1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. Кафедра «Высшая школа кибербезопасности»

**КУРСОВАЯ РАБОТА**

1. «Игра Морской бой»
2. по дисциплине «Структуры данных»
3. Выполнили
4. студенты гр. 5131001/40004 Темирханов Т.А.

Ткаченко В.М.

<*подпись*>

1. Преподаватель
2. ст. преподаватель Вагисаров В.Б.

<*подпись*>

1. «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Санкт-Петербург

1. 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc201241955)

[1. РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И МЕХАНИК ИГРЫ 4](#_Toc201241956)

[1.1 ПРАВИЛА ИГРЫ 4](#_Toc201241957)

[1.2 ПОСТАНОВКА ЗАДАЧ 5](#_Toc201241958)

[2. АРХИТЕКТУРА И ЛОГИКА РАЗРАБОТАННОЙ ПРОГРАММЫ 6](#_Toc201241959)

[2.1 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ ДЛЯ ПРЕДСТАВЛЕНИЯ ИГРОВЫХ СУЩНОСТЕЙ 6](#_Toc201241960)

[2.2 ОСНОВНЫЕ АЛГОРИТМЫ ИГРОВОГО ПРОЦЕССА 7](#_Toc201241961)

[2.3 ОСНОВЫ РАБОТЫ С OPENGL И FREEGLUT 8](#_Toc201241962)

[2.4 СИСТЕМА КООРДИНАТ И ПРИМИТИВЫ В OPENGL 9](#_Toc201241963)

[2.5 ТЕКСТУРЫ И РЕНДЕРИНГ ТЕКСТА 12](#_Toc201241964)

[3. РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И МЕХАНИК ИГРЫ 13](#_Toc201241965)

[3.1 ФУНКЦИОНАЛ ПРОГРАММЫ 13](#_Toc201241966)

[3.2 РЕАЛИЗАЦИЯ ОТОБРАЖЕНИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА 18](#_Toc201241967)

[3.3 СИСТЕМА УПРАВЛЕНИЯ 24](#_Toc201241968)

[3.4 РЕАЛИЗАЦИЯ ГЕЙМПЛЕЯ И ИГРОВОЙ ЛОГИКИ 25](#_Toc201241969)

[3.5 ИСПОЛЬЗОВАННЫЕ БИБЛИОТЕКИ 30](#_Toc201241970)

[ЗАКЛЮЧЕНИЕ 31](#_Toc201241971)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 32](#_Toc201241972)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 33](#_Toc201241973)

[ПРИЛОЖЕНИЕ Б. УСТАНОВКА ГРАФИЧЕСКИХ БИБЛИОТЕК И ЗАПУСК ПРОЕКТА 56](#_Toc201241974)

ВВЕДЕНИЕ

В мире компьютерных игр классические стратегии до сих пор остаются популярными, несмотря на развитие технологий. Одной из таких классических игр является “Морской бой” – простая, но увлекательная стратегия. Данная курсовая работа посвящена реализации “Морского боя” с использованием языка программирования С и графических библиотек.

**Цель работы**

Разработать графическую версию игры “Морской бой” на языке C с использованием библиотек OpenGL и FreeGLUT для создания окон, отрисовки графического интерфейса и обработки ввода. Реализация подразумевает 3 этапа игры (расстановка, бой и конец игры), игра рассчитана на 1 игрока (пользователь) и противника (компьютера).

**Актуальность темы**

Актуальность данной темы обусловлена в первую очередь своей образовательной ценностью – она помогает в освоении и совершенствовании навыков программирования, таких как обработка ввода, работа с графикой, проектирование архитектуры кода и отладки. Также данная работа позволит частично изучить основы разработки игр.

**Задачи работы**

1. Изучить основы работы с библиотеками OpenGL и FreeGLUT.
2. Представить игровые объекты с помощью различных структур данных
3. Реализовать основной игровой процесс и завершение игры
4. Реализовать логику игрока-компьютера, в соответствии с правилами игры
5. Создать графический интерфейс с меню и игровыми полями.
6. Оптимизировать код и логику компьютера.

1. РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА   
И МЕХАНИК ИГРЫ

1.1 ПРАВИЛА ИГРЫ

"Морской бой" — пошаговая стратегическая игра, где два участника (игрок и компьютер) играют на полях 10×10 клеток, по очереди атакуя корабли противника на скрытом игровом поле и пытаясь уничтожить все корабли противника.

Задача игрока – первым уничтожить все корабли противника.

Игровое поле представляет собой сетку 10×10 клеток, где каждая клетка имеет размер 50×50 пикселей.  По краям полей добавлены координатные оси с обозначениями от A до J по горизонтали и от 1 до 10 по вертикали.

Игра делится на две фазы: фаза расстановки кораблей и фаза игры.

В первой фазе игрок расставляет свои корабли на поле, обладая следующим флотом из 10-ти кораблей: линкор (1 корабль на 4 клетки), крейсера (2 корабля по 3 клетки), эсминцы (3 корабля по 2 клетки) и катера (4 корабля по 1 клетке). При размещении корабли не должны соприкасаться друг с другом ни сторонами, ни углами - между ними обязательно должна быть хотя бы одна свободная клетка. Корабли можно располагать как горизонтально, так и вертикально.

Управление осуществляется при помощи курсора и клавиатуры. Используемые клавиши клавиатуры – R, Q, N;   
R – поворот корабля при расстановке, Q – выход из игры, N – начало новой игры. Клавиши Q и N могут быть использованы в любой момент игры.

После расстановки кораблей начинается основной этап игры. Игроки по очереди называют координаты клетки на поле противника. В нашей реализации игрок делает выстрел, кликая мышкой по клетке правого поля (поля бота), в то время как компьютер автоматически рассчитывает свой ход. При попадании в корабль противника атакующий получает право на дополнительный выстрел, а при промахе ход переходит оппоненту. Корабль считается уничтоженным, когда поражены все занимаемые им клетки. При этом все соседние с уничтоженным кораблем клетки автоматически отмечаются как промахи, так как по правилам игры в них не могут находиться другие корабли. В нижней части экрана отображается информация о количестве оставшихся кораблей каждого типа у обоих игроков.

Игра заканчивается, как только будут уничтожены все корабли одного из противников.

1.2 ПОСТАНОВКА ЗАДАЧ

Разработка игры "Морской бой" потребовала решения нескольких взаимосвязанных задач:

* 1. **Программная реализация на языке C**: игра должна быть написана на языке C с использованием стандартных библиотек.
  2. **Использование графических библиотек**: для отрисовки графики используется OpenGL, а точнее библиотека FreeGLUT, содержащая тот же функционал, что и предшественник-библиотека GLUT, которая уже не поддерживается.
  3. **Пользовательский интерфейс**: игра должна содержать стартовое меню, экран расстановки кораблей, экран основного игрового процесса и экран завершения игры.
  4. **Логика бота:** один из игроков является компьютером, для которого было необходимо реализовать игровую логику поведения.

2. АРХИТЕКТУРА И ЛОГИКА РАЗРАБОТАННОЙ ПРОГРАММЫ

2.1 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ ДЛЯ ПРЕДСТАВЛЕНИЯ ИГРОВЫХ СУЩНОСТЕЙ

Для представления объектов в игре используются следующие структуры:

1. **Состояние игры:** структура типа enum описывает состояние игры – главное меню, фаза расстановки, основной игровой процесс, финальный этап.

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1 – Структура состояния игры

1. **Клетка**: хранит информацию о присутствии корабля, фиксирует попадание и промах.

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 – Структура клетки

1. **Корабль:** информация о типах кораблей и их количестве.

Изображение выглядит как Шрифт, текст, рукописный текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3 – Структура корабля

1. **Координаты:** пара координат, используется для хранения положений клеток.

Изображение выглядит как Шрифт, текст, снимок экрана, Графика

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4 – Структура координат

1. **Выстрел противника:** анализ попаданий и расчет дальнейших действий бота. Сохраняет координаты попаданий, количество попаданий по одному кораблю, определяет 4 возможных направления для дальнейших выстрелов и хранит координаты последнего произведенного выстрела.

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 5 – Структура выстрела противника

2.2 ОСНОВНЫЕ АЛГОРИТМЫ ИГРОВОГО ПРОЦЕССА

Основные алгоритмы:

* **Инициализация игры и размещение на поле**: создается 2 игровых поля (игрока и компьютера). Задаются типы кораблей, игрок выбирает позицию каждого корабля и его ориентацию, параллельно проверяется валидность каждой позиции.
* **Выстрелы**: выстрел игрока, проверка попадания, если попадание засчитано – проверка уничтожения корабля. Затем – выстрел бота по собственному алгоритму.
* **Завершение игры**: проверка уничтожения всех кораблей после каждого хода. Вывод экрана завершения игры.

Логика работы бота:

* **Размещение кораблей:** координаты расстановки и направление выбирается случайное, затем проверяется возможность размещения корабля на полученной позиции. Если размещение невозможно – попытка повторяется.
* **Стрельба:** когда у бота нет информации о возможном расположении кораблей противника, он выбирает случайную клетку на поле игрока, проверяет, не стрелял ли он уже в эту клетку ранее (иначе выбирает другую), и совершает выстрел в выбранную клетку. Если попадание было успешным, то он запоминает его координаты и начинает поиск остальных клеток этого корабля, проверяя соседние 4 клетки и обстреливая их, чтобы найти направление положения этого корабля. При успешном нахождении продолжает обстрел в том же направлении, пока не достигнет конца корабля. При ошибочном направлении возвращается к исходной точке и проверяет противоположное направление. Если корабль уничтожен – отмечает все клетки вокруг как промахи и возвращается к случайным выстрелам.
* **Оптимизация выстрелов**: бот не стреляет в одну и ту же клетку дважды и приоритезирует клетки, где по его расчетам с наибольшей вероятностью может находиться корабль.

2.3 ОСНОВЫ РАБОТЫ С OPENGL И FREEGLUT

**OpenGL (Open Graphics Library)** — является одним из самых популярных программных интерфейсов (API) для разработки приложений в области двумерной и трехмерной графики. Стандарт OpenGL был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения, а его основой стала библиотека IRIS GL, разработанная Silicon Graphics. На данный момент реализация OpenGL включает в себя несколько библиотек (описание базовых функций OpenGL, GLU, GLUT,GLAUX и другие). [1]

**FreeGLUT (Free OpenGL Utility Toolkit)** — это открытая альтернатива библиотеке **GLUT** (OpenGL Utility Toolkit), которая предоставляет простой API для создания окон, обработки ввода (клавиатура, мышь) и управления OpenGL-рендерингом. [2].

Основные этапы работы с OpenGL и FreeGLUT:

* 1. Инициализация OpenGL и FreeGLUT и создание окна.
  2. Настройка графического контекста OpenGL.
  3. Задания ортографической 2D проекции
  4. Основной цикл: регистрация callback-функций и главный цикл FreeGLUT – бесконечный цикл обработки событий (обновление состояния и отрисовка).

2.4 СИСТЕМА КООРДИНАТ И ПРИМИТИВЫ В OPENGL

В программе реализована **двумерная декартова система координат** с использованием ортографической проекции OpenGL. Настройка проекции выполняется функцией gluOrtho2D():



Рисунок 4 – Шаблон ортографической проекции

В нашем случае:



Рисунок 5 – Реализация ортографической проекции

Это означает, что левый верхний угол окна соответствует координатам (0, 0). Правый нижний — (WINDOW\_WIDTH, WINDOW\_HEIGHT) (1200×650 пикселей). Ось Y направлена вниз для соответствия стандартному представлению растровых изображений

Основным графическим элементом (примитивом) в нашей работе является квадрат (GL\_QUDS), которые используется для ячеек игрового поля (в нашем случае, размер клетки CELL\_SIZE = 50 пикселей) и для отображения кораблей и маркеров попаданий. В OpenGL примитивы задаются между вызовами функции glBegin() и glEnd() [4]. Для отрисовки квадрата используется следующий шаблон:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 6 – Шаблон отрисовки примитива (в нашем случае - квадрата) в OpenGL

В нашем случае:

Изображение выглядит как текст, Шрифт, снимок экрана, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 7 – Реализация примитивов OpenGL

Для визуализации состояний клеток применяется RGB-модель через функцию glColor3f():



Рисунок 8 – Шаблон задания цвета через RGB-модель в OpenGL

Параметры (r, g, b) задают интенсивность красного, зелёного и синего каналов в диапазоне 0.0-1.0.

В нашем коде применяются следующие параметры для визуализации клеток:

* (0.3, 0.3, 0.3) — неподбитые корабли (тёмно-серый)
* (1.0, 0.0, 0.0) — уничтоженные корабли (ярко-красный)
* (0.8, 0.8, 1.0) — пустые клетки (светло-голубой)

Фундаментальным элементом интерфейса в программе является графическая сетка, обеспечивающая визуальное структурирование игрового поля. Графическая сетка отрисовывается с помощью функции drawGrid:

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 9 – Отрисовка графической сетки с помощью функции drawGrid

2.5 ТЕКСТУРЫ И РЕНДЕРИНГ ТЕКСТА

Вывод текста: для отображения текстовой информации в программе используется встроенный механизм GLUT, основанный на растровых шрифтах.

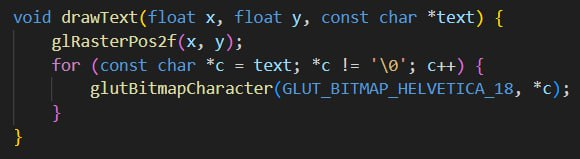


Рисунок 10 – Базовая функция отрисовки текста

Используется шрифт Helvetica 18pt (GLUT\_BITMAP\_HELVETICA\_18), поддерживаются только ASCII-символы фиксированного размера. Позиционирование осуществляется в оконных координатах.

Отображение динамических элементов интерфейса: в нашем случае – информации о количестве оставшихся кораблей игрока.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 11 – Отображение динамических элементов

В данном случае используется буферизированное форматирование (snprintf).

3. РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И МЕХАНИК ИГРЫ

3.1 ФУНКЦИОНАЛ ПРОГРАММЫ

Игра разделена на условные 4 фазы:

* Стартовое меню: инструкции управления (поворот на клавишу R, размещение кораблей и выстрел – ЛКМ, начало новой игры на клавишу N, выход из игры на клавишу Q).

Изображение выглядит как текст, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 12 – Стартовое меню

* После нажатия любой клавиши клавиатуры начинается основной игровой процесс. Сначала идет этап расстановки кораблей игрока:

Изображение выглядит как текст, снимок экрана, линия, График

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 13 – Поле расстановки кораблей игрока

На клавишу R осуществляется поворот корабля, на ЛКМ – утверждение расстановки. При попытке расположить каждый корабль также выполняется проверка доступности текущего расположения, в случае невозможности размещения (корабли пересекаются или расстояние между ними менее одной клетки) попытка блокируется и выводится предупреждение “Invalid position”.

Изображение выглядит как текст, снимок экрана, прямоугольный, Прямоугольник

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 14 – Попытка размещения крейсера в H6

* После расстановки всех кораблей игрока выводится скрытое поле противника с расстановленными в случайном порядке кораблями. Начинается фаза атаки

Изображение выглядит как текст, прямоугольный, снимок экрана, Прямоугольник

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 15 – Начало игры

Голубым цветом обозначены нетронутые клетки, светло-серым – промахи, оранжевым – попадания, красным – клетки уничтоженных кораблей. По классическим правилам игрок, успешно подбивший корабль противника, получает дополнительный выстрел. При уничтожении корабль отмечается красными клетками, а клетки вокруг автоматически отмечаются как промахи (так как в них корабли располагаться уже не могут). Осуществить выстрел дважды в одну клетку игрок не может.

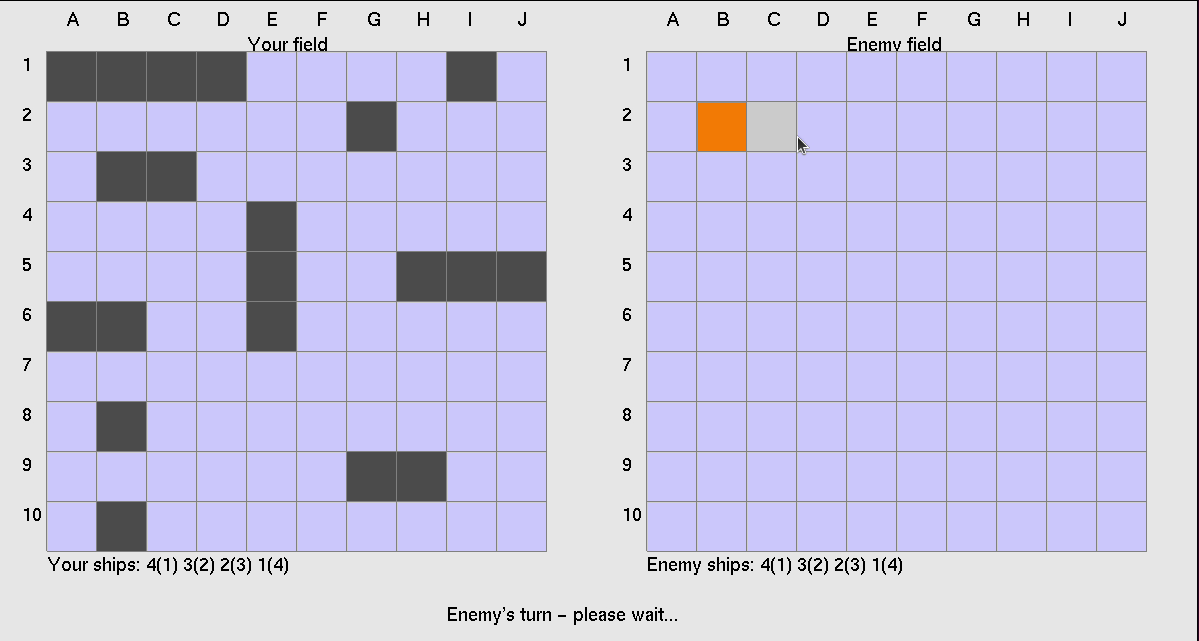


Рисунок 16 – Подбитый корабль противника и отмеченные промахи

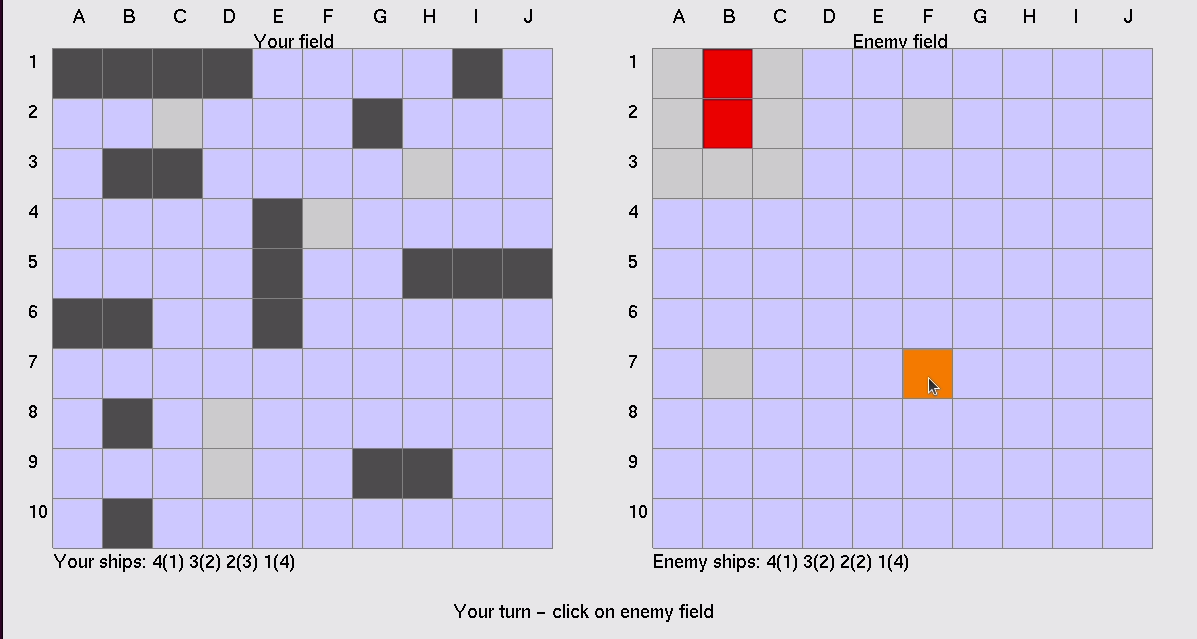


Рисунок 17 – Уничтоженный корабль противника и отмеченные промахи вокруг

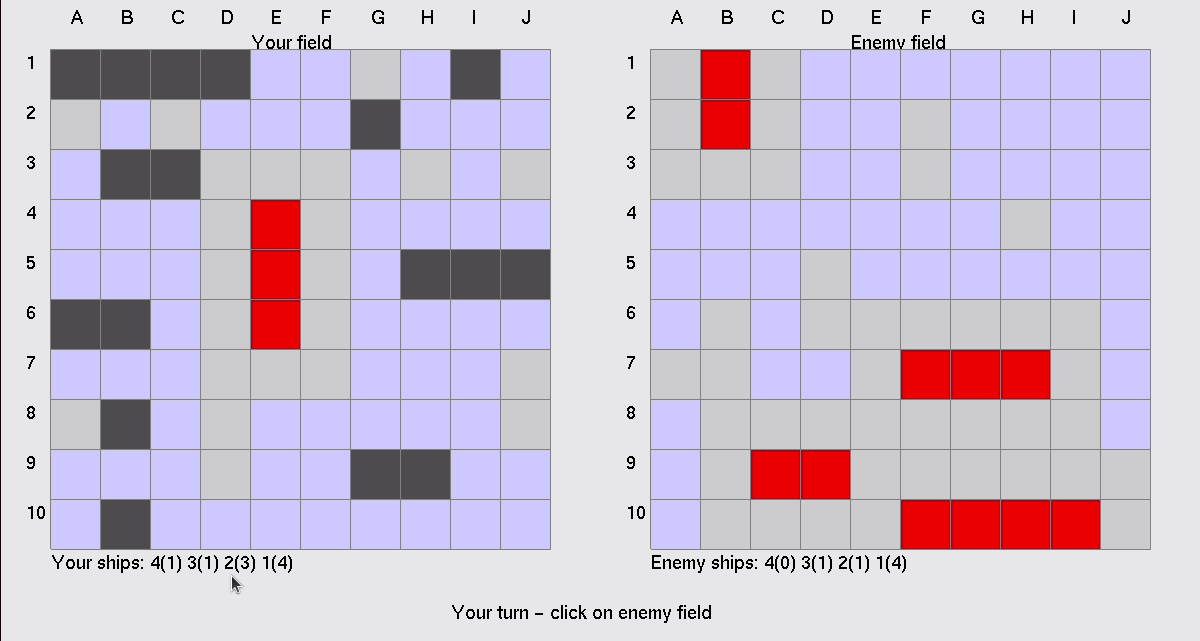


Рисунок 18 – Уничтоженный корабль игрока

Внизу каждого поля отмечены счетчики оставшихся кораблей игрока и противника в формате “размер\_корабля(количество\_кораблей)”.

* После уничтожения всех кораблей одного из оппонентов игра заканчивается – показывается экран конца игры и мини-карты итоговых полей.



Рисунок 19 – Экран конца игры (победа бота)

3.2 РЕАЛИЗАЦИЯ ОТОБРАЖЕНИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА

**Стартовое меню.** Отрисовка меню происходит в функции renderMenu(). Она сперва очищает экран перед отрисовкой объектов и задает цвет фона (светло-серый) через функцию glClear(). Затем происходит отрисовка текста “SEA BATTLE” (синим цветом) и инструкций управления (черным цветом) при помощи функции drawText().

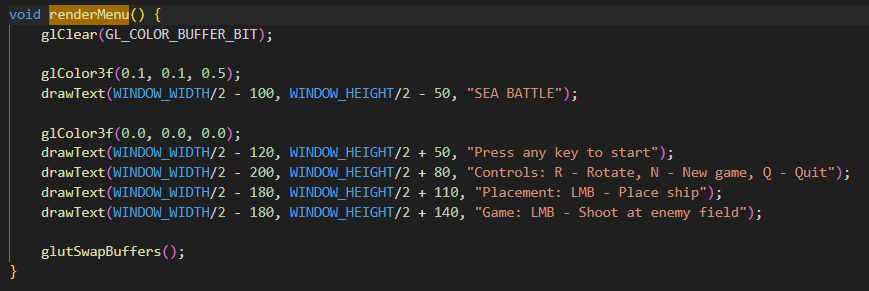


Рисунок 20 – Отрисовка меню (функция renderMenu())

Функция отрисовки текста (drawText()) выводит текст, задаваемый 3-им параметром по переданным координатам с позиции старта, задаваемой glRasterPos2f(). Цикл проходит по каждому символу строки, отрисовывая каждый по отдельности, используя встроенный растровый шрифт размера 18 пикселей (GLUT\_BITMAP\_HELVETICA\_18), и затем автоматически смещая позицию вывода следующего символа.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 21 – Отрисовка текста (функция drawText())

**Экран расстановки кораблей.** Отрисовка данного экрана происходит в функции renderPlacement(). Сначала происходит заливка фона, затем отрисовка текста обозначения поля игрока. Само поле отрисовывается через функцию drawGrid() с параметром showShips=true. Она отрисовывает сетку с клетками и визуализирует состояние каждой клетки, также принимая на вход параметр видимости кораблей на поле. Расчет координат x и y происходит посредством двух циклов for соответственно. В цикле if происходит выбор цвета в зависимости от состояния клетки (уничтожена, подбита, промах, корабль, пустая клетка), а затем ее отрисовка (прямоугольником). Также функция делает сетку, посредством отрисовки серой рамки для каждой клетки. Затем поле размечается цифрами (1-10) и буквами (A-J).

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 22 – Отрисовка поля (функция drawGrid())

Затем в панель информации сбоку поля выводится информация о текущих расставляемых кораблях: текущий размещаемый корабль, оставшееся количество кораблей данного типа для размещения, подсказка по клавише поворота.  
Затем отрисовывается размещаемый корабль с помощью цикла, следующего после glBegin() (обозначает начало описания графического примитива – корабля в виде прямоугольника – тип примитива GL\_QUADS), обозначая вершины прямоугольника корабля. При неправильном размещении в панель информации будет выведена ошибка размещения “Invalid position!”.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 22 – Отрисовка экрана расстановки кораблей (функция renderPlacement())

**Основной игровой экран.** Отрисовка игрового поля происходит в renderGame(). В данной фазе игры присутствует 2 поля: поле игрока с видимыми кораблями и поле противника со скрытыми. Оба поля отрисовываются с заголовками обозначения принадлежности поля через функцию drawGrid() (поле игрока – с параметром showShips = true, бота – showShips = false). Затем через функцию drawShipCounts() отрисовывается текущая статистика кораблей игрока и противника в формате тип\_корабля(количество\_кораблей).

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 23 – Вывод статистики по кораблям (функция drawShipCounts())

Также под полем выводится информация о текущем ходе – игрока или бота.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 24 – Отрисовка игровых полей (функция renderGame())

Цвет клетки меняется после любого изменения ее состояния внутри drawGrid(), а отрисовка происходит при вызове glutPostRedisplay(). То есть при следующем цикле вызывается display(), затем renderGame() и drawGrid(), где клетка отрисовывается с новым цветом.

**Экран завершения игры.** Игра заканчивается после уничтожения всех кораблей игрока или бота, и выводится экран окончания игры. Отрисовка экрана завершения выполняется с помощью функции renderGameOver(). Для начала происходит очистка буфера цвета и определение результата игры при помощи вызова соответственной функции checkVictory(botGrid), которая возвращает true, если игрок победил. В центральной части экрана выводится сообщение о статусе игры - "YOU WIN!" синим цветом в случае победы игрока и "YOU LOSE!" тем же оттенком синего в случае победы бота. Ниже выводятся две текстовые подсказки с предложением начать новую игру (Press N for a new game) и с предложением выйти из игры (Press Q to quit). Обработка клавиш для поворота кораблей (R), для выхода из игры (Q) и для начала новой игры (N) происходит с помощью вызова функции handleKeyPress().

В нижней части экрана отображаются миниатюрные версии игровых полей бота и игрока - слева поле игрока и справа поле бота.

Отрисовка полей происходит с использованием уменьшенного размера клеток (20 пикселей вместо 50) и для каждого поля выполняется двойной цикл по всем клеткам (10x10), далее выбирается цвет в зависимости от состояния клетки, где красный - уничтоженный корабль, оранжевый - подбитая палуба, серый - корабль (только для яполя противника при победе), светло-голубой - пустая клетка) и происходит отрисовка квадрата выбранного цвета для каждой клетки. Для центрирования элементов используется смещение (verticalOffset), размер клеток задаётся константой miniCellSize, расстояние между полями регулируется fieldSpacing. Тексты выводятся через одну функцию - drawText().

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 25 – Отрисовка экрана конца игры, результат игры и инструкции (функция renderGameOver())

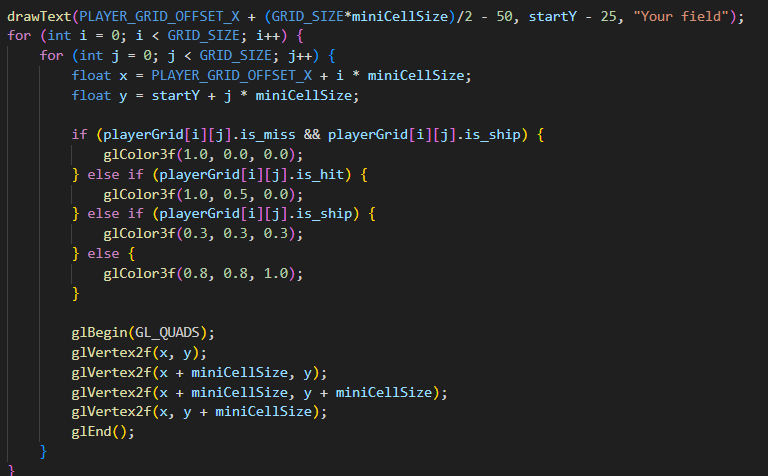


Рисунок 26 – Отрисовка финального поля игрока (функция renderGameOver())

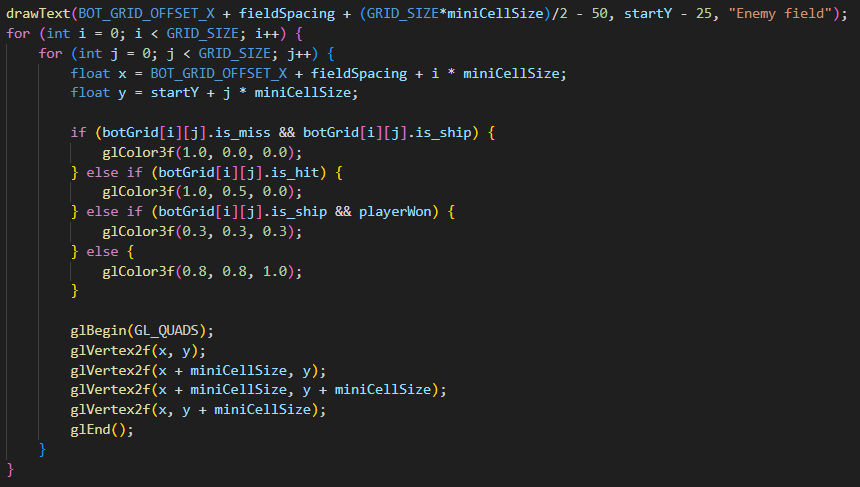


Рисунок 27 – Отрисовка финального поля противника (функция renderGameOver())

3.3 СИСТЕМА УПРАВЛЕНИЯ

В разработанной игре реализована комбинированная система управления, использующая как клавиатурный ввод, так и взаимодействие с мышью. Для обработки пользовательских действий применяются стандартные механизмы библиотеки FreeGLUT.

Обработка ввода:

1. handleMouse() – функция обработки кликов мыши (в зависимости от состояния игры).
2. handleKeyPress() – функция обработки клавиш клавиатуры.

Детализация управления по игровым состояниям:

1. Стартовое меню: любая клавиша – начать новую игру, Q – выход из игры.
2. Экран расстановки кораблей: ЛКМ – установка корабля в выбранную клетку, R – поворот текущего корабля (горизонтально/вертикально).
3. Игровой процесс: ЛКМ – выстрел по вражескому полю.
4. Экран завершения игры: N – начало новой игры, Q – выход из игры.

3.4 РЕАЛИЗАЦИЯ ГЕЙМПЛЕЯ И ИГРОВОЙ ЛОГИКИ

3.4.1 Общая игровая логика

Этап расстановки:

На экране расстановки кораблей в функции отрисовки данного экрана (renderPlacement()) при попытке размещения корабля ЛКМ вызывается handleMouse(), в которую передаются координаты текущей клетки размещения, и которая в свою очередь вызывает функцию размещения placeShip() с координатами клетки. В placeShip помимо всего прочего происходит обработка ошибки размещения через canPlaceShip(), и если ошибка была обнаружена – значение showError устанавливается на true, управление переходит обратно к renderPlacement(), где в панель информации передается сообщение об ошибке. canPlaceShip() проверяет следующие параметры: в первую очередь корабль не должен выходить за пределы сетки, затем проверяются соседние клетки – корабли не должны соприкасаться сторонами или углами. Если все проверки пройдены – функция возвращает true.



Рисунок 28 – Размещение корабля (функция placeShip())

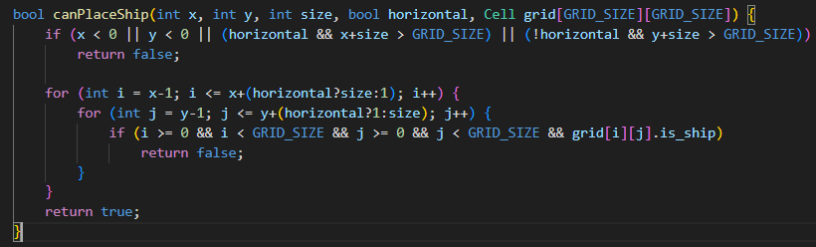


Рисунок 29 – Проверка размещения корабля (функция canPlaceShip())

Как только будут размещены все корабли (currentShipType превысит достигнет 4) – мы выйдем из цикла расстановки и начнется основной игровой процесс (фаза атаки).

Этап атаки: ход игрока всегда первый. После отрисовки экрана функции управления следят за кликами мыши, обрабатывая выстрел игрока. Сначала, после преобразования координат по клику мыши, проверяется валидность выстрела – клетка должна быть в пределах поля и не должна быть уже атакована. Затем клетка помечается как атакованная, если клетка являлась кораблем - запускается проверка уничтожения корабля (checkDestroyedShips()).

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 30 – Атака клетки, проверка на корабль (функция handleMouse())

Данная функция отвечает за проверку полного уничтожения корабля после попадания. Используется вспомогательная функция findShipParts(), которая рекурсивно обходит все соседние клетки, запоминает координаты всех палуб корабля на этих клетках и проверяет, все ли они подбиты (значение allHit) и если да – отмечает его как уничтоженный и соседние клетки – как промахи с помощью markDestroyedShip().

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 31 – Проверка подбития всех клеток корабля (функция checkDestroyedShips())

Затем проверяется условие победы (checkVictory()). Она проходит по всем клеткам поля и проверяет, есть ли в клетках корабли с неподбитыми палубами. Если находится хотя бы одна такая – игра продолжается (возвращается false), а если все корабли уничтожены – переходит к экрану конца игры.

Изображение выглядит как текст, снимок экрана, Шрифт, дисплей

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 32 – Проверка победы (функция checkVictory())

Если конца игры еще нет – ход остается у игрока. Если же игрок промахнулся (клетка не являлась кораблем) – ход переходит к боту.

3.4.2 Логика бота

Перед началом игры бот так же расставляет корабли на своем поле с помощью функции botPlaceShips(). Она перебирает все типы кораблей и делает попытки случайного размещения, генерируя случайные координаты и направление. Затем проверяет валидность размещения с помощью canPlaceShip(), размещает его, отмечая клетку как занятую (.is\_ship = true), и переходит к следующему кораблю.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 33 – Размещение кораблей бота (функция botPlaceShips())

Главная функция хода бота – botTurn(). При переходе хода к боту происходит анализ клетки – если есть попадания (botAI.count > 0), но корабль не добит, ищет валидную клетку рядом, чтобы произвести выстрел. Если таковой не находится – корабль отмечается как уничтоженный, память бота сбрасывается (обнуление попаданий в корабль).

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 34 – Сброс бота (функция resetBotAI())

Если попаданий не было, то производим случайный выстрел (задаем случайные координаты не атакованной клетки и стреляем (playerGrid[x][y].is\_hit = true)).

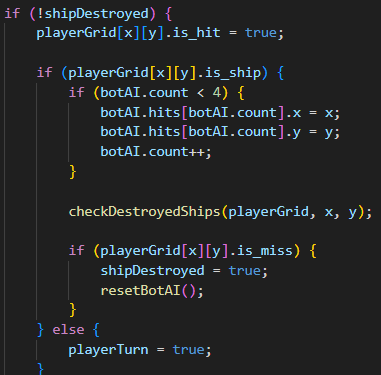


Рисунок 35 – Произведение выстрела (функция botTurn())

Если выстрел попал в корабль – бот запоминает число попаданий и координаты подбитой клетки и проверяет корабль на уничтожение. Если корабль уничтожен – сбрасываем бота. Если же бот промахнулся – ход переходит к игроку.

3.5 ИСПОЛЬЗОВАННЫЕ БИБЛИОТЕКИ

В разработке игры была использована графическая библиотека FreeGLUT (#include <GL/freeglut.h>), которая отвечает за создание оконного интерфейса, обработку пользовательского ввода с клавиатуры и мыши, а также визуализацию игровых элементов.

Для работы с базовыми функциями языка C подключены стандартные библиотеки: stdio.h (форматированный вывод текста через snprintf), stdbool.h (работа с логическими значениями), stdlib.h (генерация случайных чисел при размещении кораблей), time.h (инициализация генератора случайных чисел), string.h и memory.h (работа с памятью и обработка структур данных).

Встроенные средства FreeGLUT также позволяют выводить текст на экран без использования дополнительных библиотек.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно разработана графическая версия игры «Морской бой» на языке C с использованием библиотек OpenGL и FreeGLUT. Все поставленные задачи выполнены в полном объеме:

• Изучены и применены на практике основы работы с графическими библиотеками OpenGL и FreeGLUT, при помощи которых был реализован графический интерфейс программы.

• Игровые объекты и логика реализованы с помощью структур данных, обеспечивающих четкое разделение между состоянием игры, полями игроков и поведением компьютера.

• Полноценно функционирует основной игровой процесс, включающий расстановку кораблей, пошаговый бой и завершение игры с определением победителя.

• Разработана логика ведения игры для компьютера, позволяющая ему вести осмысленную игру с элементами стратегии.

• Создан удобный графический интерфейс с визуализацией игровых полей, подсчетом оставшихся кораблей и отображением текущего состояния игры.

Реализованный проект соответствует классическим правилам «Морского боя» и демонстрирует применение ключевых аспектов программирования: работу с памятью, обработку ввода, использование структур данных и алгоритмов. Работа над игрой позволила закрепить навыки разработки на языке C и получить практический опыт работы с графическими библиотеками.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Учебное пособие по OpenGL. [Электронный ресурс]: Лаборатория компьютерной графики при ВМиК МГУ, Фролов Антон, Игнатенко Алексей, - URL: https://www.rsdn.org/article/opengl/ogltutor.xml
2. The Open-Source OpenGL Utility Toolkit Application Programming Interface. [Электронный ресурс]: - URL: https://freeglut.sourceforge.net/docs/api.php
3. Дональд Херн, М. Паулин Бейкер "Компьютерная графика и стандарт OpenGL". - 2 изд. - Москва: Вильямс, 2004. - 688 с.
4. Программирование GLUT: окна и анимация. – [Электронный ресурс]: Linux Focus, Miguel Angel Sepulveda, - URL: http://www.linuxfocus.org/Russian/January1998/article16.html
5. Херн Д., Бейкер М.П. Компьютерная графика и стандарт OpenGL : пер. с англ. - 2-е изд. - М.: Вильямс, 2004. - 688 с.

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

#include <GL/freeglut.h>

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <memory.h>

#define GRID\_SIZE 10

#define CELL\_SIZE 50

#define WINDOW\_WIDTH 1200

#define WINDOW\_HEIGHT 650

#define PLAYER\_GRID\_OFFSET\_X 50

#define PLAYER\_GRID\_OFFSET\_Y 50

#define BOT\_GRID\_OFFSET\_X 650

#define BOT\_GRID\_OFFSET\_Y 50

typedef enum { MENU\_SCREEN, PLACEMENT\_SCREEN, GAME\_SCREEN, GAME\_OVER\_SCREEN } GameState;

typedef struct {

    bool is\_ship;

    bool is\_hit;

    bool is\_miss;

} Cell;

typedef struct {

    int size;

    int count;

} ShipType;

GameState gameState = MENU\_SCREEN;

Cell playerGrid[GRID\_SIZE][GRID\_SIZE];

Cell botGrid[GRID\_SIZE][GRID\_SIZE];

bool horizontal = true;

bool playerTurn = true;

bool gameOver = false;

bool showError = false;

const ShipType shipTypes[] = { {4,1}, {3,2}, {2,3}, {1,4} };

int currentShipType = 0;

int currentShipCount = 0;

typedef struct {

    int x;

    int y;

} Point;

typedef struct {

    Point hits[4];

    int count;

    int direction;

    Point lastShot;

} BotAI;

BotAI botAI = { 0 };

// Прототипы всех функций

void display();

void handleKeyPress(unsigned char key, int x, int y);

void handleMouse(int button, int state, int x, int y);

void initializeGame();

void resetBotAI();

void startNewGame();

bool canPlaceShip(int x, int y, int size, bool horizontal, Cell grid[GRID\_SIZE][GRID\_SIZE]);

void placeShip(int x, int y);

void botPlaceShips();

void markDestroyedShip(Cell grid[GRID\_SIZE][GRID\_SIZE], int x, int y);

void checkDestroyedShips(Cell grid[GRID\_SIZE][GRID\_SIZE], int x, int y);

bool checkVictory(Cell grid[GRID\_SIZE][GRID\_SIZE]);

void botTurn();

void drawText(float x, float y, const char\* text);

void drawGrid(int offsetX, int offsetY, bool showShips);

void drawShipCounts();

void drawPreviewShip();

void renderMenu();

void renderPlacement();

void renderGame();

void renderGameOver();

void findShipParts(Cell grid[GRID\_SIZE][GRID\_SIZE],

    int cx, int cy,

    bool visited[GRID\_SIZE][GRID\_SIZE],

    int shipCells[][2],

    int\* count\_ptr);

int main(int argc, char\*\* argv) {

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

    glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

    glutCreateWindow("Sea Battle");

    glClearColor(0.9f, 0.9f, 0.9f, 1.0f);

    gluOrtho2D(0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 0);

    glutDisplayFunc(display);

    glutKeyboardFunc(handleKeyPress);

    glutMouseFunc(handleMouse);

    srand(time(NULL));

    initializeGame();

    glutMainLoop();

    return 0;

}

void initializeGame() {

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            playerGrid[i][j] = (Cell){ false, false, false };

            botGrid[i][j] = (Cell){ false, false, false };

        }

    }

    resetBotAI();

}

void resetBotAI() {

    memset(&botAI, 0, sizeof(BotAI));

}

void startNewGame() {

    initializeGame();

    gameState = PLACEMENT\_SCREEN;

    currentShipType = 0;

    currentShipCount = 0;

    playerTurn = true;

    gameOver = false;

    showError = false;

    horizontal = true;

    botPlaceShips();

    glutPostRedisplay();

}

bool canPlaceShip(int x, int y, int size, bool horizontal, Cell grid[GRID\_SIZE][GRID\_SIZE]) {

    if (x < 0 || y < 0 || (horizontal && x + size > GRID\_SIZE) || (!horizontal && y + size > GRID\_SIZE))

        return false;

    for (int i = x - 1; i <= x + (horizontal ? size : 1); i++) {

        for (int j = y - 1; j <= y + (horizontal ? 1 : size); j++) {

            if (i >= 0 && i < GRID\_SIZE && j >= 0 && j < GRID\_SIZE && grid[i][j].is\_ship)

                return false;

        }

    }

    return true;

}

void placeShip(int x, int y) {

    if (currentShipType >= sizeof(shipTypes) / sizeof(shipTypes[0])) {

        return;

    }

    int size = shipTypes[currentShipType].size;

    if (!canPlaceShip(x, y, size, horizontal, playerGrid)) {

        showError = true;

        glutPostRedisplay();

        return;

    }

    showError = false;

    for (int i = 0; i < size; i++) {

        int nx = horizontal ? x + i : x;

        int ny = horizontal ? y : y + i;

        playerGrid[nx][ny].is\_ship = true;

    }

    currentShipCount++;

    if (currentShipCount >= shipTypes[currentShipType].count) {

        currentShipType++;

        currentShipCount = 0;

    }

    if (currentShipType >= sizeof(shipTypes) / sizeof(shipTypes[0])) {

        gameState = GAME\_SCREEN;

    }

    glutPostRedisplay();

}

void botPlaceShips() {

    for (int type = 0; type < sizeof(shipTypes) / sizeof(shipTypes[0]); type++) {

        for (int ship = 0; ship < shipTypes[type].count; ship++) {

            bool placed = false;

            int attempts = 0;

            while (!placed && attempts < 100) {

                int x = rand() % GRID\_SIZE;

                int y = rand() % GRID\_SIZE;

                bool dir = rand() % 2 == 0;

                if (canPlaceShip(x, y, shipTypes[type].size, dir, botGrid)) {

                    for (int i = 0; i < shipTypes[type].size; i++) {

                        int nx = dir ? x + i : x;

                        int ny = dir ? y : y + i;

                        botGrid[nx][ny].is\_ship = true;

                    }

                    placed = true;

                }

                attempts++;

            }

        }

    }

}

void markDestroyedShip(Cell grid[GRID\_SIZE][GRID\_SIZE], int x, int y) {

    if (grid[x][y].is\_miss) return;

    bool visited[GRID\_SIZE][GRID\_SIZE] = { false };

    int shipCells[GRID\_SIZE \* GRID\_SIZE][2];

    int count = 0;

    findShipParts(grid, x, y, visited, shipCells, &count);

    for (int i = 0; i < count; i++) {

        grid[shipCells[i][0]][shipCells[i][1]].is\_miss = true;

        for (int di = -1; di <= 1; di++) {

            for (int dj = -1; dj <= 1; dj++) {

                int ni = shipCells[i][0] + di;

                int nj = shipCells[i][1] + dj;

                if (ni >= 0 && ni < GRID\_SIZE && nj >= 0 && nj < GRID\_SIZE && !grid[ni][nj].is\_hit) {

                    grid[ni][nj].is\_hit = true;

                    grid[ni][nj].is\_miss = true;

                }

            }

        }

    }

}

void checkDestroyedShips(Cell grid[GRID\_SIZE][GRID\_SIZE], int x, int y) {

    if (!grid[x][y].is\_ship) return;

    grid[x][y].is\_hit = true;

    bool visited[GRID\_SIZE][GRID\_SIZE] = { false };

    int shipCells[GRID\_SIZE \* GRID\_SIZE][2];

    int count = 0;

    findShipParts(grid, x, y, visited, shipCells, &count);

    bool allHit = true;

    for (int i = 0; i < count; i++) {

        if (!grid[shipCells[i][0]][shipCells[i][1]].is\_hit) {

            allHit = false;

            break;

        }

    }

    if (allHit) {

        markDestroyedShip(grid, x, y);

    }

}

bool checkVictory(Cell grid[GRID\_SIZE][GRID\_SIZE]) {

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            if (grid[i][j].is\_ship && !grid[i][j].is\_hit) {

                return false;

            }

        }

    }

    return true;

}

void botTurn() {

    if (gameOver) return;

    int x = -1, y = -1;

    bool targetFound = false;

    bool shipDestroyed = false;

    int attempts = 0;

    if (botAI.count > 0) {

        const int dirs[4][2] = { {1,0}, {-1,0}, {0,1}, {0,-1} };

        for (int i = 0; i < botAI.count && !targetFound; i++) {

            for (int j = 0; j < 4 && !targetFound; j++) {

                x = botAI.hits[i].x + dirs[j][0];

                y = botAI.hits[i].y + dirs[j][1];

                if (x >= 0 && x < GRID\_SIZE && y >= 0 && y < GRID\_SIZE &&

                    !playerGrid[x][y].is\_hit) {

                    targetFound = true;

                }

            }

        }

        if (!targetFound) {

            markDestroyedShip(playerGrid, botAI.hits[0].x, botAI.hits[0].y);

            resetBotAI();

            shipDestroyed = true;

        }

    }

    if (!targetFound && !shipDestroyed) {

        do {

            x = rand() % GRID\_SIZE;

            y = rand() % GRID\_SIZE;

            attempts++;

        } while (playerGrid[x][y].is\_hit && attempts < 200);

    }

    if (!shipDestroyed) {

        playerGrid[x][y].is\_hit = true;

        if (playerGrid[x][y].is\_ship) {

            if (botAI.count < 4) {

                botAI.hits[botAI.count].x = x;

                botAI.hits[botAI.count].y = y;

                botAI.count++;

            }

            checkDestroyedShips(playerGrid, x, y);

            if (playerGrid[x][y].is\_miss) {

                shipDestroyed = true;

                resetBotAI();

            }

        }

        else {

            playerTurn = true;

        }

    }

    if (checkVictory(playerGrid)) {

        gameOver = true;

        gameState = GAME\_OVER\_SCREEN;

    }

    glutPostRedisplay();

    if (!playerTurn && !gameOver) {

        glutTimerFunc(1000, (void (\*)(int))botTurn, 0);

    }

}

void drawText(float x, float y, const char\* text) {

    glRasterPos2f(x, y);

    for (const char\* c = text; \*c != '\0'; c++) {

        glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*c);

    }

}

void drawGrid(int offsetX, int offsetY, bool showShips) {

    Cell(\*grid)[GRID\_SIZE] = showShips ? playerGrid : botGrid;

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            float x = offsetX + i \* CELL\_SIZE;

            float y = offsetY + j \* CELL\_SIZE;

            if (grid[i][j].is\_miss && grid[i][j].is\_ship) {

                glColor3f(1.0, 0.0, 0.0);

            }

            else if (grid[i][j].is\_hit && grid[i][j].is\_ship) {

                glColor3f(1.0, 0.5, 0.0);

            }

            else if (grid[i][j].is\_hit) {

                glColor3f(0.8, 0.8, 0.8);

            }

            else if (showShips && grid[i][j].is\_ship) {

                glColor3f(0.3, 0.3, 0.3);

            }

            else {

                glColor3f(0.8, 0.8, 1.0);

            }

            glBegin(GL\_QUADS);

            glVertex2f(x, y);

            glVertex2f(x + CELL\_SIZE, y);

            glVertex2f(x + CELL\_SIZE, y + CELL\_SIZE);

            glVertex2f(x, y + CELL\_SIZE);

            glEnd();

            glColor3f(0.5, 0.5, 0.5);

            glBegin(GL\_LINE\_LOOP);

            glVertex2f(x, y);

            glVertex2f(x + CELL\_SIZE, y);

            glVertex2f(x + CELL\_SIZE, y + CELL\_SIZE);

            glVertex2f(x, y + CELL\_SIZE);

            glEnd();

        }

    }

    glColor3f(0.0, 0.0, 0.0);

    for (int i = 0; i < GRID\_SIZE; i++) {

        char letter[2] = { 'A' + i, '\0' };

        drawText(offsetX + i \* CELL\_SIZE + CELL\_SIZE / 2 - 5, offsetY - 25, letter);

        char number[3];

        snprintf(number, sizeof(number), "%d", i + 1);

        drawText(offsetX - 25, offsetY + i \* CELL\_SIZE + CELL\_SIZE / 2 - 5, number);

    }

}

void drawShipCounts() {

    char text[50];

    glColor3f(0.0, 0.0, 0.0);

    // Инициализация счетчиков

    int playerShips[4] = { shipTypes[0].count, shipTypes[1].count, shipTypes[2].count, shipTypes[3].count };

    int botShips[4] = { shipTypes[0].count, shipTypes[1].count, shipTypes[2].count, shipTypes[3].count };

    // Подсчет уничтоженных кораблей бота

    bool botProcessed[GRID\_SIZE][GRID\_SIZE] = { false };

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            if (botGrid[i][j].is\_ship && botGrid[i][j].is\_miss && !botProcessed[i][j]) {

                // Нашли подбитую палубу - проверяем весь корабль

                bool visited[GRID\_SIZE][GRID\_SIZE] = { false };

                int shipSize = 0;

                bool fullyDestroyed = true;

                // Нерекурсивный обход корабля

                int stack[GRID\_SIZE \* GRID\_SIZE][2];

                int stackSize = 0;

                stack[stackSize][0] = i;

                stack[stackSize][1] = j;

                stackSize++;

                while (stackSize > 0) {

                    stackSize--;

                    int x = stack[stackSize][0];

                    int y = stack[stackSize][1];

                    if (x < 0 || x >= GRID\_SIZE || y < 0 || y >= GRID\_SIZE ||

                        visited[x][y] || !botGrid[x][y].is\_ship) {

                        continue;

                    }

                    visited[x][y] = true;

                    botProcessed[x][y] = true;

                    shipSize++;

                    // Если хоть одна палуба не подбита - корабль не уничтожен полностью

                    if (!botGrid[x][y].is\_miss) {

                        fullyDestroyed = false;

                    }

                    // Добавляем соседей в стек

                    stack[stackSize][0] = x + 1; stack[stackSize][1] = y; stackSize++;

                    stack[stackSize][0] = x - 1; stack[stackSize][1] = y; stackSize++;

                    stack[stackSize][0] = x; stack[stackSize][1] = y + 1; stackSize++;

                    stack[stackSize][0] = x; stack[stackSize][1] = y - 1; stackSize++;

                }

                // Если корабль полностью уничтожен, уменьшаем соответствующий счетчик

                if (fullyDestroyed) {

                    for (int t = 0; t < 4; t++) {

                        if (shipTypes[t].size == shipSize && botShips[t] > 0) {

                            botShips[t]--;

                            break;

                        }

                    }

                }

            }

        }

    }

    // Аналогичный подсчет для игрока

    bool playerProcessed[GRID\_SIZE][GRID\_SIZE] = { false };

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            if (playerGrid[i][j].is\_ship && playerGrid[i][j].is\_miss && !playerProcessed[i][j]) {

                bool visited[GRID\_SIZE][GRID\_SIZE] = { false };

                int shipSize = 0;

                bool fullyDestroyed = true;

                int stack[GRID\_SIZE \* GRID\_SIZE][2];

                int stackSize = 0;

                stack[stackSize][0] = i;

                stack[stackSize][1] = j;

                stackSize++;

                while (stackSize > 0) {

                    stackSize--;

                    int x = stack[stackSize][0];

                    int y = stack[stackSize][1];

                    if (x < 0 || x >= GRID\_SIZE || y < 0 || y >= GRID\_SIZE ||

                        visited[x][y] || !playerGrid[x][y].is\_ship) {

                        continue;

                    }

                    visited[x][y] = true;

                    playerProcessed[x][y] = true;

                    shipSize++;

                    if (!playerGrid[x][y].is\_miss) {

                        fullyDestroyed = false;

                    }

                    stack[stackSize][0] = x + 1; stack[stackSize][1] = y; stackSize++;

                    stack[stackSize][0] = x - 1; stack[stackSize][1] = y; stackSize++;

                    stack[stackSize][0] = x; stack[stackSize][1] = y + 1; stackSize++;

                    stack[stackSize][0] = x; stack[stackSize][1] = y - 1; stackSize++;

                }

                if (fullyDestroyed) {

                    for (int t = 0; t < 4; t++) {

                        if (shipTypes[t].size == shipSize && playerShips[t] > 0) {

                            playerShips[t]--;

                            break;

                        }

                    }

                }

            }

        }

    }

    // Вывод информации

    snprintf(text, sizeof(text), "Your ships: 4(%d) 3(%d) 2(%d) 1(%d)",

        playerShips[0], playerShips[1], playerShips[2], playerShips[3]);

    drawText(PLAYER\_GRID\_OFFSET\_X, PLAYER\_GRID\_OFFSET\_Y + GRID\_SIZE \* CELL\_SIZE + 20, text);

    snprintf(text, sizeof(text), "Enemy ships: 4(%d) 3(%d) 2(%d) 1(%d)",

        botShips[0], botShips[1], botShips[2], botShips[3]);

    drawText(BOT\_GRID\_OFFSET\_X, BOT\_GRID\_OFFSET\_Y + GRID\_SIZE \* CELL\_SIZE + 20, text);

}

void drawPreviewShip() {

    if (gameState != PLACEMENT\_SCREEN || currentShipType >= 4) return;

    int size = shipTypes[currentShipType].size;

    glColor4f(0.0, 1.0, 0.0, 0.5);

    glBegin(GL\_QUADS);

    for (int i = 0; i < size; i++) {

        float x = PLAYER\_GRID\_OFFSET\_X + GRID\_SIZE \* CELL\_SIZE + 20 + (horizontal ? i : 0) \* CELL\_SIZE;

        float y = 160 + (horizontal ? 0 : i) \* CELL\_SIZE;

        glVertex2f(x, y);

        glVertex2f(x + CELL\_SIZE, y);

        glVertex2f(x + CELL\_SIZE, y + CELL\_SIZE);

        glVertex2f(x, y + CELL\_SIZE);

    }

    glEnd();

    char info[50];

    snprintf(info, sizeof(info), "Ship: %d-decker", size);

    drawText(PLAYER\_GRID\_OFFSET\_X + GRID\_SIZE \* CELL\_SIZE + 20, 130, info);

    snprintf(info, sizeof(info), "Left: %d", shipTypes[currentShipType].count - currentShipCount);

    drawText(PLAYER\_GRID\_OFFSET\_X + GRID\_SIZE \* CELL\_SIZE + 20, 100, info);

}

void renderMenu() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.1, 0.1, 0.5);

    drawText(WINDOW\_WIDTH / 2 - 100, WINDOW\_HEIGHT / 2 - 50, "SEA BATTLE");

    glColor3f(0.0, 0.0, 0.0);

    drawText(WINDOW\_WIDTH / 2 - 120, WINDOW\_HEIGHT / 2 + 50, "Press any key to start");

    drawText(WINDOW\_WIDTH / 2 - 200, WINDOW\_HEIGHT / 2 + 80, "Controls: R - Rotate, N - New game, Q - Quit");

    drawText(WINDOW\_WIDTH / 2 - 180, WINDOW\_HEIGHT / 2 + 110, "Placement: LMB - Place ship");

    drawText(WINDOW\_WIDTH / 2 - 180, WINDOW\_HEIGHT / 2 + 140, "Game: LMB - Shoot at enemy field");

    glutSwapBuffers();

}

void renderPlacement() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    drawText(PLAYER\_GRID\_OFFSET\_X + (GRID\_SIZE \* CELL\_SIZE) / 2 - 50, 50, "Your field");

    drawGrid(PLAYER\_GRID\_OFFSET\_X, PLAYER\_GRID\_OFFSET\_Y, true);

    const int infoX = PLAYER\_GRID\_OFFSET\_X + GRID\_SIZE \* CELL\_SIZE + 20;

    const int lineHeight = 25;

    int currentY = 100;

    if (currentShipType < 4) {

        int size = shipTypes[currentShipType].size;

        char header[50];

        snprintf(header, sizeof(header), "Current: %d-decker", size);

        drawText(infoX, currentY, header);

        currentY += lineHeight;

        char remaining[50];

        snprintf(remaining, sizeof(remaining), "Left: %d/%d",

            shipTypes[currentShipType].count - currentShipCount,

            shipTypes[currentShipType].count);

        drawText(infoX, currentY, remaining);

        currentY += lineHeight;

        drawText(infoX, currentY, "Press R to rotate");

        currentY += lineHeight \* 2;

        glColor4f(0.0, 1.0, 0.0, 0.5);

        glBegin(GL\_QUADS);

        for (int i = 0; i < size; i++) {

            float px = infoX + (horizontal ? i : 0) \* CELL\_SIZE;

            float py = currentY + (horizontal ? 0 : i) \* CELL\_SIZE;

            glVertex2f(px, py);

            glVertex2f(px + CELL\_SIZE, py);

            glVertex2f(px + CELL\_SIZE, py + CELL\_SIZE);

            glVertex2f(px, py + CELL\_SIZE);

        }

        glEnd();

        currentY += (horizontal ? CELL\_SIZE : size \* CELL\_SIZE) + lineHeight;

    }

    if (showError) {

        glColor3f(1.0, 0.0, 0.0);

        drawText(infoX, currentY, "Invalid position!");

    }

    glutSwapBuffers();

}

void renderGame() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    drawText(PLAYER\_GRID\_OFFSET\_X + (GRID\_SIZE \* CELL\_SIZE) / 2 - 50, 50, "Your field");

    drawGrid(PLAYER\_GRID\_OFFSET\_X, PLAYER\_GRID\_OFFSET\_Y, true);

    drawText(BOT\_GRID\_OFFSET\_X + (GRID\_SIZE \* CELL\_SIZE) / 2 - 50, 50, "Enemy field");

    drawGrid(BOT\_GRID\_OFFSET\_X, BOT\_GRID\_OFFSET\_Y, false);

    drawShipCounts();

    glColor3f(0.0, 0.0, 0.0);

    if (playerTurn) {

        drawText(WINDOW\_WIDTH / 2 - 150, WINDOW\_HEIGHT - 30, "Your turn - click on enemy field");

    }

    else {

        drawText(WINDOW\_WIDTH / 2 - 150, WINDOW\_HEIGHT - 30, "Enemy's turn - please wait...");

    }

    glutSwapBuffers();

}

void renderGameOver() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    bool playerWon = checkVictory(botGrid);

    const int verticalOffset = 120;

    glColor3f(0.1, 0.1, 0.5);

    if (playerWon) {

        drawText(WINDOW\_WIDTH / 2 - 100, WINDOW\_HEIGHT / 2 - verticalOffset, "YOU WIN!");

    }

    else {

        drawText(WINDOW\_WIDTH / 2 - 100, WINDOW\_HEIGHT / 2 - verticalOffset, "YOU LOSE!");

    }

    glColor3f(0.0, 0.0, 0.0);

    drawText(WINDOW\_WIDTH / 2 - 150, WINDOW\_HEIGHT / 2 + 50 - verticalOffset, "Press 'N' for new game");

    drawText(WINDOW\_WIDTH / 2 - 150, WINDOW\_HEIGHT / 2 + 100 - verticalOffset, "Press 'Q' to quit");

    const int miniCellSize = 20;

    const int fieldSpacing = 250;

    const int startY = WINDOW\_HEIGHT / 2 + 130 - verticalOffset;

    drawText(PLAYER\_GRID\_OFFSET\_X + (GRID\_SIZE \* miniCellSize) / 2 - 50, startY - 25, "Your field");

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            float x = PLAYER\_GRID\_OFFSET\_X + i \* miniCellSize;

            float y = startY + j \* miniCellSize;

            if (playerGrid[i][j].is\_miss && playerGrid[i][j].is\_ship) {

                glColor3f(1.0, 0.0, 0.0);

            }

            else if (playerGrid[i][j].is\_hit) {

                glColor3f(1.0, 0.5, 0.0);

            }

            else if (playerGrid[i][j].is\_ship) {

                glColor3f(0.3, 0.3, 0.3);

            }

            else {

                glColor3f(0.8, 0.8, 1.0);

            }

            glBegin(GL\_QUADS);

            glVertex2f(x, y);

            glVertex2f(x + miniCellSize, y);

            glVertex2f(x + miniCellSize, y + miniCellSize);

            glVertex2f(x, y + miniCellSize);

            glEnd();

        }

    }

    drawText(BOT\_GRID\_OFFSET\_X + fieldSpacing + (GRID\_SIZE \* miniCellSize) / 2 - 50, startY - 25, "Enemy field");

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            float x = BOT\_GRID\_OFFSET\_X + fieldSpacing + i \* miniCellSize;

            float y = startY + j \* miniCellSize;

            if (botGrid[i][j].is\_miss && botGrid[i][j].is\_ship) {

                glColor3f(1.0, 0.0, 0.0);

            }

            else if (botGrid[i][j].is\_hit) {

                glColor3f(1.0, 0.5, 0.0);

            }

            else if (botGrid[i][j].is\_ship && playerWon) {

                glColor3f(0.3, 0.3, 0.3);

            }

            else {

                glColor3f(0.8, 0.8, 1.0);

            }

            glBegin(GL\_QUADS);

            glVertex2f(x, y);

            glVertex2f(x + miniCellSize, y);

            glVertex2f(x + miniCellSize, y + miniCellSize);

            glVertex2f(x, y + miniCellSize);

            glEnd();

        }

    }

    glutSwapBuffers();

}

void display() {

    switch (gameState) {

    case MENU\_SCREEN: renderMenu(); break;

    case PLACEMENT\_SCREEN: renderPlacement(); break;

    case GAME\_SCREEN: renderGame(); break;

    case GAME\_OVER\_SCREEN: renderGameOver(); break;

    }

}

void handleMouse(int button, int state, int x, int y) {

    if (state != GLUT\_DOWN || gameOver) return;

    if (gameState == PLACEMENT\_SCREEN && button == GLUT\_LEFT\_BUTTON) {

        int gridX = (x - PLAYER\_GRID\_OFFSET\_X) / CELL\_SIZE;

        int gridY = (y - PLAYER\_GRID\_OFFSET\_Y) / CELL\_SIZE;

        if (gridX >= 0 && gridX < GRID\_SIZE && gridY >= 0 && gridY < GRID\_SIZE) {

            placeShip(gridX, gridY);

        }

    }

    else if (gameState == GAME\_SCREEN && button == GLUT\_LEFT\_BUTTON && playerTurn) {

        int gridX = (x - BOT\_GRID\_OFFSET\_X) / CELL\_SIZE;

        int gridY = (y - BOT\_GRID\_OFFSET\_Y) / CELL\_SIZE;

        if (gridX >= 0 && gridX < GRID\_SIZE && gridY >= 0 && gridY < GRID\_SIZE &&

            !botGrid[gridX][gridY].is\_hit) {

            botGrid[gridX][gridY].is\_hit = true;

            if (botGrid[gridX][gridY].is\_ship) {

                checkDestroyedShips(botGrid, gridX, gridY);

                if (checkVictory(botGrid)) {

                    gameOver = true;

                    gameState = GAME\_OVER\_SCREEN;

                }

                else {

                    playerTurn = true;

                }

            }

            else {

                playerTurn = false;

                glutTimerFunc(1000, (void (\*)(int))botTurn, 0);

            }

            glutPostRedisplay();

        }

    }

}

void handleKeyPress(unsigned char key, int x, int y) {

    switch (key) {

    case 'r': case 'R':

        if (gameState == PLACEMENT\_SCREEN) {

            horizontal = !horizontal;

            glutPostRedisplay();

        }

        break;

    case 'n': case 'N':

        startNewGame();

        break;

    case 'q': case 'Q':

        exit(0);

        break;

    default:

        if (gameState == MENU\_SCREEN) {

            startNewGame();

        }

        break;

    }

}

void findShipParts(Cell grid[GRID\_SIZE][GRID\_SIZE], int cx, int cy, bool visited[GRID\_SIZE][GRID\_SIZE], int shipCells[][2], int\* count\_ptr) {

    if (cx < 0 || cx >= GRID\_SIZE || cy < 0 || cy >= GRID\_SIZE || visited[cx][cy] || !grid[cx][cy].is\_ship)

    {

        return;

    }

    visited[cx][cy] = true;

    shipCells[\*count\_ptr][0] = cx;

    shipCells[\*count\_ptr][1] = cy;

    (\*count\_ptr)++;

    findShipParts(grid, cx + 1, cy, visited, shipCells, count\_ptr);

    findShipParts(grid, cx - 1, cy, visited, shipCells, count\_ptr);

    findShipParts(grid, cx, cy + 1, visited, shipCells, count\_ptr);

    findShipParts(grid, cx, cy - 1, visited, shipCells, count\_ptr);

}

ПРИЛОЖЕНИЕ Б. УСТАНОВКА ГРАФИЧЕСКИХ БИБЛИОТЕК И ЗАПУСК ПРОЕКТА

Для удобного выполнения работы было решено использовать виртуальную среду Ubuntu в VirtualBox.

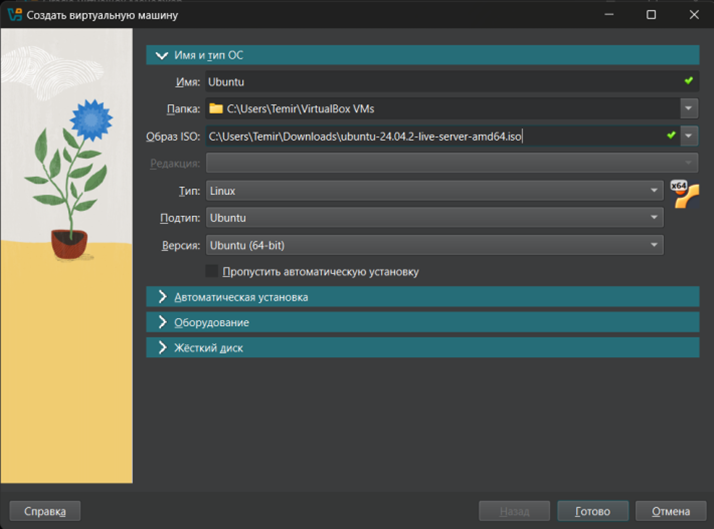


Рисунок 1 – Создание виртуальной среды Ubuntu в Oracle VirtualBox

Было изменено количество выделенной оперативной памяти (с 2-х ГБ до 4-х ГБ), количество процессоров (с 1 до 4-х) и количество выделенной видеопамяти до максимального значения в 128 МБ.

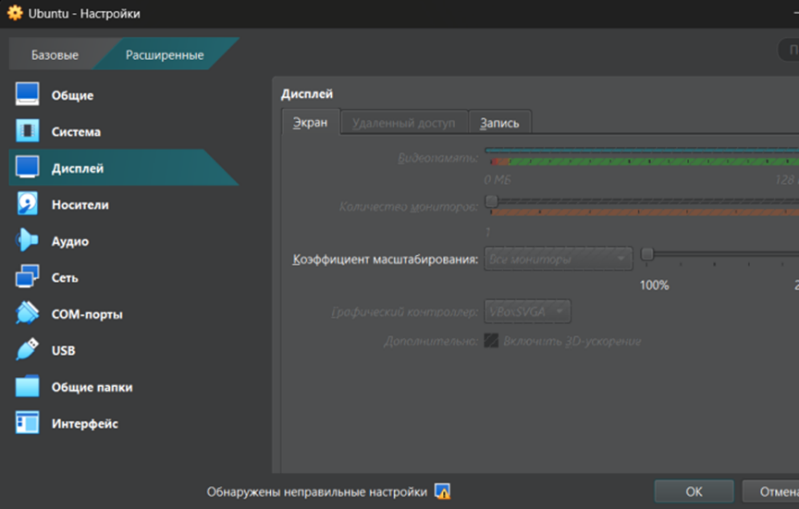


Рисунок 2 – Вид настроек виртуальной системы

Далее, в связи с тем, что сам проект GLUT не поддерживается уже около 20 лет, была использована библиотека freeglut, которая содержит в себе все те же функции, что и GLUT. Freeglut был скачан с официального сайта.

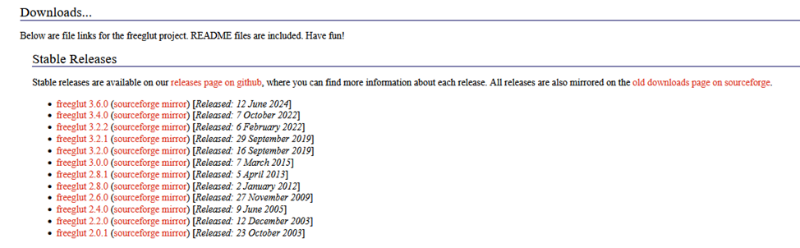


Рисунок 3 – Список выпущенных версий

Была взята новейшая версия freeglut 3.6.0.

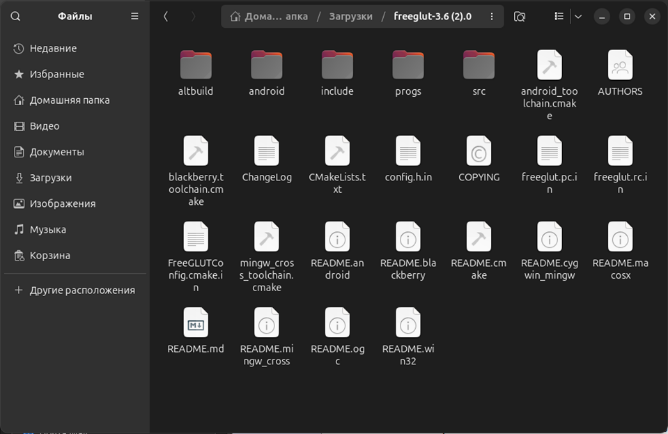


Рисунок 4 – Содержимое папки с FreeGLUT

После скачивания архив был разархивирован, откуда извлекли папку include с важным подкаталогом GL.

Папка GL из каталога include была перемещена в корневую папку проекта (course), которая также содержала файл проекта написанного на языке C (sea.c), который был создан с помощью текстового редактора nano.

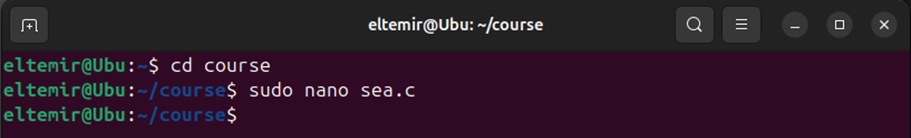


Рисунок 5 – Переход в директорию проекта и создание файла с нашим кодом (sea.c) с помощью текстового редактора Nano

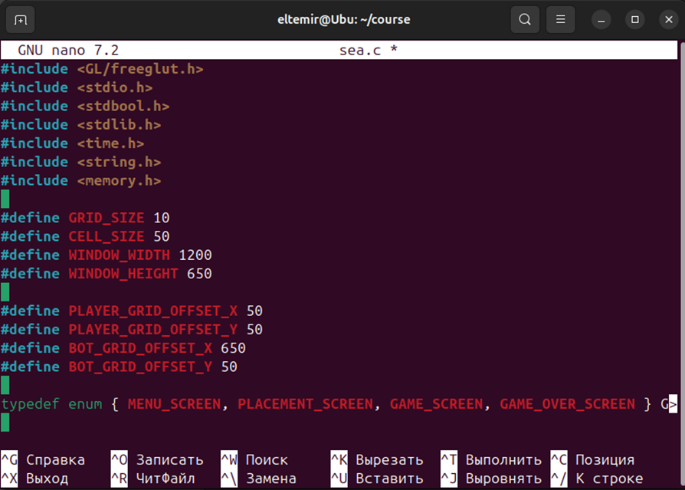


Рисунок 6 – Вид в текстовом редакторе Nano

Изображение выглядит как текст, снимок экрана, часы, число

Автоматически созданное описание

Рисунок 7 – Содержимое папки \include\GL\

В исходном коде была добавлена строка #include <GL/freeglut.h>, что позволило разработать графические функции OpenGL с использованием вызовов библиотеки.

Для успешной компиляции был использован компилятор GCC. Привязка библиотек осуществлялась с помощью аргументов -lGl, -lGLU, -lglut. Команда выполнялась следующим образом:

**gcc sea.c -o battle -lGL -lGLU -lglut**

Этот процесс обеспечил линковку необходимых библиотечных функций с основным кодом для корректной работы программы.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 8 – Компиляция исполняемого файла и запуск

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 9 – Запустившаяся игра «Морской бой»