Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

К ЗАЩИТЕ ДОПУСТИТЬ

_____*Е.В. Богдан*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ИГРА «МОРСКОЙ БОЙ»

БГУИР КП 1–40 02 01 222 ПЗ

Студент: Мацур И.А.

Руководитель: Богдан Е.В.

Минск 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой

_____
(подпись)
_____2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Мацуру Ивану Александровичу*
1. Тема проекта *Игра «Морской бой»*
2. Срок сдачи студентом законченного проекта *15 декабря 2023 г.*
3. Исходные данные к проекту *Язык программирования – C++, библиотека Qt, среда QT Creator*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
1. Лист задания.
2. Введение.
3. Обзор литературы.
3.1. Обзор методов и алгоритмов решения поставленной задачи.
4. Функциональное проектирование.
4.1. Структура входных и выходных данных.
4.2. Разработка диаграммы классов.
4.3. Описание классов.
5. Разработка программных модулей.
5.1. Разработка схем алгоритмов (два важных метода).
5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).
6. Результаты работы.
7. Заключение

8. Литература
9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.
2. Схема алгоритма проверки победителя.
3. Схема алгоритма логики выстрелов.

6. Консультант по проекту (с обозначением разделов проекта) Е. В. Богдан

7. Дата выдачи задания *15.09.2023г*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала к 06.10 – 15 %;
разделы 2, 3 к 27.10 – 10 %;
разделы 4 к 06.11 –20 %;
разделы 5 к 20.11 – 35 %;
разделы 6,7,8,9 к 27.11 – 10 %;
оформление пояснительной записки и графического материала к 01.12.23 – 10 %
Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ *Богдан Е. В.*
(подпись)

Задание принял к исполнению *Мацур И.А.*
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современной среде программирования выбор языка и технологий становится ключевым для успеха проекта. В моем случае, я сосредоточился на языке программирования C++ и фреймворке Qt как мощном сочетании для разработки разнообразных приложений. Моя цель — подробнее рассмотреть, что привлекло меня в этих технологиях.

Универсальность и эффективность программирования — ключевые факторы, и C++ выделяется своей мощью и гибкостью. Этот язык поддерживает различные подходы программирования, что делает его идеальным для решения разнообразных задач — от низкоуровневых до создания сложных систем.

C++ обладает обширной стандартной библиотекой, что упрощает разработку и повышает производительность программ, предоставляя готовые решения для многих типов задач.

Этот язык активно используется в различных отраслях, таких как системное программирование, игровая индустрия, встраиваемые системы и научные исследования. Его экосистема инструментов способствует созданию масштабных и инновационных проектов.

Qt, в свою очередь, обеспечивает удобные средства для создания интерфейсов и активно участвует в развитии современных технологий, таких как Интернет вещей (IoT) и мобильная разработка. Qt Creator, интегрированная среда разработки, упрощает процесс создания, отладки и тестирования приложений, повышая эффективность и уровень удовлетворенности разработчиков.

Qt обладает обширным набором инструментов для создания современных пользовательских интерфейсов и обеспечивает кроссплатформенную совместимость, позволяя приложениям функционировать на различных операционных системах без переписывания кода.

Система сигналов и слотов в Qt обеспечивает эффективную обработку событий и взаимодействие между компонентами приложения. Богатая стандартная библиотека Qt дополняет функциональность C++, предоставляя готовые решения для различных задач, делая этот фреймворк незаменимым для разработки.

Мой выбор C++ и Qt обусловлен их выдающейся функциональностью, универсальностью и способностью эффективно работать вместе. Это делает их оптимальным выбором для реализации проектов в сфере программной инженерии. Использование этих технологий упрощает написание работы, обеспечивая гибкость, производительность и соответствие последним трендам в разработке приложений.

Кроме того, C++ является языком с открытым исходным кодом, что способствует его широкому использованию и постоянному развитию благодаря активному сообществу разработчиков. Этот язык обеспечивает высокую производительность и эффективное использование ресурсов компьютера, что особенно важно для создания масштабируемых и высоконагруженных приложений.

Qt, в свою очередь, предоставляет инструменты для разработки не только настольных приложений, но и мобильных приложений для различных платформ, включая iOS и Android. Это значительно расширяет сферу применения фреймворка и делает его востребованным для создания переносимых приложений, которые могут работать на различных устройствах.

Сочетание C++ и Qt позволяет разработчикам создавать качественное программное обеспечение, оптимизированное под конкретные потребности проекта. Возможности этих технологий применимы как для небольших приложений, так и для разработки сложных систем, где требуется высокая производительность и надежность.

Богатство инструментов и гибкость, предоставляемые C++ и Qt, существенно упрощают процесс разработки и позволяют создавать инновационные программные продукты, соответствующие современным требованиям рынка. Такое объединение сил двух мощных инструментов открывает широкие перспективы для разработчиков и позволяет создавать программное обеспечение высокого уровня качества, что делает эту комбинацию важным выбором для реализации проектов в сфере программной инженерии.

Исходя из возможностей фреймворка Qt, создание игры "Морской бой" на этой платформе можно рассматривать как относительно несложную задачу.

1 ПОСТАНОВКА ЗАДАЧИ

Цель проекта создания игры "Морской бой" заключается в разработке интерактивного приложения, которое позволит пользователям играть в классическую игру "Морской бой".

Основной задачей проекта является создание функционального и привлекательного программного продукта с использованием Qt и языка программирования C++, который предоставит пользователю возможность насладиться игрой, взаимодействуя с удобным пользовательским интерфейсом.

Это также может включать в себя создание удобного механизма размещения кораблей, игровой логики, обработки действий игроков и визуализации игровых событий. Основной целью является предоставление увлекательного и функционального игрового опыта пользователю.

Необходимо рассмотреть добавление таких деталей как расстановка кораблей, выстрелы по противнику, создать минимальный графический интерфейс для отображения окна приложения, игрового поля, кораблей и других объектов.

Дополнительно нужно рассмотреть добавление автоматической расстановки союзных кораблей, а также добавить возможность изменения уровня сложности игры.

Для реализации программы используется объектно-ориентированный язык программирования C++, библиотека Qt, среда разработки Qt Creator.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Анализ существующих аналогов

2.1.1 “KDE KNavalBattle”

Это приложение для рабочего стола KDE в Linux, предлагающее классическую версию игры с различными уровнями сложности и возможностью играть с компьютером или другим игроком. Из предложенного функционала имеется возможность участвовать в одиночной игре или же выбрать онлайн режим.

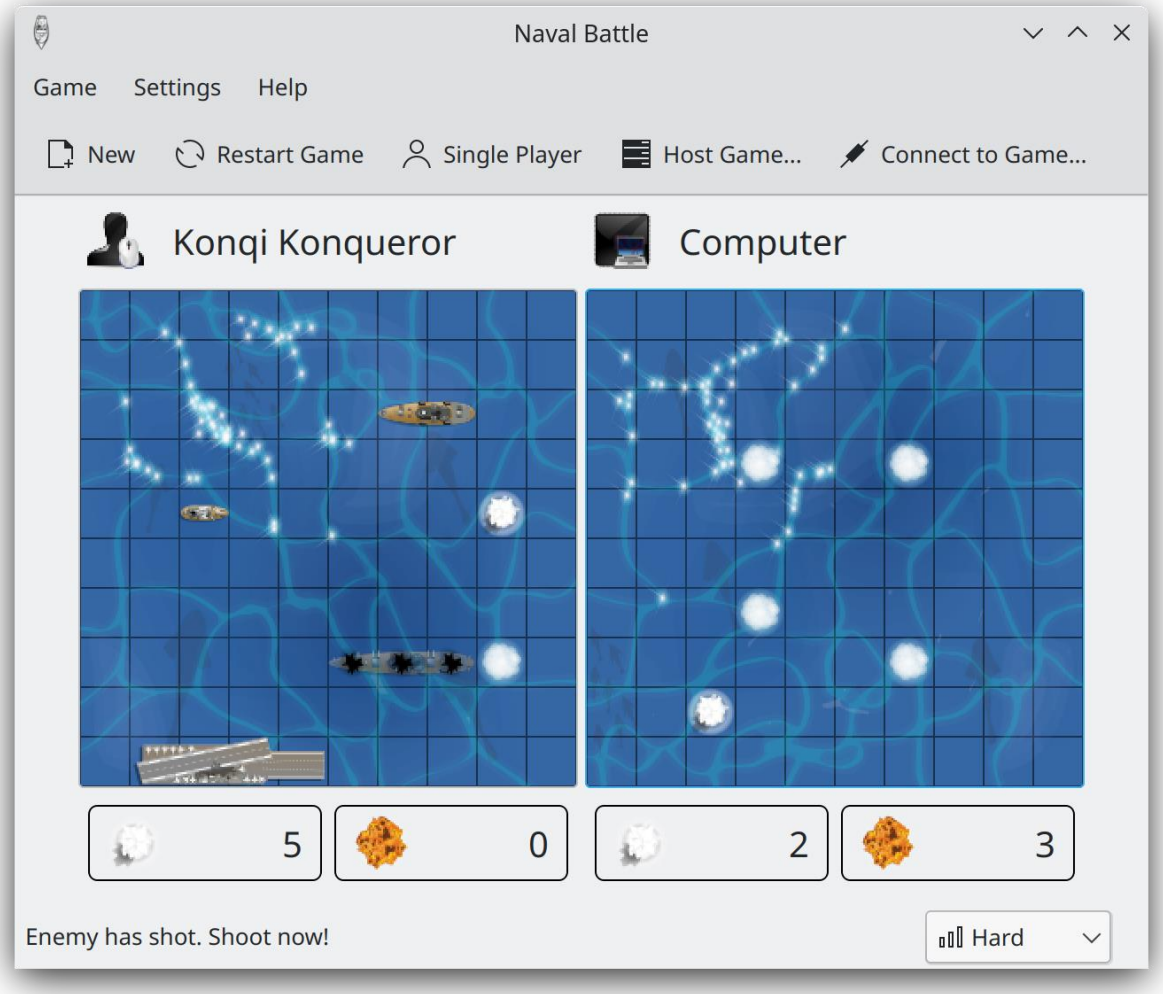

Рисунок 2.1 – Скриншот KDE KNavalBattle

2.1.2 “Sea Battle: Atomic Boom”

Мобильное приложение, которое предлагает множество разнообразных версий игры с различными режимами игры, включая одиночный режим против компьютера или мультиплеер с другими игроками. Особенностями игры являются использование различной авиатехники, позволяющая стрелять по кораблям с воздуха.

В игре также присутствует возможности заработка игровой валюты. Для её получения необходимо одержать победу в бое с противником. За счет валюты появится возможность покупки, а в дальнейшем, использовании различных военный самолетов.

Рисунок 2.2 – Заставка Sea Battle

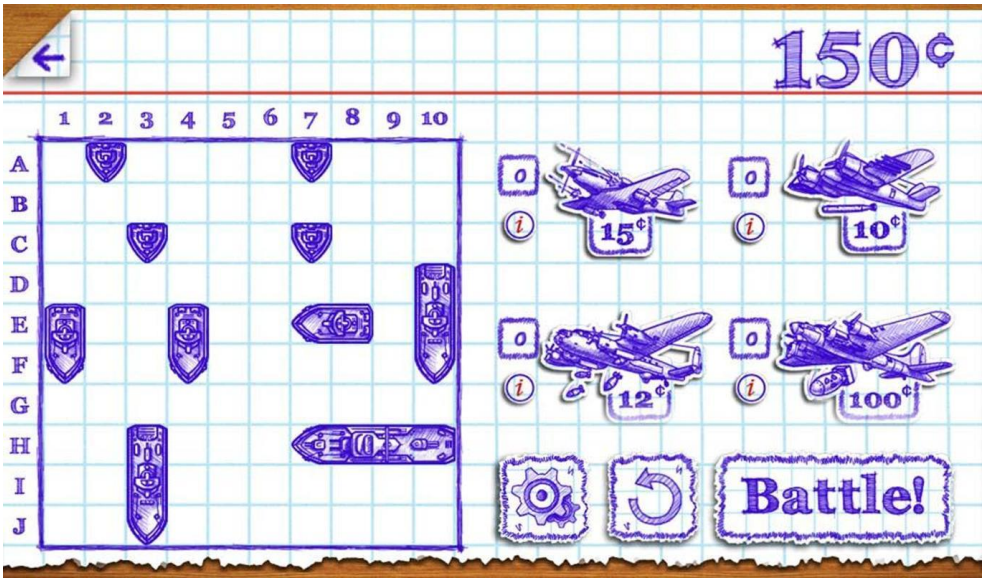

Рисунок 2.3 – Меню авиатехник

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Структура входных и выходных данных

В начале игры пользователю представится стартовое окно (класс battleShipWindow), где он должен определить следующее: начать или покинуть игру.

Перед этим выбором пользователю нужно выполнить следующие действия:

1) Установить уровень сложности игры “EASY” или же “MEDIUM”.
2) Очистить карту.
3) Выбрать автоматическую или индивидуальную расстановку кораблей. Всего необходимо расставить 10 кораблей различных типов. В бою участвует 4 однопалубных корабля, 3 двупалубных корабля, 2 трехпалубных корабля и 4 однопалубных корабля.
4) Нажать на кнопку “START GAME” для начала игры.

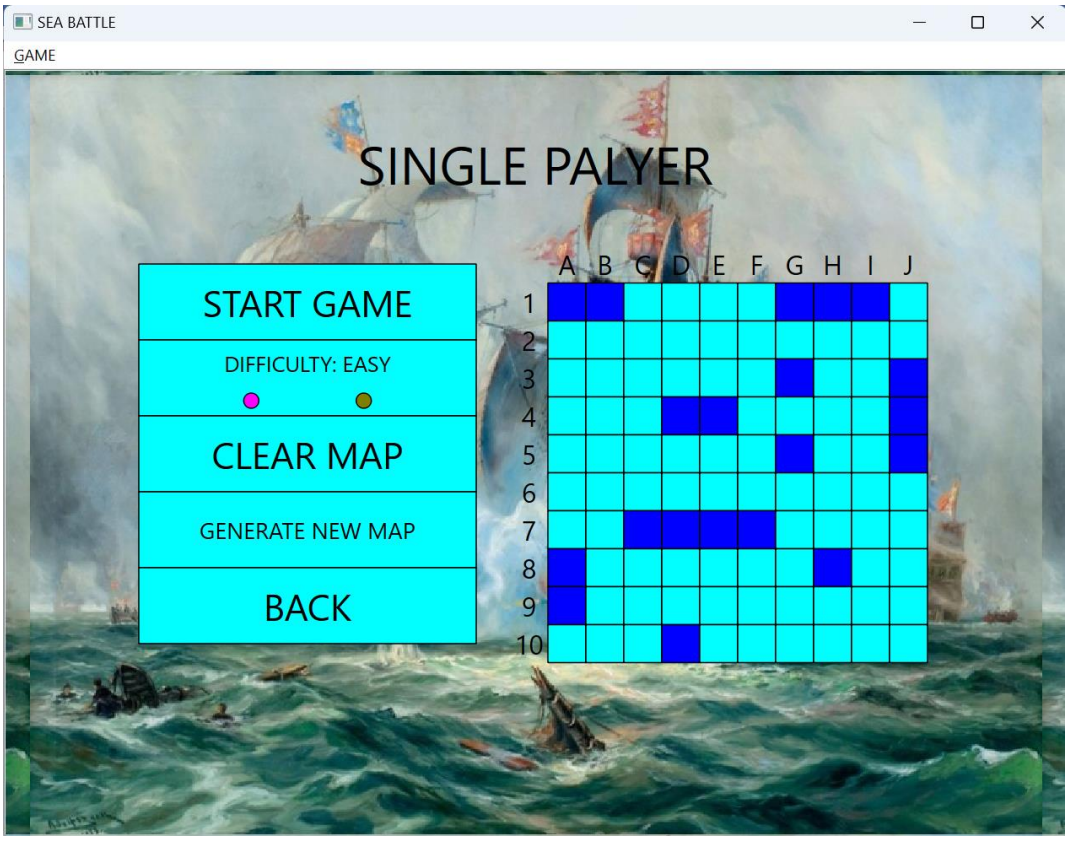

Рисунок 3.1 – Начальное окно игры

3.2 Разработка диаграммы классов

Диаграмма классов для данного курсового проекта представлена в приложении А.

3.3 Описание классов

3.3.1 Класс Ship

Описание полей класса Ship:

`int m_length` - длина корабля;

`Orientation m_orientation` – ориентация корабля;

`QVector<CellShip> m_body` - расположение корабля на карте (координаты (x, y), ориентация корабля);

Описание методов класса Ship:

`enum Orientation{Vertical, Horizontal, None}` – перечисление видов расположения корабля;

`struct CellShip{int x, int y, e_Status status}` – структура информации о ячейке поля;

`Ship(int length, Orientation orient = Orientation::None)` – конструктор;

`e_Status shot(int x, int y)` – возвращает статус выстрела по заданным координатам;

`Orientation orientation() const` – возвращает ориентацию корабля;

`void setOrientation(const Orientation &orientation)` – установка ориентации корабля;

`void setCellCoord(int numberCell, int x, int y)` – выбор расположения корабля;

`int numberCell(int x, int y) const` – возвращает номера клеток, занимаемые кораблем;

`void reset()` – сбрасывает статус корабля;

`bool isLife() const` – возвращает false, если корабль потоплен, иначе true;

`int length() const` – возвращает длину корабля;

3.3.2 Класс Player

`class Player : public QObject`

`Q_OBJECT` – макрос используется в классах QT для связывания сигналов и слотов.

Описание полей класса Player:

`QString m_playerName` – переменная для хранения имени игрока;

`QSharedPointer<GameMap> m_pGameMapPlayer` – умный указатель на карту игрока;

`QVector<QSharedPointer<Ship>> m_shipsPlayer` – вектор умных указателей на корабли игрока;

Описание методов класса Player:

`Player(QString name, QObject *parent = nullptr)` – конструктор класса `Player;`

`void setMap(QSharedPointer<GameMap> &map)` – установка карты;

`void setShips(QVector<QSharedPointer<Ship>> ships)` – установка кораблей;

`bool isDead() const` – возвращает true если игрок проиграл, иначе false;

`void reset()` – сбрасывание данных игрока;

`QString name() const` – возвращает строку, представляющую имя игрока;

`QSharedPointer<GameMap> &gameMap()` – возвращает умный указатель на карту игрока;

`QVector<QSharedPointer<Ship>> &ships()` – возвращает вектор умных указателей на корабли игрока;

`signals:`

`void mapClicked(int x, int y)` – щелчок мыши по карте игрока с координатами x, y;

3.3.3 Класс Bot

`class Bot : public Player`

Описание методов класса Bot:

`Bot(QString name, QObject *parent = nullptr)` – конструктор класса Bot;

`e_Status turn(Player *otherPlayer)` – выбор сложности бота;

`void setDifficulty(e_Difficulty difficulty)` – установка сложности игры;

`int botX() const` – возвращает координату x выстрела бота;

`int botY() const` – возвращает координату y выстрела бота;

`private:`

`e_Status easyDifficulty(Player *otherPlayer)` – реализация логики игры при легкой сложности;

`e_Status mediumDifficulty(Player *otherPlayer)` – реализация логики игры при средней сложности игры;

Описание полей класса Bot:

`bool m_expertMode` – переменная для определения режима эксперта;

`bool m_changeShotDirection` – переменная для изменения определения направления выстрела;

`int m_botX` – координата x выстрела бота;

`int m_botY` – координата y выстрела бота;

`e_Direction m_shotDirection` – переменная перечисления e_Direction для определения направления выстрела;

`e_Difficulty m_difficulty` - переменная перечисления e_Direction для определения уровня сложности бота;

3.3.4 Класс Cell

`class Cell : public QGraphicsObject`

Описание методов класса Cell:

`Cell(int width, int heigth, int idX, int idY)` – конструктор класса Cell;

`void setShowShip(bool flag)` – отображение корабля в клетке;

`void reset()` – сброс состояния клетки;

`QRectF boundingRect() const override` – виртуальный метод, возвращающий прямоугольник объекта;

`void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override` – метод разрисовки клетки;

`e_Status status() const` – метод, который возвращает статус клетки;

`void setStatus(const e_Status &status)` – метод, который устанавливает статус клетки;

`QSharedPointer<Ship> &ship()` – метод, возвращающий умный указатель на корабль, находящийся на клетке;

`void setShip(const QSharedPointer<Ship> &ship)` – установить корабль на клетке или клетках;

Описание полей класса Cell:

`e_Status m_status` – хранение статуса клетки;

`QSharedPointer<Ship> m_pShip` – представление корабля на клетке;

`int m_width` – ширина клетки;

`int m_height` – высота клетки;

`int m_idX` – номер клетки по оси X;

`int m_idY` – номер клетки по оси Y;

`bool m_hover` – логическая переменная для отслеживания наведения мыши на клетку;

`bool m_showShip` – логическая переменная для отображения корабля на клетке;

3.3.5 Класс GameMap

class GameMap : public QGraphicsObject

Описание методов класса GameMap:

GameMap(int width, int height, bool disable = false, const QString &textLayout = "ABCDEFGHIJ") – конструктор класса GameMap;

e_Status shot(int x, int y) – выстрел по указанным координатам и возвращает статус попадания;

void setCellShip(int x, int y, QSharedPointer<Ship> &ship) – установка корабля в указанные ячейки;

void setCellStatus(int x, int y, e_Status st) – установка статуса ячейки;

e_Status cellStatus(int x, int y) const – получение статуса ячейки;

bool isEmptyCell(int x, int y) const – проверка, является ли указанная ячейка пустой;

void resetStatusMesh() – сбрасывание статуса всех ячеек;

void setDestroyedArea(int x, int y) – образование уничтоженной области по координатам;

void setLabelText(const QString &textLayout) – установка текстового макета для карты;

QRectF boundingRect() const override – возвращает прямоугольник, описывающий ограничивающую рамку объекта;

void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override – отрисовка объекта;

void clicked(int x, int y) – эмитируется при клике на карту с указанием координаты x и y.

Описание полей класса GameMap:

int m_width – ширина карты;

int m_height – высота карты;

QVector<QSharedPointer<TextLabel>> m_textLayout – макет текстовых меток для размещения на карте;

QVector<QVector<QSharedPointer<Cell>>> m_mesh – информация о ячейках на карте;

3.3.6 Класс Utilities

Описание полей класса Utilities:

const int g_MAP_SIZE = 10 – константный размер поля в качестве 10 ячеек;

Описание методов класса Utilities:

`QFont computeFontForText(QRect rect, int flags, const QString &text, double scale = 0.65)` – метод для вычисления подходящего шрифта;

`int generateRandomNumber(int from, int to)` – генерация случайных чисел в диапазоне от from до to;

`enum class s_Status{Empty, NearbyShip, Life, Hit, Destroyed, Miss, Impossible}` – перечисление, определяющее различные статусы для ячеек поля;

`enum class e_Direction{Down, Up, Left, Right, None}` – перечисление, определяющее различные направления;

3.3.7 Класс BattleShipCore

```
class BattleShipCore : public QObject
Q_OBJECT
```

Описание методов класса BattleShipCore:

`BattleShipCore(QObject *parent = nullptr)` – конструктор класса;

`void setShipsFromEditorToPlayer()` – установка кораблей из редактора на игровую карту игрока;

`QVector<QSharedPointer<Ship>> standartShips()` – создание стандартных кораблей;

`bool isChange() const` – возвращает информацию, сменился ход или нет;

`void setTurnInterval(int msec)` – установка интервала хода;

`void resetGame()` – сброс игры;

`QSharedPointer<GameMap> playerHumanMap() const` – метод, возвращающий умный указатель на игровую карту человека;

`QSharedPointer<GameMap> playerBotMap() const` – метод, возвращающий умный указатель на игровую карту бота;

`QSharedPointer<GameMapEditor> gameMapEditor()` – метод, возвращающий умный указатель на редактор игровой карты;

`TurnIndicator *turnIndicator() const` – метод, возвращающий указатель на индикатор хода;

`void setRandShip(QSharedPointer<GameMap> &map, QVector<QSharedPointer<Ship>> &ships)` – установка кораблей в случайном порядке;

`bool setShip(QSharedPointer<GameMap> &map, QSharedPointer<Ship> &ship, int x, int y, bool isHor)` – установка кораблей в индивидуальном порядке;

`void endGame(QString winner);`

`bool winnerChecker()` - определение победителя;

Описание полей класса BattleShipCore:

`bool m_change` – логическая переменная, указывающая на смену хода;

`bool m_turn` – логическая переменная, хранящая информацию о текущем ходе;

`Player *m_pPlayerHuman` – указатель на объект класса Player, представляющий игрока;

`Bot *m_pPlayerBot` – указатель на объект класса Bot, представляющий бота;

`TurnIndicator *m_pTurnIndicator` – указатель на объект класса TurnIndicator, обозначающий текущий ход.

`QSharedPointer<GameMapEditor> m_pGameMapEditor` – указатель на объект класса GameMapEditor, представляющий редактор карты;

3.3.8 Класс BattleShipWindow

Описание полей класса BattleShipWindow:

`QMenu *m_pMenuGame` - указатель на меню для игры;

`QAction *m_pSettingsMenuGame` - указатель на действие меню для настроек игры;

`QAction *m_pAboutMenuGame` - указатель на действие меню "О программе";

`QAction *m_pExitMenuGame` - указатель на действие меню для выхода из игры;

`BattleShipView *m_pView` - указатель на объект класса BattleShipView, представляющий представление игры;

Описание методов класса BattleShipWindow:

`BattleShipWindow(QWidget *parent = nullptr)` - конструктор класса BattleShipWindow.

`void closeEvent(QCloseEvent *event) override` - обработка события закрытия окна;

`void changeEvent(QEvent *event) override` - изменение состояния окна;

`bool event(QEvent *event) override` - обработка общих событий;

`void aboutBattleShip()` - нажатие на действие "О программе";

`void settingWindow()` - нажатие на действие "Настройки";

3.3.9 Класс Settings:

Описание полей класса:

`QSize m_windowSize` - размер окна;

`QString m_playerName` - имя игрока;

`int m_animationDelay` - задержка анимации;

Описание методов класса Settings:

`static Settings &inst()` - получение экземпляра класса Settings;

`int animationDelay()` - получение задержки анимации;

`void setAnimationDelay(int animationDelay)` - установка задержки анимации;

`QSize windowSize() const` - получение размера окна;

`QRect sceneRect() const` - получение прямоугольника сцены;

`void setWindowSize(const QSize &windowSize)` - установка размера окна;

`QString playerName() const` - получение имени игрока;

`void setPlayerName(const QString &playerName)` - установка имени игрока;

`Settings()` - конструктор класса Settings;

`~Settings()` - деструктор класса Settings;

`Settings(const Settings &other) = delete` - запрет конструктора копирования;

`Settings& operator=(const Settings &other) = delete` - запрет оператора присваивания;

`void load()` - загрузка настроек;

`void save()` - сохранение настроек;

3.3.10 Класс SettingsWindow

Описание полей класса SettingsWindow:

`QGroupBox *m_pGroupGeneralSetting` - группа общих настроек;

`QLabel *m_pPlayerNameLabel` - метка имени игрока;

`QLineEdit *m_pPlayerNameLineEdit` - поле ввода имени игрока;

`QGroupBox *m_pGroupGameSetting` - группа настроек игры;

`QLabel *m_pAnimationSpeedLabel` - метка скорости анимации;

`QSlider *m_pAnimationSpeedSlider` - слайдер скорости анимации;

`QSpinBox *m_pAnimationSpeedSpinBox` - спинбокс скорости анимации;

`QPushButton *m_pButtonOk` - кнопка "ОК";

`QPushButton *m_pButtonApply` - кнопка "Применить";

`QPushButton *m_pButtonCancel` - кнопка "Отмена";

Описание методов класса SettingsWindow:

`SettingsWindow(QWidget *parent = nullptr)` - конструктор класса;

`void changeEvent(QEvent *event) override` - переопределенный метод события изменения;

`void clickedButtonOk()` - слот для обработки нажатия кнопки "ОК";

`void clickedButtonApply()` - слот для обработки нажатия кнопки "Применить";

`void clickedButtonCancel()` - слот для обработки нажатия кнопки "Отмена".

3.3.11 Класс TurnIndicator

Описание полей класса TurnIndicator:

`QVector<QPointF> m_points` - вектор точек;

`int m_width` - ширина индикатора;

`int m_height` - высота индикатора;

`qreal m_angle` - угол поворота индикатора;

`QColor m_color` - цвет индикатора;

QPropertyAnimation m_colorAnimation - анимация изменения цвета.

Описание методов класса TurnIndicator:

`TurnIndicator(int width, int height)` - конструктор класса;

`void setAnimationDelay(int msec)` - установка задержки анимации;

`void reset()` - сброс индикатора;

`QColor color() const` - получение текущего цвета;

`QRectF boundingRect() const override` - метод получения ограничивающего прямоугольника;

`void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override` - рисование;

public slots: - слоты класса;

`void change(double angle, QColor color)` - изменение угла поворота и цвета индикатора;

`void setColor(QColor color)` - установка цвета;

`void colorChanged(QColor color)` - сигнал изменения цвета.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка схем алгоритмов

Метод winnerChecker() возвращает true или false в зависимости от победы или поражения игрока. Схема метода winnerChecker() показана в приложении Б.

Метод shot(int x, int y) принимает значение координаты ячейки, затем обрабатывает информацию о ней. Возвращает состояние ячейки после выстрела. Схема метода shot(int x, int y) показана в приложении В.

4.2 Разработка алгоритмов

4.2.1 Алгоритм по шагам для формирования разрушенной области

Этот алгоритм проверяет соседние ячейки вокруг заданных координат и изменяет их статусы в зависимости от текущего статуса ячейки и заданного направления. Он продолжает этот процесс до тех пор, пока не найдет область с разрушенными ячейками, после чего завершает свое выполнение.

1. Создание логической переменной find, начальное значение которой false. Также используем константную переменную g_MAP_SIZE = 10.
2. Начинается цикл do-while, который будет выполняться, пока значение false будет оставаться true.
3. Внутри цикла do-while происходит двойной цикл for, перебирающий комбинации i и j от -1 до 1 включительно. Проверяется, находится ли ячейка (x + i, y + j) в пределах игровой карты. Если да, то проверяем статус ячейки.
4. Если статус ячейки == e_Status::NearbyShip, то переходим в пункт 5, иначе если статус ячейки == e_Status::Hit, то переходим в пункт 6.
5. Статус ячейки изменяется на e_Status::Miss.
6. Меняем статус ячейки на e_Status::Destroyed.
7. Выбирается соответствующее направление (dir), и координаты (x, y) изменяются соответственно. Если dir == e_Direction::Right, x увеличивается на 1, если dir == e_Direction::Left, x уменьшается на 1, если dir == e_Direction::Up, y уменьшается на 1, если dir == e_Direction::Down, y увеличивается на 1.
8. Проверяем новые координаты. Если новые координаты (x, y) выходят за границы карты (x >= g_MAP_SIZE, y >= g_MAP_SIZE, x < 0, y < 0), цикл продолжается снова с начала, иначе переходим в п. 9.

9. Если статус ячейки по новым координатам (x, y) равен e_Status::Destroyed, устанавливается find в true.

10. Цикл do-while продолжает выполняться, пока find остается true, иначе цикл заканчивается.

4.2.2 Алгоритм по шагам для обработки хода

Этот алгоритм реализует последовательность действий для обработки хода человеческого игрока и взаимодействия с ботом в игре "Морской бой".

1. Проверяем, разрешен ли ход игроку (m_turn == true). Если не разрешен (m_turn == false), выходим из функции.
2. Если разрешен, то устанавливаем m_turn == true, чтобы запретить дополнительные ходы до завершения текущего хода.
3. Устанавливаем флаг m_change == true, что указывает на изменение состояния игры.
4. Вызываем метод shot() игровой карты бота (m_pPlayerBot -> gameMap()), чтобы выстрелить в указанную точку с координатами (x, y).
5. Если клетка недоступна (e_Status::Impossible) или есть попадание (e_Status::Hit), то m_turn == false и выходим из функции.
6. Если корабль уничтожен (e_Status::Destroyed), то вызываем метод setDestroyArea на карте бота для обозначения уничтоженной области. Затем проверяем победителя winnerChecker().
7. Если не выполняются пункты 5 и 6, то продолжаем выполнение программы.
8. Запускаем бесконечный цикл while (true), который обрабатывает ход бота. Вызываем метод turn для передачи хода боту, передавая методу объект игровой карты игрока.
9. Аналогично выполняем действия пунктов 4, 5, 6. Если не выполняется ни один пункт, то продолжаем выполнение программы, передавая ход игроку.

5 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

На рисунке 5.1 изображено начальное окно приложения. Окно имеет элементы управления программой. Кнопка “Single Player” отвечает за переход к следующему этапу с расстановкой кораблей, а также выбора сложности. Кнопка “Exit” отвечает за выход из программы. Помимо этой кнопки можно использовать сочетание клавиш “Ctrl + Q” для выхода из программы.

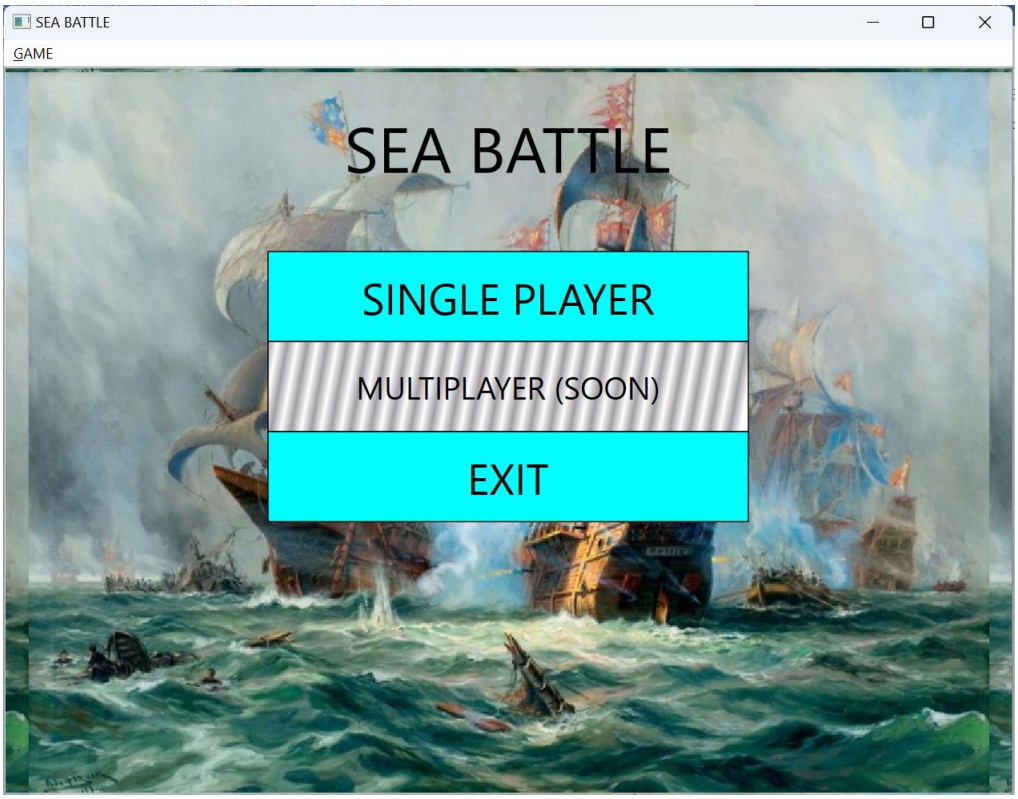

Рисунок 5.1 – Начальное окно приложения

На рисунке 5.2 представлено главное меню, представляющий собой основную часть приложения, отображающую доступные изменения полей и кнопок. В распоряжении игрока присутствуют различные кнопки, которые отвечают за свои действия.

Кнопка “Start Game” – начало игры. Нажатие данной кнопки разрешается только при полной расстановке кораблей и выборе уровня сложности.

Кнопка “Difficulty” – сложность игры. На данном этапе разработки приложения представлены 2 различные сложности игры: “Easy” и “Medium”. Различие заключается только в том, как бот будет стрелять по карте игрока.

Кнопка “Clear Map” – очистка карты. При не желаемом расположении боевых кораблей, пользователь может нажатием кнопки легко сбросить расположение кораблей. Однако придется заново располагать корабли.

Кнопка “Generate New Map” – случайная генерация расположения кораблей на карте. При нажатии корабли автоматически будут расставлены на поле, следуя всем правилам игры “Морской бой”.

Кнопка “Back” – вернуться к начальному окну. Кнопка отвечает за возврат в начальное окно.

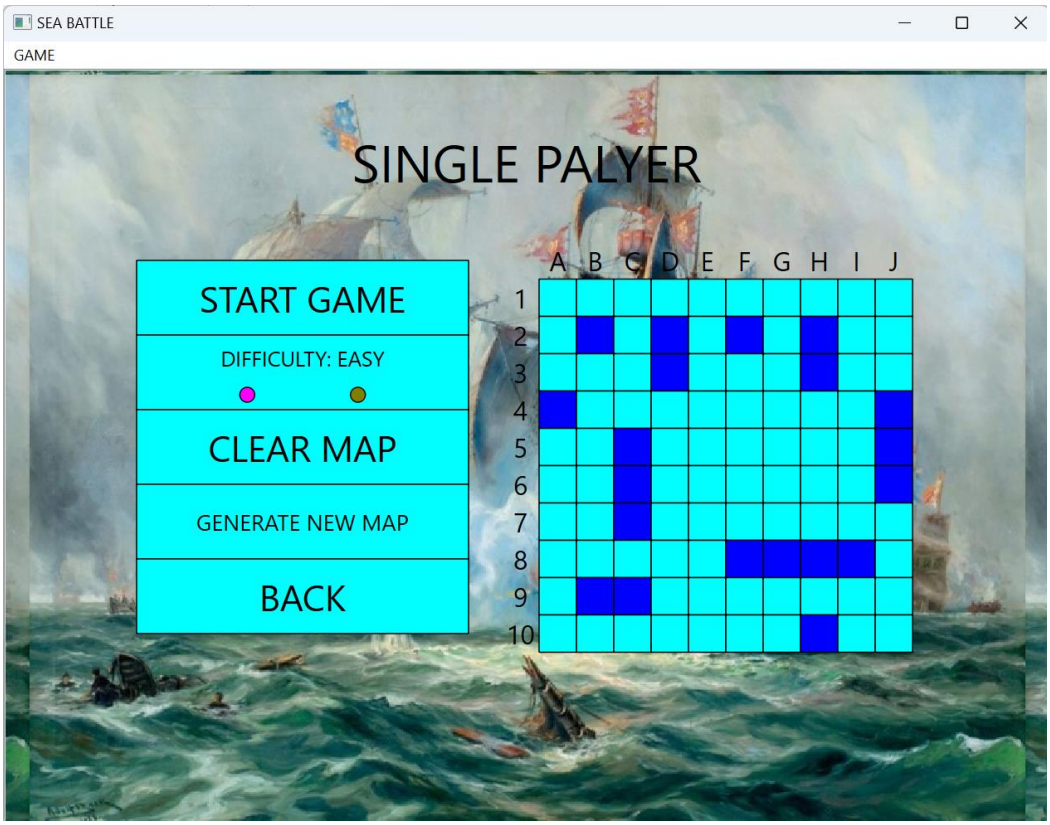

Рисунок 5.2 – Главное окно приложения

На рисунке 5.3 изображено то же главное окно приложения. Однако для получения такого окна необходимо нажать на кнопку “Clear Map”. После чего открывается возможность расстановки кораблей.

Для комфортного использования расстановка кораблей происходит путем перетаскивания их на поле битвы. Для изменения ориентации кораблей необходимо выполнить двойное нажатие на выбранный корабль.

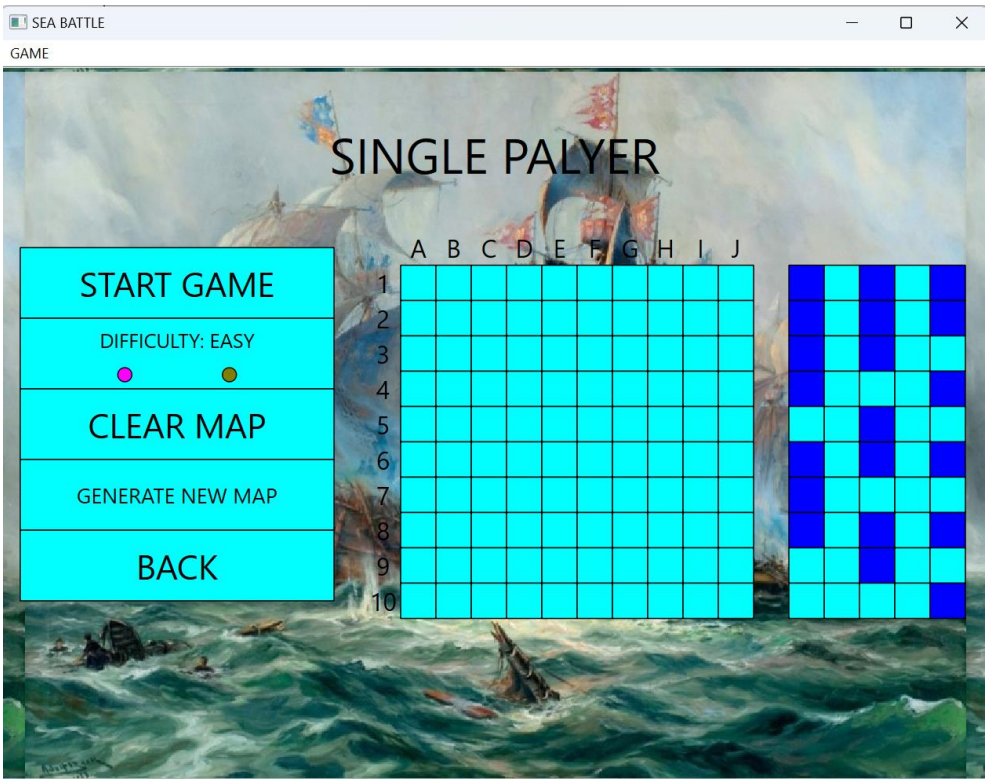

Рисунок 5.3 – Окно с возможностью расстановки кораблей

На рисунке 5.4 представлен конец программы. В открывшемся окне представлена информация о победе или поражении, а также имени пользователя, которое он указал в настройках игры.

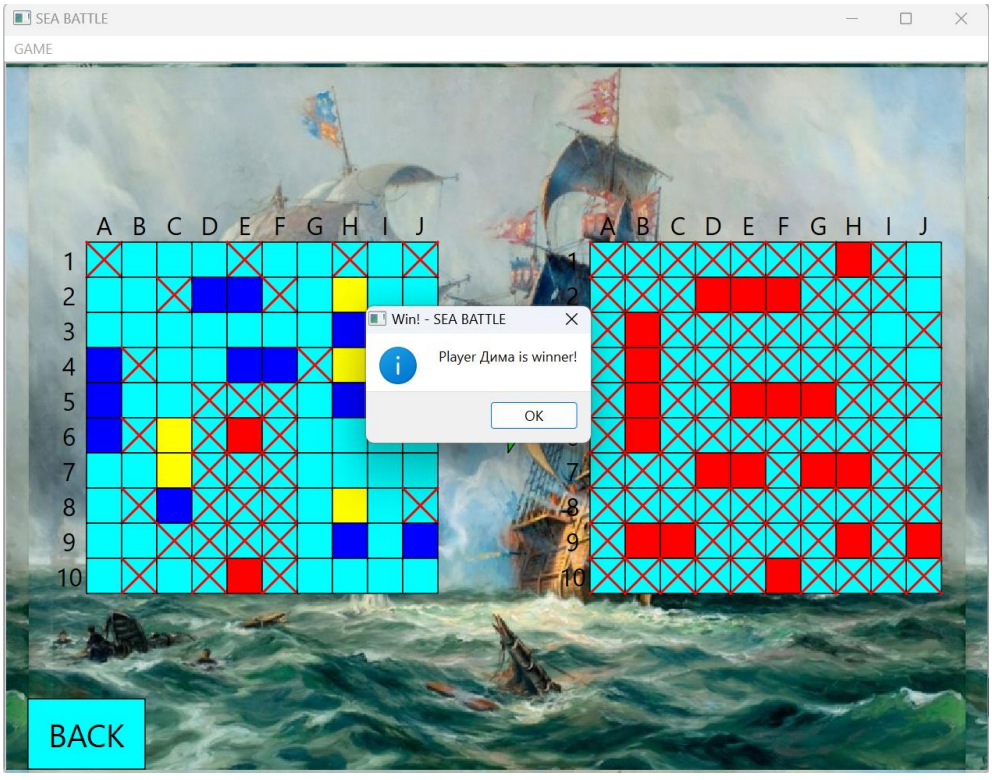

Рисунок 5.4 – Окно с концовкой игры

ЗАКЛЮЧЕНИЕ

В врамках данной курсовой работы был рассмотрен и реализован классический игровой проект "Морской бой" с использованием языка программирования C++ и библиотеки Qt. Целью работы было создание интерактивной и функциональной версии игры, позволяющей пользователю насладиться игровым процессом и улучшить навыки в разработке программного обеспечения.

Основной функционал игры был реализован с применением объектно-ориентированного подхода, что способствовало удобству и гибкости в управлении игровым процессом. В процессе разработки были использованы основные принципы программирования, такие как создание классов, работа с массивами, управление потоками выполнения, обработка событий пользовательского ввода и визуализация с помощью графического интерфейса Qt.

Благодаря применению библиотеки Qt удалось создать интуитивно понятный и привлекательный пользовательский интерфейс. Игровое поле, возможность установки кораблей и процесс атаки были реализованы в соответствии с классическими правилами "Морского боя", что позволяет игроку полностью погрузиться в игровой процесс и насладиться стратегическими аспектами игры.

Однако, существует потенциал для дальнейшего улучшения проекта. Дополнительные функции, такие как реализация различных уровней сложности, расширенные анимации, оптимизация алгоритмов для поиска кораблей и улучшение графического интерфейса могут быть внедрены для улучшения игрового опыта пользователей.

В заключение, разработка игры "Морской бой" на языке программирования C++ с использованием библиотеки Qt была успешно завершена. Полученный опыт в разработке программного обеспечения позволил углубить знания по ООП и применению графических интерфейсов для создания интерактивных приложений.

Этот проект может служить основой для дальнейших исследований и улучшений в области разработки игровых приложений, а также представляет собой пример эффективной работы с языком программирования C++ и библиотекой Qt для создания интересных и функциональных программных продуктов.

СПИСОК ЛИТЕРАТУРЫ

[1] "Объектно-ориентированное программирование на C++" Бьярн Страуструп

[2] "Язык программирования C++" Герберт Шилдт

[3] "C++ Primer" Липман, Лажойе, Му, Хопкинс

[4] "Алгоритмы. Построение и анализ" Кормен, Лейзерсон, Ривест, Штайн

[5] "Введение в алгоритмы" Кормен, Лейзерсон, Ривест, Штайн

[6] "Алгоритмы на C++" Роберт Седжвик, Кевин Уэйн

[7] C++: эффективное программирование. 55 способов улучшения структуры программ и стиля кода" Scott Meyers

[8] "Алгоритмы. Построение и анализ" Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

[9] "Структуры данных и алгоритмы в C++" Robert Lafore

[10] "C++ for Game Programmers" Noel Llopis

ПРИЛОЖЕНИЕ А

(Обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б

(Обязательное)

Блок схема алгоритма проверки победителя

ПРИЛОЖЕНИЕ В

(Обязательное)

Блок схема алгоритма логики выстрела

ПРИЛОЖЕНИЕ Г
(Обязательное)
Листинг кода

ПРИЛОЖЕНИЕ Д

(Обязательное)

Ведомость документов