)

{

nextRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(nextRoadPlace != *nullptr*)

*break*;

}

steps = *new* QTimer();

connect(steps, SIGNAL(timeout()), *this*, SLOT(move()));

steps->start(10000 / speed);

}

EnemyRunner::**EnemyRunner**(QPointF \*startCoordinats, int wave, QGraphicsScene \* level)

{

*this*->level = level;

passedWay = 0;

sprite = *new* QPixmap(":/res/images/EnemyRunner.png");

max\_hp = 100 \* pow(1.15, wave);

current\_hp = max\_hp;

speed = 550 \* pow(1.05, wave);

armor = wave;

prize = 7 \* pow(1.15, wave);

*this*->setPos(startCoordinats->x(),

startCoordinats->y());

QPointF tmpPoint;

QList<QGraphicsItem \*> listItems;

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y());

listItems = level->items(tmpPoint);

*for*(QGraphicsItem \* item : listItems)

{

nextRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(nextRoadPlace != *nullptr*)

*break*;

}

steps = *new* QTimer();

connect(steps, SIGNAL(timeout()), *this*, SLOT(move()));

steps->start(10000/ speed);

}

void EnemyRunner::***move***()

{

QPointF tmpPoint;

QList<QGraphicsItem \*> listItems;

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y());

listItems = level->items(tmpPoint);

RoadPlace \*currentRoadPlace = *nullptr*;

*for*(QGraphicsItem \* item : listItems)

{

currentRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(currentRoadPlace != *nullptr*)

*break*;

}

*if*( currentRoadPlace == *nullptr* )

*return*;

*if*( *this*->x() == nextRoadPlace->x() && *this*->y() == nextRoadPlace->y())

{

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y() - currentRoadPlace->getSprite()->height());

listItems = level->items(tmpPoint);

RoadPlace \*upRoadPlace = *nullptr*;

*for*(QGraphicsItem \* item : listItems)

{

upRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(upRoadPlace != *nullptr*)

*break*;

}

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y() + currentRoadPlace->getSprite()->height());

listItems = level->items(tmpPoint);

RoadPlace \*downRoadPlace = *nullptr*;

*for*(QGraphicsItem \* item : listItems)

{

downRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(downRoadPlace != *nullptr*)

*break*;

}

tmpPoint.setX(*this*->x() + currentRoadPlace->getSprite()->width());

tmpPoint.setY(*this*->y());

listItems = level->items(tmpPoint);

RoadPlace \*rightRoadPlace = *nullptr*;

*for*(QGraphicsItem \* item : listItems)

{

rightRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(rightRoadPlace != *nullptr*)

*break*;

}

tmpPoint.setX(*this*->x() - currentRoadPlace->getSprite()->width());

tmpPoint.setY(*this*->y());

listItems = level->items(tmpPoint);

RoadPlace \*leftRoadPlace = *nullptr*;

*for*(QGraphicsItem \* item : listItems)

{

leftRoadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(leftRoadPlace != *nullptr*)

*break*;

}

RoadPlace \*minRoadPlace = currentRoadPlace;

*if*(upRoadPlace != *nullptr*)

*if*(upRoadPlace->getWeight() < minRoadPlace->getWeight())

minRoadPlace = upRoadPlace;

*if*(downRoadPlace != *nullptr*)

*if*(downRoadPlace->getWeight() < minRoadPlace->getWeight())

minRoadPlace = downRoadPlace;

*if*(rightRoadPlace != *nullptr*)

*if*(rightRoadPlace->getWeight() < minRoadPlace->getWeight())

minRoadPlace = rightRoadPlace;

*if*(leftRoadPlace != *nullptr*)

*if*(leftRoadPlace->getWeight() < minRoadPlace->getWeight())

minRoadPlace = leftRoadPlace;

nextRoadPlace = minRoadPlace;

}

int dy = (*this*->y() < nextRoadPlace->y() ? 1 : (*this*->y() > nextRoadPlace->y() ? -1 : 0));

int dx = (*this*->x() < nextRoadPlace->x() ? 1 : (*this*->x() > nextRoadPlace->x() ? -1 : 0));

moveBy(dx,dy);

passedWay += sqrt(pow(dx,2) + pow(dy, 2));

tmpPoint.setX(*this*->x());

tmpPoint.setY(*this*->y());

listItems = *this*->collidingItems();

CastlePlace \*castlePlace =*nullptr*;

*for*(QGraphicsItem \* item : listItems)

{

castlePlace = *dynamic\_cast*<CastlePlace \*>(item);

*if*(castlePlace != *nullptr*)

{

steps->stop();

steps->disconnect();

emit *win*();

*delete* *this*;

*break*;

}

}

}

void EnemyRunner::***stop***()

{

steps->stop();

steps->disconnect();

}

void EnemyRunner::***damaged***(int damage)

{

damage -= armor;

*if*(damage >= 0)

current\_hp -=damage;

*if*(current\_hp <= 0 )

{

emit *dead*(prize);

}

}

QRectF EnemyRunner::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

QPainterPath EnemyRunner::***shape***() *const*

{

QPainterPath path;

path.addEllipse(*boundingRect*());

*return* path;

}

void EnemyRunner::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(QBrush(\*sprite));

painter->setPen(Qt::*NoPen*);

painter->drawRect(0, 0, sprite->width(), sprite->height());

painter->setBrush(Qt::*green*);

painter->drawRect(0, 0, current\_hp\*sprite->width()/max\_hp, 3);

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

EnemyRunner::~***EnemyRunner***()

{

*if* (steps != *nullptr*)

{

steps->stop();

steps->disconnect();

*delete* steps;

}

*delete* sprite;

}

*[Листинг 4 – класс Game](#Содержание)*

#include <QtMultimedia>

#include <QtWidgets>

#include "enemy/IEnemyFactory.h"

#include "hud.h"

#include "level.h"

#include "level/ILevelParser.h"

#include "roadFinder/RoadFinder.h"

#include "tower/buildingui.h"

#include "tower/towerui.h"

*class* **Game** : *public* QWidget

{

Q\_OBJECT

*public*:

**Game**(

QWidget\* parent,

IEnemyFactory \*enemyFactory,

ILevelParser\* levelParser

);

void **gameOver**();

void **win**();

signals:

void **stopEnemy**();

void **change\_lifes**(int);

void **change\_wave**(int);

void **changeMoney**(int);

void **change\_enemy**(int);

void **btn\_wave**(bool);

*public* slots:

void **createEnemies**();

void **spawnEnemy**();

void **wastelifes**();

void **reduceMoney**(int);

void **sellTower**();

void **selectTower**(ITower \*tower = *nullptr*);

void **upgradeTower**();

*private* slots:

void **killEnemy**(int = 0);

*private*:

RoadFinder \* roadFinder;

IEnemyFactory \*enemyFactory;

ITower \* selectedTower;

QMediaPlayer \*music;

ILevelParser\* levelParser;

QTimer \* spawnTimer;

TowerUI \* towerUI;

Level \* level;

HUD\* hud;

BuildingUI build\_ui;

QList<DenPlace \*> \* listDens;

int enemies;

int currentEnemy;

int wave;

int lifes;

int money;

};

#include "enemy/IEnemy.h"

#include "roadFinder/RoadFinder.h"

Game::**Game**(QWidget \*parent , IEnemyFactory \*enemyFactory, ILevelParser\* levelParser)

{

*//\*\*\*\*Background* *image\*\*\*\*//*

QPalette pal;

pal.setBrush(*this*->backgroundRole(), QBrush(QImage(":/res/images/Fon.jpg")));

*this*->setPalette(pal);

*this*->setAutoFillBackground(*true*);

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

adjustSize();

setWindowTitle("Tower Defense");

move(20,20);

*//\*\*\*\*Background* *music\*\*\*\*//*

QMediaPlaylist \*playlist = *new* QMediaPlaylist();

playlist->addMedia(QUrl("qrc:/res/music/back.mp3"));

playlist->setPlaybackMode(QMediaPlaylist::*Loop*);

music = *new* QMediaPlayer(*this*);

music->setPlaylist(playlist);

music->setVolume(1);

music->play();

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*//\*\*\*\*\*\*\*\*Font\*\*\*\*\*\*\*\*//*

int id = QFontDatabase::addApplicationFont(":/res/fonts/CyrilicOld.TTF");

QString family = QFontDatabase::applicationFontFamilies(id).at(0);

QFont f(family);

f.setPixelSize(16);

setFont(f);

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*//\*\*\*\*Initialisation\*\*\*\*//*

*this*->levelParser = levelParser;

*this*->enemyFactory = enemyFactory;

towerUI = *new* TowerUI(*this*);

hud = *new* HUD(*this*);

level = *new* Level( *levelParser->getMap()*, *this*);

listDens = *new* QList<DenPlace \*>();

QList<QGraphicsItem\*> listOfItems = level->items();

*for* (QGraphicsItem\* item : listOfItems)

{

DenPlace \* denPlace = *dynamic\_cast*<DenPlace\*>(item);

*if*(denPlace != *nullptr*)

{

listDens->push\_back(denPlace);

}

}

RoadFinder \*roadFinder = *new* RoadFinder(

level,

levelParser->*getMap*().getTileH(),

levelParser->*getMap*().getTileW()

);

emit roadFinder->calculationMap();

selectedTower = *nullptr*;

wave = 0;

enemies = 0;

currentEnemy = 0;

lifes = listDens->size() \* 7.5;

money = listDens->size() \* 125;

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*//\*\*\*\*SIGNALS\*\*\*\*//*

connect(*this*, SIGNAL(changeMoney(int)), &build\_ui, SLOT(setPossible(int)));

connect(*this*, SIGNAL(changeMoney(int)), towerUI, SLOT(setMoney(int)));

connect(level,SIGNAL(setUI(ITower\*)), *this*, SLOT(selectTower(ITower\*)));

connect(level, SIGNAL(successBuild(int)), *this*, SLOT(reduceMoney(int)));

connect(&build\_ui, SIGNAL(build(ITower\*)), level, SLOT(createTower(ITower\*)));

emit change\_enemy(enemies);

emit change\_lifes(lifes);

emit changeMoney(money);

emit change\_wave(wave);

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*//\*\*\*\*View\*\*\*\*//*

QGraphicsView \* q = *new* QGraphicsView(*this*);

q->setScene(level);

q->setMouseTracking(*true*);

q->setHorizontalScrollBarPolicy(Qt::*ScrollBarAlwaysOff*);

q->setVerticalScrollBarPolicy(Qt::*ScrollBarAlwaysOff*);

q->setRenderHint(QPainter::*Antialiasing*, *false*);

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*//\*\*\*\*Layout\*\*\*\*//*

QVBoxLayout \* v\_box = *new* QVBoxLayout();

QHBoxLayout \* h\_box = *new* QHBoxLayout();

h\_box->addLayout(v\_box);

v\_box->addWidget(hud);

v\_box->addWidget(towerUI);

h\_box->setSizeConstraint(QLayout::*SetFixedSize*);

h\_box->addWidget(q);

h\_box->addWidget(&build\_ui);

v\_box->setSizeConstraint(QLayout::*SetFixedSize*);

*this*->setLayout(h\_box);

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

}

void Game::**gameOver**()

{

QMessageBox \* restart = *new* QMessageBox (QMessageBox::*Question*,"GameOver","Do you want to restart level?", QMessageBox::*Yes*| QMessageBox::*No*);

music->stop();

spawnTimer->stop();

spawnTimer->disconnect();

emit stopEnemy();

int choice = restart->*exec*();

*if*(choice == QMessageBox::*Yes*)

{

Game \* game = *new* Game(*nullptr*, *this*->enemyFactory, *this*->levelParser);

*this*->close();

game->show();

}

*else*

{

close();

}

}

void Game::**wastelifes**()

{

--lifes;

emit change\_lifes(lifes);

--enemies;

emit change\_enemy(enemies);

*if* (lifes <= 0 ) gameOver();

}

void Game::**reduceMoney**(int cash)

{

money-=cash;

emit changeMoney(money);

}

void Game::**selectTower**(ITower \* tower)

{

selectedTower = tower;

towerUI->setUi(selectedTower);

}

void Game::**upgradeTower**()

{

money-=selectedTower->*getCost*();

selectedTower->*upgrade*();

emit changeMoney(money);

}

void Game::**sellTower**()

{

*if*(selectedTower)

{

money+=selectedTower->*getSalePrice*();

level->createPlace(selectedTower->scenePos());

emit changeMoney(money);

level->clearSelection();

level->removeItem(selectedTower);

*delete* selectedTower;

selectedTower = *nullptr*;

towerUI->setUi();

}

}

void Game::**createEnemies**()

{

++wave;

emit change\_wave(wave);

++lifes;

emit change\_lifes(lifes);

money += lifes \* wave \* 5;

emit changeMoney(money);

emit btn\_wave(*false*);

spawnTimer = *new* QTimer();

connect(spawnTimer, SIGNAL(timeout()), *this*, SLOT(spawnEnemy()));

*if*(wave < 100)

spawnTimer->start(1000 - 9 \* wave);

*else*

spawnTimer->start(100);

}

void Game::**spawnEnemy**()

{

QList<IEnemy \*> \* listEnemies = enemyFactory->*createEnemies*(listDens, wave, level);

*for*(IEnemy \* enemy : \*listEnemies)

{

level->addItem(enemy);

++currentEnemy;

++enemies;

emit change\_enemy(enemies);

connect(*this*, SIGNAL(stopEnemy()), enemy, SLOT(stop()));

connect(enemy, SIGNAL(win()), *this*, SLOT(wastelifes()));

connect(enemy, SIGNAL(dead(int)), *this*, SLOT(killEnemy(int)));

*if* (currentEnemy >= 10 + wave \* 2)

{

spawnTimer->disconnect();

currentEnemy = 0;

emit btn\_wave(*true*);

}

}

}

void Game::**killEnemy**(int prize)

{

*delete* sender();

money += prize;

--enemies;

emit change\_enemy(enemies);

emit changeMoney(money);

}

*[Листинг 5 – класс HUD](#Содержание)*

#include <QtWidgets>

*class* **HUD** : *public* QWidget

{

Q\_OBJECT

*public*:

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

~***HUD***();

*public* slots:

void **lifes**(int);

void **money**(int);

void **enemy**(int);

void **wave**(int);

*private*:

QLabel \* l\_lifes ;

QLabel \* l\_money;

QLabel \* l\_wave;

QLabel \* l\_enemy;

QLabel \* l\_score;

QPushButton \* startWave;

QVBoxLayout \* v\_box;

};

HUD::**HUD**(QWidget \*parent) : QWidget(parent)

{

startWave = *new* QPushButton("Start wave!");

l\_wave = *new* QLabel(*this*);

l\_lifes = *new* QLabel(*this*);

l\_money = *new* QLabel(*this*);

l\_enemy = *new* QLabel(*this*);

v\_box = *new* QVBoxLayout(*this*);

setLayout(v\_box);

v\_box->addWidget(startWave);

v\_box->addWidget(l\_wave);

v\_box->addWidget(l\_lifes);

v\_box->addWidget(l\_money);

v\_box->addWidget(l\_enemy);

v\_box->setAlignment(Qt::*AlignTop*);

connect(parent, SIGNAL(change\_lifes(int)), *this*, SLOT(lifes(int)));

connect(parent, SIGNAL(changeMoney(int)), *this*, SLOT(money(int)));

connect(parent, SIGNAL(change\_enemy(int)), *this*, SLOT(enemy(int)));

connect(parent, SIGNAL(change\_wave(int)), *this*, SLOT(wave(int)));

connect(startWave, SIGNAL(clicked()), parent, SLOT(createEnemies()));

connect(parent, SIGNAL(btn\_wave(bool)), startWave, SLOT(setEnabled(bool)));

}

HUD::~***HUD***()

{

*delete* l\_enemy;

*delete* l\_lifes;

*delete* l\_money;

*delete* l\_wave;

*delete* startWave;

*delete* v\_box;

}

void HUD::**lifes**(int \_lifes)

{

l\_lifes->setText(QString("lifes: %1").arg(\_lifes));

}

void HUD::**money**(int cost)

{

l\_money->setText(QString("Money: %1").arg(cost));

}

void HUD::**enemy**(int size)

{

l\_enemy->setText(QString("Number of enemies: %1").arg(size));

}

void HUD::**wave**(int current)

{

l\_wave->setText(QString("Wave: %1").arg(current));

}

*[Листинг 6 – класс Level](#Содержание)*

#include <QGraphicsView>

#include "map.h"

#include "places/towerplace.h"

#include "places/castlePlace.h"

#include "places/denPlace.h"

#include "places/roadPlace.h"

#include "roadFinder/RoadFinder.h"

#include "tower/ITower.h"

*class* **Level** : *public* QGraphicsScene

{

Q\_OBJECT

*public*:

**Level**(Map &\_map,QWidget \* parent = *nullptr* );

void **createPlace**(QPointF);

signals:

void **successBuild**(int);

void **setUI**(ITower\* tower = *nullptr*);

*public* slots:

void **createTower**(ITower \* tower);

*private*:

RoadFinder\* roadFinder;

Map map;

QGraphicsPixmapItem\* build\_curs;

bool building;

int lvl\_h;

int lvl\_w;

ITower \* tower;

void **renderMap**();

*protected*:

void ***mouseMoveEvent***(QGraphicsSceneMouseEvent \* event);

void ***mousePressEvent***(QGraphicsSceneMouseEvent \* event);

};

Level::**Level**(Map & \_map, QWidget \*parent)

{

map = \_map;

build\_curs = *nullptr*;

building = *false*;

renderMap();

}

void Level::**createPlace**(QPointF point)

{

TowerPlace \* place = *new* TowerPlace();

place->setPos(point);

addItem(place);

}

void Level::**renderMap**()

{

*if* (*nullptr* == roadFinder)

roadFinder = *new* RoadFinder(

*this*,

*this*->map.getTileH(),

*this*->map.getTileW()

);

QVector<QVector<int>> sc\_lvl = map.getlvlID();

lvl\_h = map.mapH()\*map.getTileH();

lvl\_w = map.mapW()\*map.getTileW();

setSceneRect(0,0, lvl\_w, lvl\_h);

*for*(int i = 0; i < map.mapH(); ++i)

{

*for* (int j =0 ; j < map.mapW(); ++j)

{

*switch* (sc\_lvl[i][j])

{

*case* 1 :

{

TowerPlace \* tplace = *new* TowerPlace();

tplace->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(tplace);

*break*;

}

*case* 2:

{

RoadPlace \* rplace = *new* RoadPlace();

rplace->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(rplace);

*break*;

}

*case* 3:

{

RoadPlace \* rplace = *new* RoadPlace();

rplace->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(rplace);

DenPlace \* dplace = *new* DenPlace();

dplace->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(dplace);

*break*;

}

*case* 4:

{

QGraphicsPixmapItem \* sc\_obj = *new* QGraphicsPixmapItem (QPixmap(":/res/images/Blue.png"));

sc\_obj->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(sc\_obj);

*break*;

}

*case* 5:

{

RoadPlace \* rplace = *new* RoadPlace();

rplace->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(rplace);

CastlePlace \* cplace = *new* CastlePlace();

cplace->setPos(j\*map.getTileW(), i\*map.getTileH());

addItem(cplace);

*break*;

}

}

}

}

}

void Level::**createTower**(ITower \* tower)

{

*this*->tower = tower;

*if* (build\_curs) removeItem(build\_curs);

build\_curs = *new* QGraphicsPixmapItem();

build\_curs->setPixmap(\*(tower->*getSprite*()));

build\_curs->hide();

addItem(build\_curs);

building = *true*;

}

void Level::***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event)

{

*if* (build\_curs)

{

build\_curs->show();

build\_curs->setPos(event->scenePos().x()-16,event->scenePos().y()-16 );

}

QGraphicsScene::mouseMoveEvent(event);

}

void Level::***mousePressEvent***(QGraphicsSceneMouseEvent \*event)

{

TowerPlace \* place = *dynamic\_cast*<TowerPlace\*>(items(event->scenePos()).last());

ITower \* tow = *dynamic\_cast*<ITower\*>(items(event->scenePos()).last());

clearSelection();

*if*(event->button() == Qt::*LeftButton*)

{

*if* (building == *true* && place != *nullptr*)

{

ITower \* tower = *this*->tower;

*this*->tower = *nullptr*;

tower->setPos(place->scenePos());

*this*->addItem(tower);

*this*->removeItem(place);

*this*->removeItem(build\_curs);

building = *false*;

clearSelection();

tower->setSelected(*true*);

emit setUI(tower);

emit successBuild(tower->*getCost*());

*return*;

}

*if* (building == *true* && place == *nullptr*)

{

QGraphicsScene::mousePressEvent(event);

*return*;

}

*if*(tow != *nullptr* && (!tow->isSelected()))

{

clearSelection();

tow->setSelected(*true*);

emit setUI(tow);

*return*;

}

emit setUI(*nullptr*);

*return*;

}

*if* (event->button() == Qt::*RightButton*)

{

*if* (building == *true* && build\_curs->isVisible())

{

clearSelection();

removeItem(build\_curs);

building = *false*;

}

QGraphicsScene::mousePressEvent(event);

}

}

*[Листинг 7.1 – интерфейс IlevelParser](#Содержание)*

#include "map.h"

#include "road.h"

*class* **ILevelParser** {

*public*:

*//* *The* *method* *for* *provide* *map.*

*virtual* Map& ***getMap***() = 0;

*//* *The* *method* *for* *provide* *main* *road.*

*virtual* Road& ***getRoad***() = 0;

};

*[Листинг 7.2 – класс TextFileLevelParser](#Содержание)*

#include <QString>

#include "ILevelParser.h"

*class* **TextFileLevelParser**: *public* ILevelParser

{

*public*:

**TextFileLevelParser**(QString filePath);

~**TextFileLevelParser**();

Map& ***getMap***() *override*;

Road& ***getRoad***() *override*;

*private*:

QString TAG = "TextFileLevelParser";

Map\* map;

Road\* road;

bool **parseParametersLine**(QString parametersLine);

bool **parseLineOfMap**(QString mapLine, int lineNumber);

bool **parseRoadLine**(QString roadLine);

};

#include <QDebug>

#include <QFile>

#include <QRegularExpression>

#include <QTextStream>

#include "TextFileLevelParser.h"

TextFileLevelParser::**TextFileLevelParser**(QString filePath)

{

QFile\* fileOfMap = *new* QFile(filePath);

*if* (!fileOfMap->*open*(QIODevice::*ReadOnly* | QIODevice::*Text*))

{

qDebug() << TAG << " can't open file " << filePath;

exit(1);

}

TAG = QString(TAG + "[" + filePath + "]");

QTextStream textStream(fileOfMap);

*if* (!parseParametersLine( textStream.readLine() )) exit(-1);

qDebug() << *this*->*getMap*().mapH();

*for* (int mapLine = 0; mapLine < *this*->*getMap*().mapH(); mapLine++)

{

*if* (!parseLineOfMap(textStream.readLine(), mapLine)) exit(-2);

}

*if* (QRegularExpression("^road").match(textStream.readLine()).hasMatch()) {

*if* (!parseRoadLine(textStream.readLine())) exit(-3);

} *else* {

qDebug() << TAG << "road line not found.";

}

}

bool TextFileLevelParser::**parseParametersLine**(QString parametersLine)

{

QStringList propertiesList = parametersLine.split(" ");

*if* (4 != propertiesList.size()) {

qDebug() << TAG

<< " can't parse information line, wrong number of the parameters, "

<< " right format of the parameters line is "

<< "'width=%d height=%d tilewidth=%d tileheight=%d' in any order.";

*return* *false*;

}

int width(0), height(0), tilewidth(0), tileheight(0);

*for* (QString property: propertiesList)

{

QStringList nameValueParameter = property.split("=");

*if* (2 != nameValueParameter.size()) {

qDebug() << TAG << " wrong parameter " << property;

*return* *false*;

}

bool ok;

*if* ("width" == nameValueParameter[0]) {

width = nameValueParameter[1].toInt(&ok);

*if* (!ok) {

qDebug() << " can't parse width '" << property << "'.";

*return* *false*;

}

} *else* *if* ("height" == nameValueParameter[0]) {

height = nameValueParameter[1].toInt(&ok);

*if* (!ok) {

qDebug() << " can't parse height '" << property << "'.";

*return* *false*;

}

} *else* *if* ("tilewidth" == nameValueParameter[0]) {

tilewidth = nameValueParameter[1].toInt(&ok);

*if* (!ok) {

qDebug() << " can't parse tilewidth '" << property << "'.";

*return* *false*;

}

} *else* *if* ("tileheight" == nameValueParameter[0]) {

tileheight = nameValueParameter[1].toInt(&ok);

*if* (!ok) {

qDebug() << " can't parse tileheight '" << property << "'.";

*return* *false*;

}

} *else* {

qDebug() << " unexpected parameter " << nameValueParameter[0];

*return* *false*;

}

}

*this*->map = *new* Map(width, height, tilewidth, tileheight);

*return* *true*;

}

bool TextFileLevelParser::**parseLineOfMap**(QString mapLine, int lineNumber)

{

bool ok;

QVector<int> line;

*for* (QChar number: mapLine)

{

line.push\_back(QString(number).toInt(&ok));

*if* (!ok) {

qDebug() << " can't parse line #" << lineNumber

<< ". Sumbol " << number << " not a number.";

*return* *false*;

}

}

*if* (line.size() != *this*->*getMap*().mapW()) {

qDebug() << " unexpected length of the line #" << lineNumber

<< ". Expected size is " << *this*->*getMap*().mapW()

<< ", but actual size is " << line.size();

*return* *false*;

}

*this*->*getMap*().getlvlID().push\_back(line);

*return* *true*;

}

bool TextFileLevelParser::**parseRoadLine**(QString roadLine)

{

QPolygon roadPoints;

*for* (QString xCommaYPoint: roadLine.split(" "))

{

QStringList xYPair = xCommaYPoint.split(",");

*if* (2 != xYPair.size()) {

qDebug() << " wrong format of the point " << xCommaYPoint;

*return* *false*;

}

bool ok;

int x = xYPair[0].toInt(&ok);

*if* (!ok) {

qDebug() << " 'x' (" << xCommaYPoint << ") not a number .";

*return* *false*;

}

x \*= *this*->*getMap*().getTileW();

int y = xYPair[1].toInt(&ok);

*if* (!ok) {

qDebug() << " 'y' (" << xCommaYPoint << ") not a number .";

*return* *false*;

}

y \*= *this*->*getMap*().getTileH();

*//* *qDebug()* *<<* *x* *<<* *y;*

roadPoints << QPoint(x, y);

}

*this*->road = *new* Road(roadPoints.at(0), roadPoints);

*return* *true*;

}

*[Листинг 8 – класс Map](#Содержание)*

#include <QVector>

*class* **Map**

{

*public*:

**Map**(int = 0 , int = 0, int = 0 , int = 0);

int **getTileW**() *const*;

int **getTileH**() *const*;

QVector<QVector<int> > &**getlvlID**();

~**Map**();

int **mapH**() *const*;

int **mapW**() *const*;

*private*:

QVector<QVector<int>>\* lvlID;

int tile\_w;

int tile\_h;

int map\_w;

int map\_h;

};

Map::**Map**(int w, int h, int w\_t, int h\_t)

{

map\_w = w;

map\_h = h;

tile\_w = w\_t;

tile\_h = h\_t;

lvlID = *new* QVector<QVector<int>>();

}

Map::~**Map**()

{

*delete* lvlID;

}

*[Листинг 9 – класс RoadFinder](#Содержание)*

#pragma once

#include <QGraphicsScene>

#include "places/roadPlace.h"

*class* **RoadFinder** {

*public*:

**RoadFinder**(

QGraphicsScene\* level,

int cellHeight,

int cellWidth

);

*public* slots:

void **calculationMap**();

*private*:

int cellWidth;

int cellHeight;

QGraphicsScene\* level;

QPointF castle;

void **calculationPlacesWithTowers**();

void **calculationWaysFromCastle**();

void **compareAndCalculationWeightWithCurrent**(

RoadPlace \*roadPlace,

RoadPlace \*currentRoadPlace,

QMap<float, QList<RoadPlace \*>\*> \*calculationPoints

);

void **printAllMap**();

};

#include <QDebug>

#include <QGraphicsItem>

#include <QList>

#include <QPointF>

#include "RoadFinder.h"

#include "places/castlePlace.h"

#include "places/roadPlace.h"

#include "places/towerplace.h"

#include "tower/ITower.h"

#include <QTime>

RoadFinder::**RoadFinder**(

QGraphicsScene\* level,

int cellHeight,

int cellWidth

)

{

*this*->level = level;

*this*->cellWidth = cellWidth;

*this*->cellHeight = cellHeight;

}

void RoadFinder::**calculationMap**()

{

QTime time = QTime().currentTime();

calculationPlacesWithTowers();

calculationWaysFromCastle();

qDebug()<<"msec for calculation map: "<< time.msecsTo(QTime().currentTime());

}

void RoadFinder::**calculationPlacesWithTowers**()

{

*if* (*nullptr* == level) {

qDebug() << "RF: level is nil.";

*return*;

}

QList<QGraphicsItem\*> listOfItems = level->items();

*for* (QGraphicsItem\* item : listOfItems)

{

RoadPlace \* roadPlace = *dynamic\_cast*<RoadPlace\*>(item);

*if*(roadPlace != *nullptr*)

{

roadPlace->initializationAllWeight();

}

}

listOfItems = level->items();

*for* (QGraphicsItem\* item : listOfItems)

{

ITower\* iTower = *dynamic\_cast*<ITower\*>(item);

*if* (iTower != *nullptr*)

{

*if*(iTower->*getTowerArea*() == *nullptr*)

*break*;

QList<QGraphicsItem \*> colliding\_items = iTower->*getTowerArea*()->collidingItems(Qt::*IntersectsItemShape*);

*for* (int i = 0, n = colliding\_items.size(); i < n; ++i)

{

RoadPlace \*roadPlace = *dynamic\_cast*<RoadPlace \*>(colliding\_items[i]);

*if*(roadPlace != *nullptr*)

{

roadPlace->increaseDamageWeight(iTower->*getPower*() \* iTower->*getFiringRate*());

*//qDebug()<<* *"newweight:* *"<<* *roadPlace->getWeight();*

}

}

}

CastlePlace \* castlePlace = *dynamic\_cast*<CastlePlace\*>(item);

*if*(castlePlace != *nullptr*)

{

castle = castlePlace->pos();

}

}

}

void RoadFinder::**calculationWaysFromCastle**()

{

QTransform \* qtransform = *new* QTransform();

QMap<float, QList<RoadPlace \*>\*> calculationPoints;

RoadPlace \*roadPlace;

QGraphicsItem \* upperItem = level->itemAt(castle.x(), castle.y(), \*qtransform);

QList<QGraphicsItem \*> listItems = upperItem->collidingItems();

*for*(QGraphicsItem \* item : listItems)

{

roadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(roadPlace != *nullptr*)

*break*;

}

*if*( roadPlace != *nullptr*)

{

roadPlace->increaseWeight(1);

*if*(!calculationPoints.contains(roadPlace->getWeight()))

calculationPoints.insert(roadPlace->getWeight(), *new* QList<RoadPlace \*>());

calculationPoints[roadPlace->getWeight()]->push\_back(roadPlace);

}

*while*( !calculationPoints.isEmpty() )

{

RoadPlace \*currentRoadPlace = calculationPoints.first()->first();

*//qDebug()<<"xcp:* *"<<currentRoadPlace->x()<<"\tycp:* *"<<currentRoadPlace->y();*

upperItem = level->itemAt(currentRoadPlace->pos().x(), currentRoadPlace->pos().y() + cellHeight, \*qtransform);

*if*(upperItem != *nullptr*)

{

roadPlace = *nullptr*;

roadPlace = *dynamic\_cast*<RoadPlace \*>(upperItem);

*if*(*nullptr* == roadPlace)

{

listItems = upperItem->collidingItems();

*for*(QGraphicsItem \* item : listItems)

{

roadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(roadPlace != *nullptr*)

*break*;

}

}

*if*( roadPlace != *nullptr*)

compareAndCalculationWeightWithCurrent(roadPlace, currentRoadPlace, &calculationPoints);

}

upperItem = (level->itemAt(currentRoadPlace->pos().x(), currentRoadPlace->pos().y() - cellHeight, \*qtransform));

*if*(upperItem != *nullptr*)

{

roadPlace = *nullptr*;

roadPlace = *dynamic\_cast*<RoadPlace \*>(upperItem);

*if*(*nullptr* == roadPlace)

{

listItems = upperItem->collidingItems();

*for*(QGraphicsItem \* item : listItems)

{

roadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(roadPlace != *nullptr*)

*break*;

}

}

*if*( roadPlace != *nullptr*)

compareAndCalculationWeightWithCurrent(roadPlace, currentRoadPlace, &calculationPoints);

}

upperItem = (level->itemAt(currentRoadPlace->pos().x() + cellWidth, currentRoadPlace->pos().y(), \*qtransform));

*if*(upperItem != *nullptr*)

{

roadPlace = *nullptr*;

roadPlace = *dynamic\_cast*<RoadPlace \*>(upperItem);

*if*(*nullptr* == roadPlace)

{

listItems = upperItem->collidingItems();

*for*(QGraphicsItem \* item : listItems)

{

roadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(roadPlace != *nullptr*)

*break*;

}

}

*if*( roadPlace != *nullptr*)

compareAndCalculationWeightWithCurrent(roadPlace, currentRoadPlace, &calculationPoints);

}

upperItem = (level->itemAt(currentRoadPlace->pos().x() - cellWidth, currentRoadPlace->pos().y(), \*qtransform));

*if*(upperItem != *nullptr*)

{

roadPlace = *nullptr*;

roadPlace = *dynamic\_cast*<RoadPlace \*>(upperItem);

*if*(*nullptr* == roadPlace)

{

listItems = upperItem->collidingItems();

*for*(QGraphicsItem \* item : listItems)

{

roadPlace = *dynamic\_cast*<RoadPlace \*>(item);

*if*(roadPlace != *nullptr*)

*break*;

}

}

*if*( roadPlace != *nullptr*)

compareAndCalculationWeightWithCurrent(roadPlace, currentRoadPlace, &calculationPoints);

}

calculationPoints.first()->removeFirst();

*if*(calculationPoints.first()->isEmpty())

calculationPoints.remove(calculationPoints.firstKey());

}

}

void RoadFinder::**compareAndCalculationWeightWithCurrent**(

RoadPlace \*roadPlace,

RoadPlace \*currentRoadPlace,

QMap<float, QList<RoadPlace \*>\*> \*calculationPoints

)

{

*if*( roadPlace != *nullptr*)

{

*//* *qDebug()<<"xrp:* *"<<roadPlace->x()<<"\tyrp:* *"<<roadPlace->y();*

*if*(roadPlace->getWeight() == roadPlace->BASEWEIGHT + roadPlace->getDamageWeight())

{

roadPlace->increaseWeight(currentRoadPlace->getWeight());

*if*(!calculationPoints->contains(roadPlace->getWeight()))

calculationPoints->insert(roadPlace->getWeight(), *new* QList<RoadPlace \*>());

(\*calculationPoints)[roadPlace->getWeight()]->push\_back(roadPlace);

}*else*

*if*(roadPlace->getWeight() - roadPlace->getDamageWeight() - roadPlace->BASEWEIGHT > currentRoadPlace->getWeight())

{

roadPlace->initializationWeight();

roadPlace->increaseWeight(currentRoadPlace->getWeight() + roadPlace->getDamageWeight());

*if*(!calculationPoints->contains(roadPlace->getWeight()))

calculationPoints->insert(roadPlace->getWeight(), *new* QList<RoadPlace \*>());

(\*calculationPoints)[roadPlace->getWeight()]->push\_back(roadPlace);

}

}

}

void RoadFinder::**printAllMap**() {

qDebug() << "RoadFinder::printAllMap() START";

QList<QGraphicsItem\*> listOfAllItems = level->items();

*for* (QGraphicsItem\* item: listOfAllItems) {

int x = item->pos().x() / cellHeight;

int y = item->pos().y() / cellWidth;

RoadPlace\* itemAsRoadPlace = *dynamic\_cast*<RoadPlace\*>(item);

*if* (*nullptr* == itemAsRoadPlace) {

qDebug() << "[" << x << ", " << y << "] is not RoadPlace.";

*continue*;

}

qDebug() << "[" << x << ", " << y << "] is RoadPlace and weight is "

<< itemAsRoadPlace->getWeight();

}

qDebug() << "RoadFinder::printAllMap() END";

}

*[Листинг 10 – класс TowerPlace](#Содержание)*

#include <QGraphicsObject>

#include <QPainter>

*class* **TowerPlace** : *public* QGraphicsObject

{

Q\_OBJECT

*public*:

**TowerPlace**();

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

*private*:

QPixmap \* sprite;

};

TowerPlace::**TowerPlace**()

{

sprite = *new* QPixmap(":/res/images/TowerPlace.png");

}

QRectF TowerPlace::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

void TowerPlace::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(QBrush(\*sprite));

painter->setPen(Qt::*NoPen*);

painter->drawRect(0, 0, sprite->width(), sprite->height());

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

*[Листинг 11 – класс CastlePlace](#Содержание)*

#include <QGraphicsObject>

#include <QPainter>

*class* **CastlePlace** : *public* QGraphicsObject

{

*public*:

**CastlePlace**();

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

*private*:

QPixmap \* sprite;

};

CastlePlace::**CastlePlace**()

{

sprite = *new* QPixmap(":/res/images/Red.png");

}

QRectF CastlePlace::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

void CastlePlace::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(QBrush(\*sprite));

painter->setPen(Qt::*NoPen*);

painter->drawRect(0, 0, sprite->width(), sprite->height());

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

*[Листинг 12 – класс RoadPlace](#Содержание)*

#include <QGraphicsObject>

#include <QPainter>

*class* **RoadPlace** : *public* QGraphicsObject

{

Q\_OBJECT

*public*:

*const* float BASEWEIGHT = 1;

**RoadPlace**();

void **increaseWeight**(int);

void **increaseDamageWeight**(int);

void **decreaseWeight**(int);

void **initializationWeight**();

void **initializationDamageWeight**();

void **initializationAllWeight**();

float **getWeight**();

float **getDamageWeight**();

QPixmap\* **getSprite**();

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

*private*:

QPixmap \* sprite;

float weight = BASEWEIGHT;

float damageWeight = 0;

};

RoadPlace::**RoadPlace**()

{

sprite = *new* QPixmap(":/res/images/Road.png");

}

void RoadPlace::**increaseWeight**(int iWeight)

{

weight += iWeight;

}

void RoadPlace::**increaseDamageWeight**(int idWeight)

{

damageWeight += idWeight;

increaseWeight(idWeight);

}

void RoadPlace::**decreaseWeight**(int dWeight)

{

*if*(weight > dWeight)

weight -= dWeight;

*else*

weight = 0;

}

void RoadPlace::**initializationWeight**()

{

weight = BASEWEIGHT + damageWeight;

}

void RoadPlace::**initializationDamageWeight**()

{

decreaseWeight(damageWeight);

damageWeight = 0;

}

void RoadPlace::**initializationAllWeight**()

{

weight = BASEWEIGHT;

damageWeight = 0;

}

QRectF RoadPlace::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

void RoadPlace::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(QBrush(\*sprite));

painter->setPen(Qt::*NoPen*);

painter->drawRect(0, 0, sprite->width(), sprite->height());

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

*[Листинг 13 – класс DenPlace](#Содержание)*

#include <QGraphicsObject>

#include <QPainter>

*class* **DenPlace** : *public* QGraphicsObject

{

Q\_OBJECT

*public*:

**DenPlace**();

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

*private*:

QPixmap \* sprite;

};

DenPlace::**DenPlace**()

{

sprite = *new* QPixmap(":/res/images/Spawn.png");

}

QRectF DenPlace::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

void DenPlace::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(QBrush(\*sprite));

painter->setPen(Qt::*NoPen*);

painter->drawRect(0, 0, sprite->width(), sprite->height());

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

*[Листинг 14.1 – интерфейс ITower](#Содержание)*

#include <QtWidgets>

#include <QGraphicsObject>

#include "tower/towerarea.h"

*class* ITower : *public* QGraphicsObject

{

Q\_OBJECT

*public*:

*virtual* float getMULTIPLIERRADIUS() = 0;

*virtual* float getMULTIPLIERPOWER() = 0;

*virtual* float getMULTIPLIERCOST() = 0;

*virtual* float getMULTIPLIERFIRINGRATE() = 0;

*virtual* float getBASEFIRINGRATE() = 0;

*virtual* int getBASECOST() = 0;

*virtual* int getBASEPOWER() = 0;

*virtual* int getBASERADIUS() = 0;

*virtual* void setArea() = 0;

*virtual* void fire() = 0;

*virtual* QPixmap\* getSprite() = 0;

*virtual* int getLevel() = 0;

*virtual* int getPower() = 0;

*virtual* int getRadius() = 0;

*virtual* int getCost() = 0;

*virtual* int getSalePrice() = 0;

*virtual* float getFiringRate() = 0;

*virtual* TowerArea\* getTowerArea() = 0;

*virtual* ITower \* copyTower() = 0;

*signals*:

*virtual* void updateTower(ITower\*) = 0;

*public* *slots*:

*virtual* void chooseEnemy() = 0;

*virtual* void upgrade() = 0;

*protected*:

*virtual* QRectF boundingRect() *const* = 0;

*virtual* void paint(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget) = 0;

*virtual* void hoverEnterEvent(QGraphicsSceneHoverEvent \* event) = 0;

*virtual* void hoverLeaveEvent(QGraphicsSceneHoverEvent \* event) = 0;

};

*[Листинг 14.2 – классы Tower, TowerAntiArmor, TowerRapidFire](#Содержание)*

#include <QtWidgets>

#include <QPainter>

#include "bullet.h"

#include "tower/ITower.h"

#include "tower/towerarea.h"

*class* **TowerRapidFire** : *public* ITower

{

Q\_OBJECT

*public*:

*const* float MULTIPLIERRADIUS = 1.1;

*const* float MULTIPLIERPOWER = 1.45;

*const* float MULTIPLIERCOST = 2.15;

*const* float MULTIPLIERFIRINGRATE= 1.3;

*const* float BASEFIRINGRATE = 5;

*const* int BASECOST = 80;

*const* int BASERADIUS = 130;

*const* int BASEPOWER = 7;

*class* **TowerAntiArmor** : *public* ITower

{

Q\_OBJECT

*public*:

*const* float MULTIPLIERRADIUS = 1.2;

*const* float MULTIPLIERPOWER = 2.3;

*const* float MULTIPLIERCOST = 2.25;

*const* float MULTIPLIERFIRINGRATE= 1.05;

*const* float BASEFIRINGRATE = 0.2;

*const* int BASECOST = 200;

*const* int BASERADIUS = 350;

*const* int BASEPOWER = 150;

*class* **Tower** : *public* ITower

{

Q\_OBJECT

*public*:

*const* float MULTIPLIERRADIUS = 1.15;

*const* float MULTIPLIERPOWER = 1.75;

*const* float MULTIPLIERCOST = 2.1;

*const* float MULTIPLIERFIRINGRATE= 1.05;

*const* float BASEFIRINGRATE = 0.9;

*const* int BASECOST = 50;

*const* int BASERADIUS = 200;

*const* int BASEPOWER = 35;

float ***getMULTIPLIERRADIUS***();

float ***getMULTIPLIERPOWER***();

float ***getMULTIPLIERCOST***();

float ***getMULTIPLIERFIRINGRATE***();

float ***getBASEFIRINGRATE***();

int ***getBASECOST***();

int ***getBASEPOWER***();

int ***getBASERADIUS***();

**Tower**(QGraphicsObject \* parent = *nullptr*);

void ***setArea***();

void ***fire***();

ITower \* ***copyTower***();

QPixmap\* ***getSprite***();

int ***getLevel***();

int ***getPower***();

int ***getRadius***();

int ***getCost***();

int ***getSalePrice***();

float ***getFiringRate***();

TowerArea\* ***getTowerArea***();

~***Tower***();

signals:

void ***updateTower***(ITower\*);

*public* slots:

void ***chooseEnemy***();

void ***upgrade***();

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

void ***hoverEnterEvent***(QGraphicsSceneHoverEvent \* event);

void ***hoverLeaveEvent***(QGraphicsSceneHoverEvent \* event);

*private*:

QPixmap \* sprite = *new* QPixmap(":/res/images/Tower.png");

TowerArea\* area;

QPointF enemyPlace;

QTimer \* detecter;

IEnemy \* target;

double centerX;

double centerY;

double l\_centerX;

double l\_centerY;

int point;

int radius = BASERADIUS;

int power = BASEPOWER;

int cost = BASECOST;

int level = 0;

int salePrice = 0;

float firingRate = BASEFIRINGRATE;

};

Tower::**Tower**(QGraphicsObject \*parent)

{

area = *nullptr*;

setAcceptHoverEvents(*true*);

setFlag(QGraphicsItem::*ItemIsSelectable*);

l\_centerX = sprite->width()/2;

l\_centerY = sprite->height()/2;

detecter = *new* QTimer();

connect(detecter, SIGNAL(timeout()), *this*, SLOT(chooseEnemy()));

detecter->start(1000/firingRate);

}

void Tower::***setArea***()

{

centerX = scenePos().x() + l\_centerX;

centerY = scenePos().y() + l\_centerY;

area->setPos(centerX-radius/2, centerY-radius/2);

}

void Tower::***chooseEnemy***()

{

*if*(area == *nullptr*)

*return*;

QList<QGraphicsItem \*> colliding\_items = area->collidingItems(Qt::*IntersectsItemShape*);

int max = 0;

*for* (int i = 0, n = colliding\_items.size(); i < n; ++i)

{

IEnemy \*enemy = *dynamic\_cast*<IEnemy \*>(colliding\_items[i]);

*//выбирает* *в* *цель* *врага* *с* *самым* *большим* *пройденным* *путём*

*if*(enemy && enemy->*getpassedWay*() > max)

{

max = enemy->*getpassedWay*();

target = enemy;

}

}

*if*(max)

{

enemyPlace = QPointF(target->pos());

*fire*();

}

}

void Tower::***upgrade***()

{

*if*(!isSelected())

{

disconnect(sender(),SIGNAL(clicked()), *this*, SLOT(upgrade()));

}

*else*

{

*if*(INT\_MAX / MULTIPLIERRADIUS > radius)

radius \*= MULTIPLIERRADIUS;

*else*

radius = INT\_MAX;

*if*(INT\_MAX / MULTIPLIERPOWER > power)

power \*= MULTIPLIERPOWER;

*else*

power = INT\_MAX;

*if*(radius == INT\_MAX && power == INT\_MAX)

cost = 0;

*else*

++level;

*if*(INT\_MAX / MULTIPLIERCOST > cost)

cost \*= MULTIPLIERCOST;

*else*

cost = INT\_MAX;

*if*(INT\_MAX - cost/2 > salePrice)

salePrice += cost/2;

*else*

salePrice = INT\_MAX;

firingRate \*= MULTIPLIERFIRINGRATE;

detecter->start(1000/firingRate);

*switch* (level)

{

*case* 5:

{

power \*= 2;

*break*;

}

*case* 20:

{

power \*= 3;

*break*;

}

*default*: *break*;

}

emit *updateTower*(*this*);

}

}

void Tower::***fire***()

{

Bullet\* bullet = *new* Bullet(*this*);

bullet->setPos(centerX, centerY);

QLineF ln(QPointF(centerX-l\_centerX,centerY-l\_centerY), enemyPlace);

double angle = -1\*ln.angle();

bullet->setRotation(angle);

scene()->addItem(bullet);

target->*damaged*(power);

}

ITower \* Tower::***copyTower***()

{

Tower \*tower = *new* Tower();

tower->sprite = *this*->sprite;

tower->radius = *this*->radius;

tower->power = *this*->power;

tower->cost = *this*->cost;

tower->radius = *this*->radius;

tower->level = *this*->level;

tower->salePrice = *this*->salePrice;

tower->firingRate= *this*->firingRate;

*return* tower;

}

Tower::~***Tower***()

{

*delete* sprite;

*delete* area;

*delete* detecter;

}

QRectF Tower::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

void Tower::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

*if*(area == *nullptr*)

{

area = *new* TowerArea(*this*);

scene()->addItem(area);

area->setRadius(radius);

*setArea*();

area->show();

}

*else*

{

area->setRadius(radius);

*setArea*();

}

painter->setBrush(QBrush(\*sprite));

painter->drawRect(0, 0, sprite->width(), sprite->height());

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

void Tower::***hoverEnterEvent***(QGraphicsSceneHoverEvent \*event)

{

*if*(!area->isVisible())

{

area->show();

}

QGraphicsItem::hoverEnterEvent(event);

}

void Tower::***hoverLeaveEvent***(QGraphicsSceneHoverEvent \*event)

{

*if*(area->isVisible())

{

area->hide();

}

QGraphicsItem::hoverLeaveEvent(event);

}

*[Листинг 15 – класс TowerArea](#Содержание)*

#pragma once

#include <QGraphicsObject>

#include <QPainter>

*class* TowerArea : *public* QGraphicsObject

{

Q\_OBJECT

*public*:

TowerArea(QGraphicsObject \*parent = *nullptr*);

void setRadius(int);

*protected*:

void paint(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget = *nullptr*);

QRectF boundingRect() *const*;

QPainterPath shape() *const*;

*private*:

int radius;

bool visible;

};

TowerArea::**TowerArea**( QGraphicsObject \*parent)

{

visible = *false*;

}

void TowerArea::**setRadius**(int r)

{

radius = r;

}

void TowerArea::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setPen(QPen(Qt::*darkGreen*, 3, Qt::*SolidLine*));

painter->drawEllipse(0, 0, radius, radius);

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

QRectF TowerArea::***boundingRect***() *const*

{

*return* QRectF(0, 0, radius, radius);

}

QPainterPath TowerArea::***shape***() *const*

{

QPainterPath path;

path.addEllipse(*boundingRect*());

*return* path;

}

*[Листинг 16 – класс TowerUI](#Содержание)*

#pragma once

#include <QtWidgets>

#include "tower/ITower.h"

*class* **TowerUI** : *public* QWidget

{

Q\_OBJECT

*public*:

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

*public* slots:

void **setUi**(ITower\* tower = *nullptr*);

void **checkUp**();

void **setMoney**(int);

*private*:

QPushButton \* upgrade;

QPushButton \* sell;

QLabel \* power;

QLabel \* radius;

QLabel \* level;

QLabel \* firingRate;

int levelTower;

int costUp;

int money;

};

#include <math.h>

TowerUI::**TowerUI**(QWidget \*parent) : QWidget(parent)

{

setFixedSize(220,145);

power = *new* QLabel("Power: ", *this*);

radius = *new* QLabel("Radius: ", *this*);

level = *new* QLabel("Level: ", *this*);

firingRate = *new* QLabel("Shoot per sec: ", *this*);

sell = *new* QPushButton("Sell", *this*);

sell->setEnabled(*false*);

upgrade = *new* QPushButton(*this*);

upgrade->setEnabled(*false*);

levelTower = 999;

QVBoxLayout \* v\_box = *new* QVBoxLayout();

QHBoxLayout \* h\_box = *new* QHBoxLayout();

h\_box->addWidget(upgrade);

h\_box->addWidget(sell);

v\_box->addWidget(level);

v\_box->addWidget(power);

v\_box->addWidget(radius);

v\_box->addWidget(firingRate);

v\_box->addLayout(h\_box);

v\_box->setAlignment(Qt::*AlignBottom*);

setLayout(v\_box);

setUi();

connect(upgrade, SIGNAL(clicked()), parent, SLOT(upgradeTower()));

connect(sell, SIGNAL(clicked()), parent, SLOT(sellTower()));

}

void TowerUI::**setUi**(ITower \* tower)

{

*if*(tower != *nullptr*)

{

levelTower = tower->*getLevel*();

costUp = tower->*getCost*();

level->setText(QString("Level: %1 (%2)").arg(levelTower).arg(levelTower+1));

power->setText(QString("Power: %1 (+%2)").arg(tower->*getPower*()).arg((int)(tower->*getPower*() \* (tower->*getMULTIPLIERPOWER*() - 1))));

radius->setText(QString("Radius: %1 (+%2)").arg(tower->*getRadius*()).arg((int)(tower->*getRadius*() \* (tower->*getMULTIPLIERRADIUS*() - 1))));

firingRate->setText(QString("Shoot per sec: %1 (+%2)").arg(tower->*getFiringRate*()).arg(tower->*getFiringRate*() \* (tower->*getMULTIPLIERFIRINGRATE*() - 1)));

sell->setText(QString("Sell %1").arg(tower->*getSalePrice*()));

upgrade->setText(QString("Upgrade %1").arg(costUp));

checkUp();

sell->setEnabled(*true*);

connect(tower, SIGNAL(updateTower(ITower\*)), *this*, SLOT(setUi(ITower\*)));

}

*else*

{

levelTower = -1;

upgrade->setText("Upgrade");

upgrade->setEnabled(*false*);

sell->setText("Sell");

sell->setEnabled(*false*);

power->setText("Power: ");

radius->setText("Radius: ");

level->setText("Level: ");

firingRate->setText("Shoot per sec: ");

}

}

void TowerUI::**checkUp**()

{

*if*(money >= costUp)

{

upgrade->setEnabled(*true*);

}

*else*

{

upgrade->setEnabled(*false*);

}

}

void TowerUI::**setMoney**(int \_money)

{

money = \_money;

checkUp();

}

*[Листинг 17 – класс BuildingUI](#Содержание)*

#include <QWidget>

#include "tower/uiobject.h"

#include "tower/ITower.h"

*class* **BuildingUI** : *public* QWidget

{

Q\_OBJECT

*public*:

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

*private*:

QVBoxLayout v\_box;

QWidgetList list;

*public* slots:

void **sendBuild**(ITower\*);

void **setPossible**(int money);

signals:

void **build**(ITower\*);

};

#include "tower/buildingui.h"

#include "tower/tower.h"

#include "tower/towerRapidFire.h"

#include "tower/towerAntiArmor.h"

BuildingUI::**BuildingUI**(QWidget \*parent) : QWidget(parent)

{

v\_box.setAlignment(Qt::*AlignTop*);

QLabel \* name = *new* QLabel("<b>Towers:<b>");

v\_box.addWidget(name);

UiObject \* baseTower = *new* UiObject(*nullptr*, *new* Tower());

QLabel \* costBT = *new* QLabel(QString("Cost: %1").arg(baseTower->getTower()->*getBASECOST*()));

QLabel \* radiusBT = *new* QLabel(QString("Radius: %1").arg(baseTower->getTower()->*getBASERADIUS*()));

QLabel \* powerBT = *new* QLabel(QString("Power: %1").arg(baseTower->getTower()->*getBASEPOWER*()));

QLabel \* firingRateBT = *new* QLabel(QString("Shoot per sec: %1").arg(baseTower->getTower()->*getBASEFIRINGRATE*()));

connect(baseTower, SIGNAL(create(ITower\*)), *this*, SLOT(sendBuild(ITower\*)));

list.push\_back(baseTower);

list.push\_back(costBT);

list.push\_back(radiusBT);

list.push\_back(powerBT);

list.push\_back(firingRateBT);

UiObject \* rapidFireTower = *new* UiObject(*nullptr*, *new* TowerRapidFire());

QLabel \* costRFT = *new* QLabel(QString("Cost: %1").arg(rapidFireTower->getTower()->*getBASECOST*()));

QLabel \* radiusRFT = *new* QLabel(QString("Radius: %1").arg(rapidFireTower->getTower()->*getBASERADIUS*()));

QLabel \* powerRFT = *new* QLabel(QString("Power: %1").arg(rapidFireTower->getTower()->*getBASEPOWER*()));

QLabel \* firingRateRFT = *new* QLabel(QString("Shoot per sec: %1").arg(rapidFireTower->getTower()->*getBASEFIRINGRATE*()));

connect(rapidFireTower, SIGNAL(create(ITower\*)), *this*, SLOT(sendBuild(ITower\*)));

list.push\_back(rapidFireTower);

list.push\_back(costRFT);

list.push\_back(radiusRFT);

list.push\_back(powerRFT);

list.push\_back(firingRateRFT);

UiObject \* towerAntiArmor = *new* UiObject(*nullptr*, *new* TowerAntiArmor());

QLabel \* costTAA = *new* QLabel(QString("Cost: %1").arg(towerAntiArmor->getTower()->*getBASECOST*()));

QLabel \* radiusTAA = *new* QLabel(QString("Radius: %1").arg(towerAntiArmor->getTower()->*getBASERADIUS*()));

QLabel \* powerTAA = *new* QLabel(QString("Power: %1").arg(towerAntiArmor->getTower()->*getBASEPOWER*()));

QLabel \* firingRateTAA = *new* QLabel(QString("Shoot per sec: %1").arg(towerAntiArmor->getTower()->*getBASEFIRINGRATE*()));

connect(towerAntiArmor, SIGNAL(create(ITower\*)), *this*, SLOT(sendBuild(ITower\*)));

list.push\_back(towerAntiArmor);

list.push\_back(costTAA);

list.push\_back(radiusTAA);

list.push\_back(powerTAA);

list.push\_back(firingRateTAA);

*for*(*auto* wgt : list)

{

v\_box.addWidget(wgt);

}

*this*->setContentsMargins(1,1,1,1);

*this*->setLayout(&v\_box);

}

void BuildingUI::**sendBuild**(ITower\* tower)

{

emit build(tower);

}

void BuildingUI::**setPossible**(int money)

{

*for*(*auto* i : list)

{

*if* (UiObject \* obj = *dynamic\_cast*<UiObject\*>(i))

{

*if*(money < obj->getTower()->*getBASECOST*())

{

i->setEnabled(*false*);

}

*else*

{

i->setEnabled(*true*);

}

}

}

}

*[Листинг 18 – класс UiObject](#Содержание)*

#include <QtWidgets>

#include "tower/ITower.h"

*class* UiObject : *public* QLabel

{

Q\_OBJECT

*public*:

UiObject(QWidget \*parent, ITower \* tower);

ITower\* getTower();

*protected*:

void mousePressEvent(QMouseEvent \* event);

*private*:

ITower \* tower;

QPixmap \* sprite;

*signals*:

void create(ITower\*);

};

UiObject::UiObject(QWidget \* parent, ITower \* tower)

{

sprite = *new* QPixmap(\*(tower->getSprite()));

*this*->tower = tower;

*this*->setPixmap(\*sprite);

*this*->setFixedSize(sprite->width(),sprite->height());

}

ITower\* UiObject::getTower()

{

*return* tower;

}

void UiObject::mousePressEvent(QMouseEvent \*event)

{

*emit* create(tower->copyTower());

}

#include <QGraphicsObject>

#include <QTimer>

#include <QPainter>

#include "enemy/IEnemy.h"

*[Листинг 19 – класс Bullet](#Содержание)*

*class* **Bullet**: *public* QGraphicsObject

{

Q\_OBJECT

*public*:

**Bullet**(QGraphicsObject \* parent);

~***Bullet***();

*public* slots:

void **move**();

*private*:

QPixmap \* sprite;

QTimer \* move\_timer;

int STEP\_SIZE;

double theta;

double dy;

double dx;

*protected*:

QRectF ***boundingRect***() *const*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget);

QPainterPath ***shape***() *const*;

};

#include <QtMath>

Bullet::**Bullet**(QGraphicsObject \* parent)

{

STEP\_SIZE = 20;

sprite = *new* QPixmap(":/res/images/Arrow.png");

move\_timer = *new* QTimer(*this*);

connect(move\_timer, SIGNAL(timeout()), *this*, SLOT(move()));

move\_timer->start(30);

}

void Bullet::**move**()

{

IEnemy\* enemy = *static\_cast*<IEnemy\*>(collidingItems().first());

*if*(enemy)

{

move\_timer->stop();

move\_timer->disconnect();

*delete* *this*;

}

*else*

{

theta = rotation();

dy = STEP\_SIZE \* qSin(qDegreesToRadians(theta));

dx = STEP\_SIZE \* qCos(qDegreesToRadians(theta));

moveBy(dx,dy);

}

}

QRectF Bullet::***boundingRect***() *const*

{

*return* QRectF(0, 0, sprite->width(), sprite->height());

}

void Bullet::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(QBrush(\*sprite));

painter->setPen(Qt::*NoPen*);

painter->drawRect(0, 0, sprite->width(), sprite->height());

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

QPainterPath Bullet::***shape***() *const*

{

QPainterPath path;

path.addEllipse(*boundingRect*());

*return* path;

}

Bullet::~***Bullet***()

{

*delete* sprite;

move\_timer->stop();

move\_timer->disconnect();

*delete* move\_timer;

}

**[Литература](#Содержание)**

<https://ru.wikipedia.org>

<https://ravesli.com>

<https://www.youtube.com>

<https://github.com>

<https://habr.com>

<https://prog-cpp.ru>

<https://refactoring.guru>

<https://stackoverflow.com>

<https://doc.qt.io>

встроенная в QT Creator документация