Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**Разработка сетевой компьютерной игры "Морской бой"**

БГУИР КП 1-40 01 01 003 ПЗ

Студент Вайткун М.Г.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[Введение 5](#_Toc85934173)

[1 Анализ предметной области 6](#_Toc85934174)

[1.1 Обзор аналогов 6](#_Toc85934175)

[1.2 Постановка задачи 8](#_Toc85934176)

[2 Проектирование программного средства 9](#_Toc85934177)

[2.1 Структура программы 9](#_Toc85934178)

[2.2 Проектирование интерфейса программного средства 9](#_Toc85934179)

[2.3 Проектирование функционала программного средства 13](#_Toc85934180)

[3 Разработка программного средства 16](#_Toc85934181)

[3.1 Навигация по сайтам 16](#_Toc85934182)

[3.2 Реализация вкладок 17](#_Toc85934183)

[3.3 Работа с закладками веб-браузера 19](#_Toc85934184)

[3.4 Работа с историей посещения сайтов веб-браузера 22](#_Toc85934185)

[3.5 Работа с данными веб-браузера 23](#_Toc85934186)

[4 Тестирование программного средства 29](#_Toc85934187)

[5 Руководство пользователя 30](#_Toc85934188)

[5.1 Интерфейс программного средства 30](#_Toc85934189)

[5.2 Управление программным средством 33](#_Toc85934190)

[Заключение 34](#_Toc85934191)

[Список использованных источников 35](#_Toc85934192)

[Приложение А. Исходный код программы 36](#_Toc85934196)

# ВВЕДЕНИЕ

В современном мире компьютерные игры занимают значительное место в индустрии развлечений и образования. С развитием технологий и доступностью инструментов для разработки, создание игр стало возможным не только для крупных студий, но и для индивидуальных разработчиков и небольших команд. Одной из таких игр, которая сохраняет свою популярность на протяжении многих лет, является «Морской бой».

«Морской бой» — одна из самых узнаваемых и любимых стратегических игр, история которой насчитывает более века. Её простые, но увлекательные правила сделали её популярной среди игроков всех возрастов и поколений.Истоки игры восходят к началу XX века, когда она существовала в виде бумажной игры с использованием карандаша и бумаги. В 1967 году компания Milton Bradley выпустила пластиковую версию игры под названием «Battleship». Эта версия включала в себя пластиковые корабли и специальные доски с отверстиями для размещения кораблей и фиксации выстрелов. Именно с этого момента игра приобрела массовую популярность и стала известна во всём мире. С тех пор «Морской бой» пережил множество адаптаций и переизданий. Появились электронные версии игры, компьютерные и мобильные приложения

Популярность «Морского боя» объясняется его универсальностью и доступностью. Игра развивает логическое и стратегическое мышление, требует внимания и памяти, а также способствует развитию навыков планирования и предвидения действий соперника. Благодаря этим качествам, «Морской бой» остаётся актуальной и любимой игрой для многих поколений игроков.

В ходе работы планируется спроектировать архитектуру сетевой игры, включая клиентскую и серверную части. Разработать пользовательский интерфейс для расстановки кораблей, отображения игрового поля и результатов выстрелов, также важно реализовать логику ходов игроков и обработку событий попадания/промаха, провести тестирование стабильности соединения и корректности игрового процесса.

Реализация данной игры позволит получить практический опыт в разработке многопользовательских приложений, освоить архитектуру клиент-серверных систем и познакомиться с трудностями синхронизации данных в реальном времени. Это актуально для разработки современных многопользовательских игр и приложений, а также способствует развитию навыков программирования, проектирования и тестирования программного обеспечения.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

Игра «Морской бой» вдохновила множество разработчиков на создание различных версий и аналогов, как в настольном, так и в цифровом формате. Рассмотрим некоторые из них. Начнем с World of Warships- это массовая многопользовательская онлайн-игра, предлагающая игрокам управлять различными военными кораблями в масштабных морских сражениях. Игра отличается высокой степенью реализма и разнообразием корабельных классов. Внешний вид данной игры представлен на рисунке 1.1.



Рисунок 1.1 – игра «World of Warships»

Далее стоит рассмотреть популярную стратегическую игру сапер. Внешний вид данной игры представлен на рисунке 1.2.



Рисунок 1.2 – игра «Сапер»

"Сапёр" — это классическая логическая игра, в которой игроку необходимо очистить поле от мин, не подорвавшись на них. Игровое поле представляет собой прямоугольную сетку из закрытых клеток, некоторые из которых содержат скрытые мины. Цель игры заключается в том, чтобы открыть все безопасные клетки, избегая мин, и, при желании, пометить все мины флажками. Игрок начинает с открытия любой клетки: если в ней нет мины, появляется число, показывающее, сколько мин находится в соседних клетках. Используя эти числа, игрок должен вычислить, где находятся мины, и открыть безопасные клетки. Например, если клетка с числом "1" имеет только одну закрытую соседнюю клетку, там точно мина, а если клетка с числом "0", все соседние клетки безопасны. Игрок может ставить флажки на клетки, где, по его мнению, находятся мины, чтобы не открыть их случайно. Победа достигается, когда открыты все безопасные клетки (или помечены все мины, в зависимости от версии), а проигрыш наступает, если игрок открывает клетку с миной.

Анализ этих и других аналогов позволяет выделить ключевые особенности, которые следует учитывать при разработке собственной версии игры, а именно простота и доступность игрового процесса, возможность сетевой игры с другими игроками и интуитивно понятный пользовательский интерфейс.

## Постановка задачи

В рамках данной курсовой работы планируется разработать онлайн игру морской бой. В процессе разработки должны быть реализованы базовые функции:

* Начало новой игры: возможность начать новую сессию игры.
* Подключение к игре: возможность подключиться к существующей игре по IP-адресу.
* Возврат в главное меню: возврат на главный экран из любой точки игры.

В игровом процессе планируется разработать:

* Размещение кораблей: игроки размещают свои корабли на сетке перед началом игры как захотят сами либо наугад как предложит игра.
* Атака по координатам.
* Отображение результатов: индикация попаданий, промахов и потопленных кораблей.
* Определение победителя: игра завершается, когда у одного из игроков потоплены все корабли.
* Создание и подключение к сессии.
* Синхронизация состояния игры: обмен данными между игроками для поддержания актуального состояния игры.
* Обработка ходов: передача информации о ходе от клиента к серверу и обратно.

Для разработки программного средства будет использоваться язык программирования С# и среда разработки Godot 4.4.1 .

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

В проекте «Морской бой» используются следующие модули (скрипты и узлы), каждый из которых отвечает за свою часть логики и интерфейса:

* MainMenu.gd – отображение главного меню.
* MainMenuOnline.gd – отдельный экран выбора «Host/Join» для сетевой

игры.

* Lobby.gd – модуль для установления сетевого соединения через ENet;

содержит логику запуска сервера (host) или подключение к уже запущенному (client).

* Game.gd – центральный модуль игрового процесса.
* Bot.gd – «ИИ»-бот для одиночной игры. Хранит список доступных

клеток, случайным образом выбирает следующий выстрел.

* ShipPlacement.gd – логика автоматической (random) и ручной

расстановки кораблей на 10×10-поле.

* PauseMenu.gd – модуль паузы;
* Settings.gd – окно настроек.
* PlacementPanel – панель для расстановки кораблей.
* EndGameLayer (CanvasLayer) – полупрозрачный слой поверх игрового

поля, который при окончании партии становится видимым.

## 2.2 Проектирование интерфейса программного средства

При разработке программного средства за основу будет взят стандартный внешний вид игр.

2.2.1 Главное меню

Главное меню должно состоять из нескольких частей. В самом начале всегда должны быть кнопки начала игры, посередине настройки и в конце выход из игры. Макет главного меню представлен на рисунке 2.1.

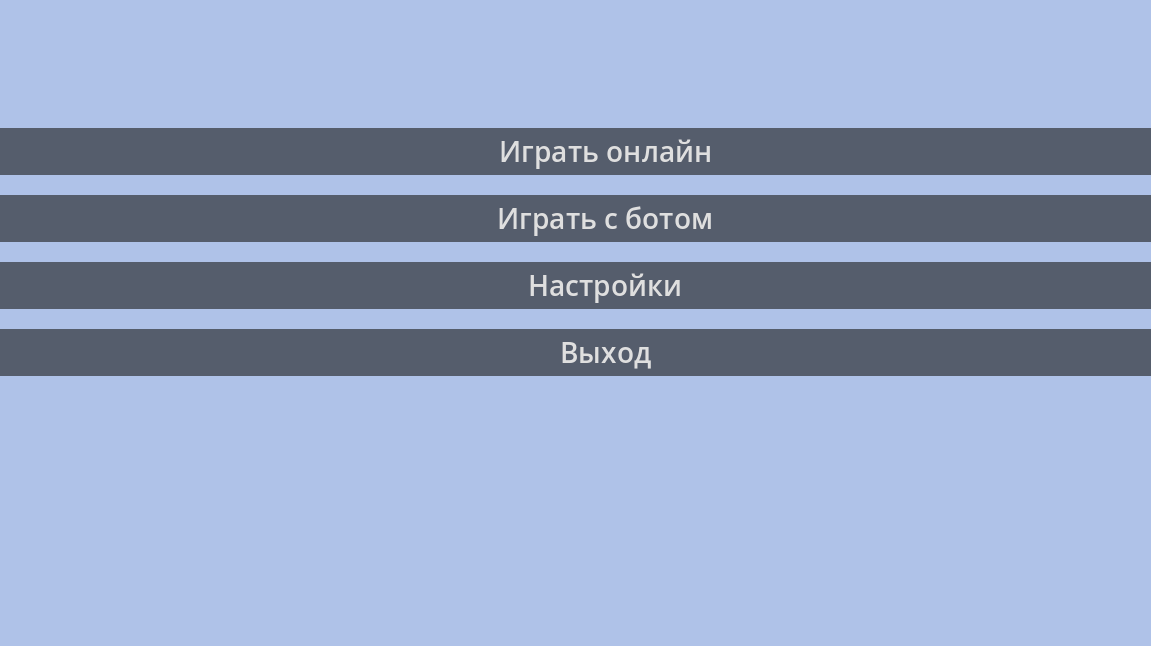


Рисунок 2.1 – Главное меню игры

2.2.2 Элементы управления игры «Морской бой»

В качестве элементов управления в игровом окне представлены следующие компоненты:

* кнопки расстановки кораблей (cамому или рандомно)
* завершить расстановку
* сама игровая сетка с кнопками (по 10×10) для отображения и выбора клетки;
* индикатор чьего хода (Label);
* кнопка «Меню» (возврат в игру, возврат в главное меню, выход из игры);
* панель с надписью «Ждём противника…» для сетевой игры.

Игровая сетка реализована как GridContainer, в котором каждая клетка это кнопка. В режиме расстановки эти кнопки принимают клик игрока для размещения одной из оставшихся частей флота. В режиме боя те же кнопки на чужой сетке позволяют совершить выстрел — при клике отправляется запрос к игровому коду, проверяется попадание, окрашивается кнопка («X» для попадания, «O» для промаха) и блокируется для повторного нажатия.

Внешний вид элементов управления игры «Морской бой» представлен на рисунке 2.2. и 2.2.1

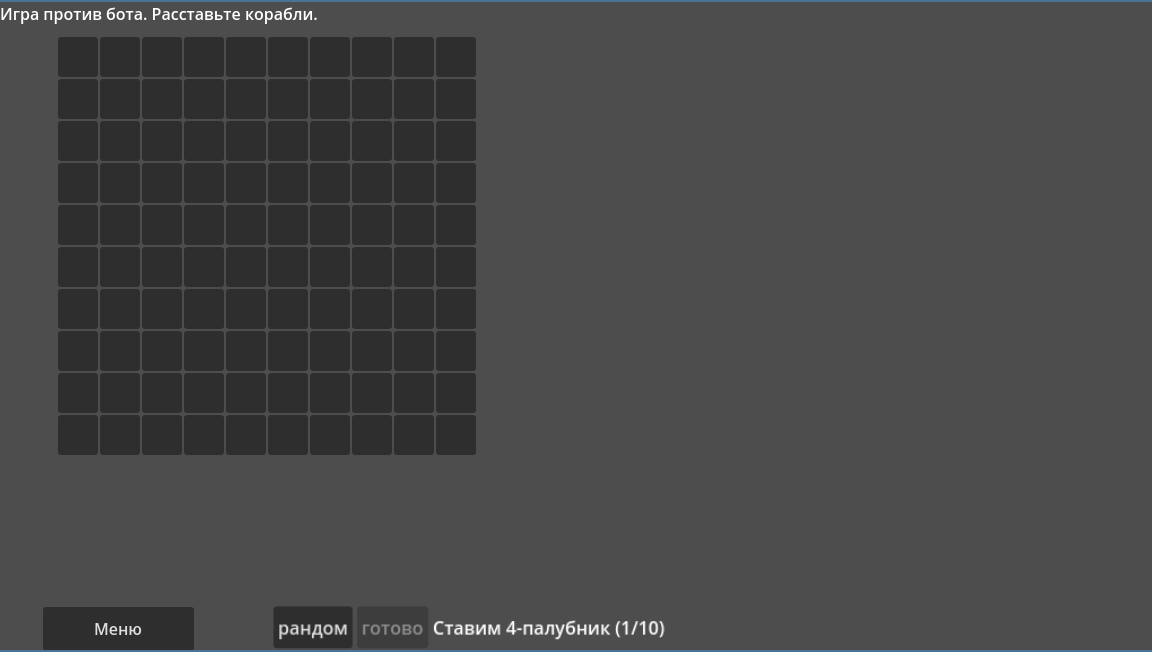


Рисунок 2.2 – Элементы управления игры «Морской бой»

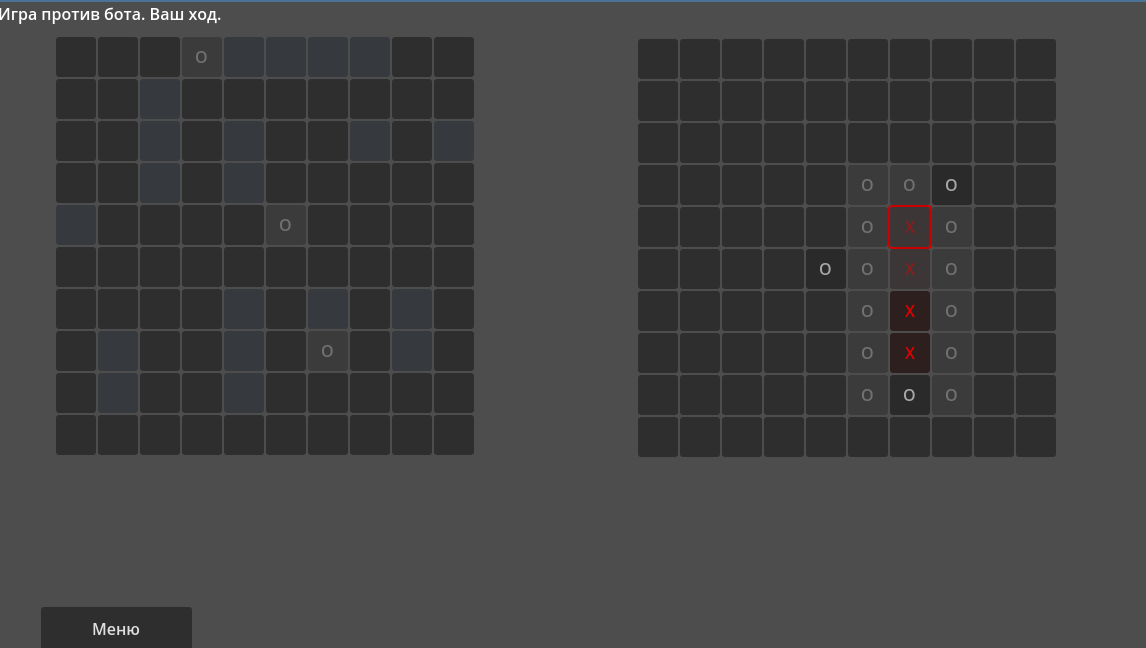


Рисунок 2.2.1 – Элементы управления игры «Морской бой»

2.2.3 Окно настроек для игры «Морской бой»

Окно настроек в проекте состоит из двух областей. В верхней части располагаются элементы управления звуком, а именно ползунок громкости (HSlider) для регулировки фоновой музыки и внизу  
кнопка «Назад», возвращающая игрока в главное меню. Макет окна настроек представлен на рисунке 2.3



Рисунок 2.3 – Макет окна настроек игры «Морской бой»

2.2.4 Окно подключения к сетевому бою

При попытке игры по сети появляется меню подключения к другому игроку, если Ip не ввести то автоматически хостом будете вы, если ввести ты вы подключитесь к хосту. Внешний вид окон представлен на рисунке 2.4.

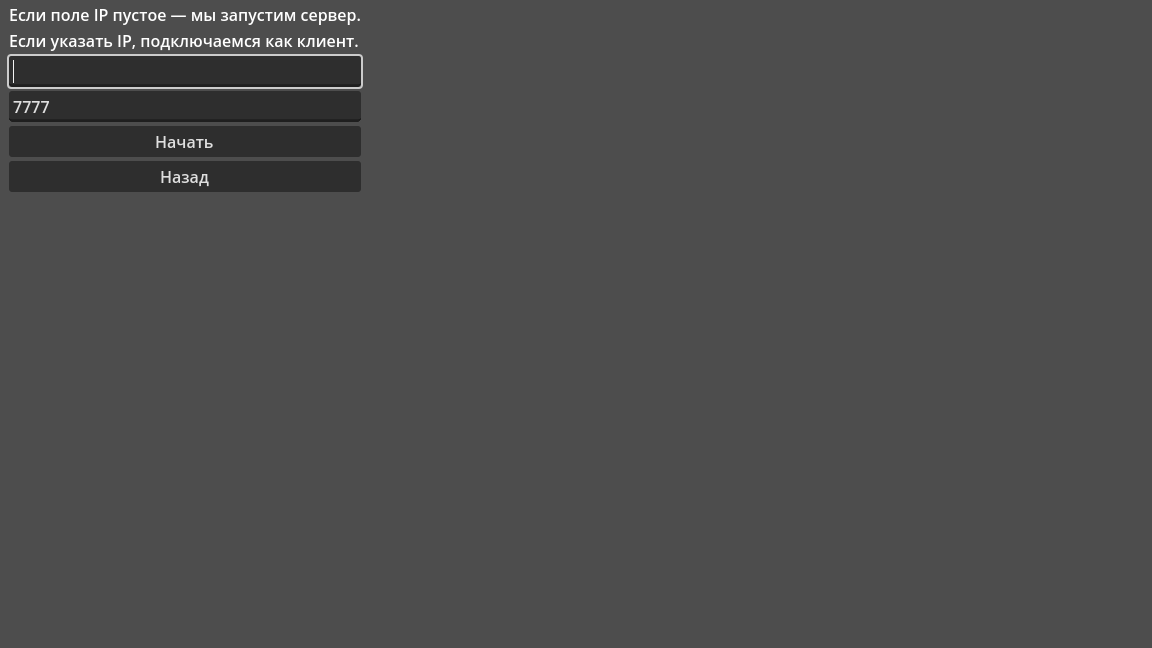


Рисунок 2.4 – Окно подключения к сетевому бою

## 2.3 Проектирование функционала программного средства

В этой секции рассмотрим логику одного из ключевых модулей проекта — ShipPlacement (для расстановки кораблей) — и приведём пример текстового описания блок-схемы для основного игрового скрипта Game.gd.

2.3.1 Размещение корабля на поле

Модуль ShipPlacement отвечает за проверку возможности разместить корабль на поле и за саму расстановку (как вручную, так и полностью «рандомом»). Блок-схема кода, осуществляющего размещение корабля на поле, представлена на рисунке 2.5.

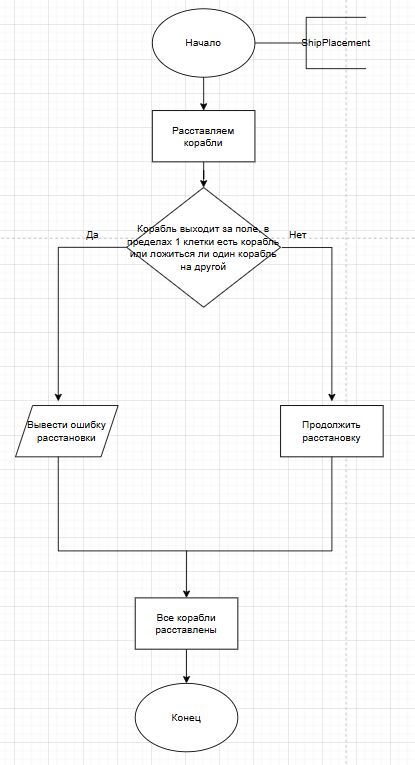


Рисунок 2.5 – Блок-схема ShipPlacement

2.3.2 Модуль бота

Бот выполняют функцию виртуального игрока если пользователю будет неудобно или неинтересно играть с реальными людьми, либо из-за каких-либо сетевых проблем он не сможет подключиться к другому пользователю. Блок-схема модуля бота представлена на рисунке 2.6.

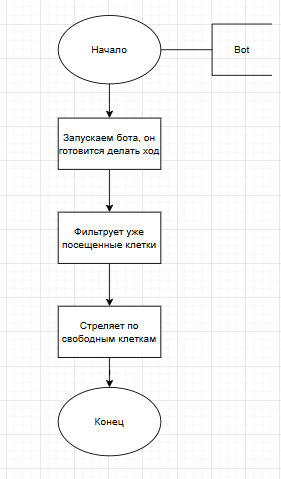


Рисунок 2.6 – Блок-схема модуля бота

2.3.3 Метод Lobby

Метод lobby отвечает за создание сервера для игроков. Если сервер уже создан, то пользователь может к нему подключиться по автоматическому порту 7777, введя ip адрес сервера, если же сервер еще не создан, то хостом автоматически становимся мы. Блок-схема метода Lobby представлена на рисунке 2.8.

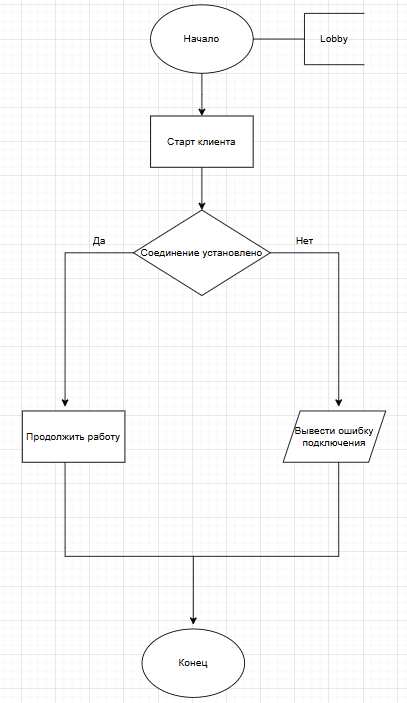


Рисунок 2.8 – Блок-схема метода Lobby

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Расстановка кораблей

Игра поддерживает два варианта расстановки: ручной и автоматический (Random). Логика сосредоточена в модуле ShipPlacement.gd, а взаимодействие с пользователем происходит через панель PlacementPanel в сцене Game.tscn.

3.1.1 Ручная расстановка

При старте фазы PLACEMENT (GameState.PLACEMENT) фоново инициализируется пустая логическая карта 10×10:

**var sp = ShipPlacement.new()**

**player\_grid = sp.initialize\_grid() # возвращает Array из 10 строк × 10 нулей**

**enemy\_grid\_logical = sp.initialize\_grid()**

**bot\_grid = sp.initialize\_grid()**

3.1.2 Автоматическая (Random) расстановка

По нажатию кнопки Random сбрасывается UI и player\_grid, затем вызывается:

**func \_on\_RandomButton\_pressed() -> void:**

**for y in range(FIELD\_SIZE):**

**for x in range(FIELD\_SIZE):**

**player\_grid[y][x] = 0**

**var btn = get\_node("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button**

**btn.text = ""**

**btn.modulate = Color.WHITE**

**btn.disabled = false**

**var sp = ShipPlacement.new()**

**sp.place\_all\_ships\_randomly(player\_grid)**

**player\_ships\_data = \_extract\_ships\_from\_grid(player\_grid)**

**for ship\_info in player\_ships\_data:**

**var sx = ship\_info["x"]**

**var sy = ship\_info["y"]**

**var sz = ship\_info["size"]**

**var sd = ship\_info["dir"]**

**for i in range(sz):**

**var dx = sx + (i if sd == 0 else 0)**

**var dy = sy + (i if sd == 1 else 0)**

**var btn\_deck = get\_node("PlayerGrid/Button\_%d\_%d" % [dy, dx]) as Button**

**btn\_deck.modulate = Color(0, 0.5, 1)**

**btn\_deck.disabled = true**

**current\_ship\_index = manual\_ships.size()**

**btn\_done.disabled = false**

**lbl\_place.text = "Все корабли расставлены!"**

Функции ShipPlacement.can\_place\_ship(...) и ShipPlacement.place\_ship(...) проверяют соседние клетки и заполняют значением 1. Метод place\_all\_ships\_randomly() последовательно проходит по списку кораблей заданных размеров и случайным образом ставит их, пока все не будут расставлены.

## 3.2 Логика выстрелов

После завершения расстановки игроков (и/или получения сетевой расстановки противника) переходим в фазу BATTLE. Игрок стреляет по вражескому полю , а при промахе ход переходит либо боту, либо второму игроку.

3.2.1 Обработка кликов на вражеском поле

В Game.gd в методе \_connect\_enemy\_grid\_buttons\_once() связываются кнопки EnemyGrid/Button\_y\_x с функцией player\_shoot(x, y):

**func \_connect\_enemy\_grid\_buttons\_once() -> void:**

**for y in range(FIELD\_SIZE):**

**for x in range(FIELD\_SIZE):**

**var path = "EnemyGrid/Button\_%d\_%d" % [y, x]**

**var btn = get\_node\_or\_null(path)**

**if btn is Button:**

**btn.text = ""**

**btn.modulate = Color.WHITE**

**btn.disabled = true**

**btn.connect("pressed", Callable(self, "player\_shoot").bind(x, y))**

Таким образом каждый выстрел по полю инициализируется и засчитывается.

3.2.2 Проверка победы и финальная табличка

Метод \_check\_win(grid) проверяет, остались ли неничейные клетка. Если нет, значит все корабли потоплены:

func \_check\_win(grid: Array) -> bool:

for row in grid:

for cell in row:

if cell == 1:

return false

return true

При окончании игры вызывается \_show\_endgame(message):

func \_show\_endgame(message: String) -> void:

**game\_over = true**

**lbl\_result.text = message # здесь ResultLabel на EndGameLayer**

**endgame\_layer.visible = true**

В сетевой игре — это RPC

**@rpc("any\_peer")**

**func rpc\_game\_over(client\_won: bool) -> void:**

**if is\_host:**

**if client\_won:**

**\_show\_endgame("Вы проиграли!")**

**else:**

**\_show\_endgame("Вы выиграли!")**

**else:**

**if client\_won:**

**\_show\_endgame("Вы выиграли!")**

**else:**

**\_show\_endgame("Вы проиграли!")**

Таким образом, табличка показывает «Вы выиграли!» только тому, кто действительно победил.

## 3.3 Работа бота

Модуль Bot.gd создаётся и добавляется в дерево при старте одиночной игры. Он хранит массив всех возможных мишеней (bot\_targets), перемешивает их и выбирает случайно. После каждого хода бот удаляет из списка уже поражённые/промахнутые клетки.

**3.3.1** Инициализация списка целей

Функцей \_initialize\_bot\_targets() поля для бота инициализируются и он принимает поля по которым нужно стрелять

extends Node

const FIELD\_SIZE: int = 10

var bot\_targets: Array = []

func \_ready() -> void:

randomize()

\_initialize\_bot\_targets()

func \_initialize\_bot\_targets() -> void:

bot\_targets.clear()

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

bot\_targets.append(Vector2(x, y))

**3.3.2** Ход бота

Когда очередь бота, Game.gd вызывает у него bot\_move().

func bot\_move() -> void:

# Находим Game-сцену (корень текущей сцены)

var game\_scene = get\_tree().get\_current\_scene() as Node

if game\_scene == null or not game\_scene.has\_method("process\_shot"):

push\_error("Bot.gd: Не найден метод process\_shot в Game!")

return

# Отфильтровать клетки, куда бот уже стрелял (2 или 3)

var new\_targets: Array = []

for coord in bot\_targets:

var x = int(coord.x)

var y = int(coord.y)

var grid\_val = game\_scene.player\_grid[y][x]

if grid\_val == 0 or grid\_val == 1:

new\_targets.append(coord)

bot\_targets = new\_targets.duplicate()

if bot\_targets.size() == 0:

return

var rnd\_idx = randi() % bot\_targets.size()

var target = bot\_targets[rnd\_idx]

var tx = int(target.x)

var ty = int(target.y)

bot\_targets.remove\_at(rnd\_idx)

print("Bot.gd: Ход бота в клетку [%d, %d]" % [tx, ty])

game\_scene.process\_shot(tx, ty)

Бот случайно выбирает одну клетку из bot\_targets и стреляет по ней

**3.3.3** Обработка выстрела

Чтобы бот мог «стрелять» в одиночном режиме, в Game.gd добавлен метод:

**func process\_shot(x: int, y: int) -> void:**

**# Проверка попадания по player\_grid**

**if player\_grid[y][x] == 1:**

**# Попадание**

**player\_grid[y][x] = 2**

**var btn\_host = get\_node("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button**

**if btn\_host:**

**btn\_host.text = "X"**

**btn\_host.modulate = Color(1, 0.5, 0)**

**btn\_host.disabled = true**

**# Обновляем hits для игрока**

**for ship in player\_ships\_data:**

**var sx = ship["x"]**

**var sy = ship["y"]**

**var sz = ship["size"]**

**var sd = ship["dir"]**

**for i in range(sz):**

**if sx + (i if sd == 0 else 0) == x and sy + (i if sd == 1 else 0) == y:**

**ship["hits"] += 1**

**if ship["hits"] == sz:**

**\_mark\_perimeter(ship, player\_grid, player\_grid\_node)**

**break**

**if \_check\_win(player\_grid):**

**\_show\_endgame("Вы проиграли!")**

**else:**

**# Промах**

**player\_grid[y][x] = 3**

**var btn\_host = get\_node("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button**

**if btn\_host:**

**btn\_host.text = "O"**

**btn\_host.modulate = Color.GRAY**

**btn\_host.disabled = true**

**# Передаём ход игроку (чтобы он мог продолжить)**

**player\_turn = true**

**\_enable\_enemy\_buttons(true)**

## 3.4 Отображение интерфейса

В сцене Game.tscn есть узел PlacementPanel: который содержит: Кнопку RotateButton – для смены ориентации. Кнопку RandomButton – для автоматической расстановки. Кнопку DoneButton – чтобы завершить фазу расстановки (разблокируется после размещения всех кораблей). Label PlacementLabel – информирует, какой корабль сейчас ставится.

btn\_rotate.connect("pressed", Callable(self, "\_on\_RotateButton\_pressed"))

btn\_random.connect("pressed", Callable(self, "\_on\_RandomButton\_pressed"))

btn\_done.connect("pressed", Callable(self, "\_on\_DoneButton\_pressed"))

**3.4.1** PlayerGrid и EnemyGrid

Обе сетки – GridContainer размером 10×10, в каждой ячейке лежит

Button\_y\_x. Изначально кнопки PlayerGrid разрешены (для расстановки), а кнопки EnemyGrid заблокированы. После окончания PLACEMENT EnemyGrid становится видимым, а кнопки включаются только во время хода.При изменении состояния (попадание/промах) меняются:

**btn.text → «X» или «O».**

**btn.modulate → Color.RED (попадание) или Color.GRAY (промах).**

**btn.disabled = true.**

**3.4.2** EndGameLayer

Над игровым полем расположен слой с именем EndGameLayer (изначально скрыт). Внутри цветная панель с подпанелью VBoxContainer, где находится: ResultLabel – текст «Вы выиграли!» или «Вы проиграли!». Кнопки реванша RematchButton и выхода в меню ExitToMenuButton.

**func \_on\_RematchButton\_pressed() -> void:**

**# сброс состояния и UI (смотри полный код Game.gd)**

**func \_on\_ExitToMenuButton\_pressed() -> void:**

**get\_tree().change\_scene\_to\_file("res://MainMenu.tscn")**

Когда \_show\_endgame(message) вызывается, EndGameLayer становится видимым, и кнопки перестают реагировать на обычную логику поверхности.

## 3.5 Сетевая игра

В модуле сетевой игры (“Network”) ключевыми задачами являются: установка связи между двумя клиентами (хост и клиент) через ENet, обмен данными о расстановке кораблей (отправка/приём массивов player\_ships\_data), организация очередности ходов через RPC-вызовы

Ниже приведена детальная разбивка по подпунктам, аналогично тому, как это было сделано в тексте про браузер.

**int** BrowserManager::getFolderSize(AnsiString folder, **int** &size)

{

    TSearchRec searchRec;

**if** (folder[folder.Length()] == '**\\**')

    {

        folder.SetLength(folder.Length() - 1);

    }

**if** (FindFirst(folder + "**\\**\*.\*", faAnyFile, searchRec) == 0)

    {

**do**

        {

**if** (searchRec.Name != "." && searchRec.Name != "..")

            {

**if** ((searchRec.Attr & faDirectory) != 0)

                {

                    getFolderSize(folder + "**\\**" + searchRec.Name, size);

                }

**else**

                {

                    size += searchRec.Size;

                }

            }

        } **while** (FindNext(searchRec) == 0);

    }

    FindClose(searchRec);

**return** size;

}

**3.5.1** Настройка Lobby и инициация соединения

Сцена Lobby.tscn содержит UI для ввода IP и порта, две кнопки «Start» и «Back», а также метку.

При старте сервера выводит подсказку: «Если поле IP пустое — мы запустим сервер. Если указать IP — подключаемся как клиент.» PortField по умолчанию содержит «7777». Все кнопки и поля редактирования активны (без блокировок).

**func \_on\_StartButton\_pressed() -> void:**

**var port: int = int(port\_field.text)**

**var typed\_ip: String = ip\_field.text.strip\_edges()**

**if typed\_ip == "":**

**# ====== Старт сервера ======**

**var peer := ENetMultiplayerPeer.new()**

**var err := peer.create\_server(port, 2)**

**if err != OK:**

**status\_label.text = "Не удалось запустить сервер на порту %d." % port**

**return**

**get\_tree().multiplayer.multiplayer\_peer = peer**

**status\_label.text = "Сервер запущен на порту %d.\nЖдём подключения клиента..." % port**

**ip\_field.editable = false**

**port\_field.editable = false**

**start\_button.disabled = true**

**back\_button.disabled = true**

**else:**

**# ====== Старт клиента ======**

**var peer := ENetMultiplayerPeer.new()**

**var err := peer.create\_client(typed\_ip, port)**

**if err != OK:**

**status\_label.text = "Не удалось подключиться к %s:%d." % [typed\_ip, port]**

**return**

**get\_tree().multiplayer.multiplayer\_peer = peer**

**status\_label.text = "Подключаемся к %s:%d..." % [typed\_ip, port]**

**ip\_field.editable = false**

**port\_field.editable = false**

**start\_button.disabled = true**

**back\_button.disabled = true**

Если IP пустое — создаётся сервер, слушающий до 2 клиентов (единственный другой клиент-партнёр). Если IP задан — клиент пытается подключиться к хосту по указанному адресу и порту. Поля редактирования и кнопки “Start”/“Back” блокируются, чтобы предотвратить повторные клики.

**func \_on\_peer\_connected(id: int) -> void:**

**status\_label.text = "Соединение установлено (peer id=%d). Переход в игру..." % id**

**await get\_tree().create\_timer(0.5).timeout**

**get\_tree().change\_scene\_to\_file("res://Game.tscn")**

**func \_on\_connection\_failed() -> void:**

**status\_label.text = "Ошибка подключения."**

**ip\_field.editable = true**

**port\_field.editable = true**

**start\_button.disabled = false**

**back\_button.disabled = false**

**func \_on\_server\_disconnected() -> void:**

**status\_label.text = "Сервер разорвал соединение."**

**ip\_field.editable = true**

**port\_field.editable = true**

**start\_button.disabled = false**

**back\_button.disabled = false**

**func \_on\_peer\_disconnected(id: int) -> void:**

**status\_label.text = "Игрок %d отключился." % id**

При успешном peer\_connected (у хоста — клиент, у клиента — он сам) ждём 0.5 с и переключаемся в Game.tscn.В случае неудачи соединения или разрыва — возвращаем возможность редактирования полей.

**3.5.2** Отправка и приём расстановки кораблей

После того как игрок нажал “Done” на этапе расстановки (\_on\_DoneButton\_pressed() в Game.gd) и если выбран сетевой режим:

**func \_on\_DoneButton\_pressed() -> void:**

**placement\_panel.visible = false**

**enemy\_grid\_node.visible = true**

**if game\_mode == MODE\_NETWORK:**

**state = GameState.WAITING\_FOR\_OPPONENT**

**wait\_label.visible = true**

**status\_label.text = "Вы готовы. Ждём, пока противник тоже будет готов..."**

**if is\_host:**

**# Отправляем всем подключённым клиентам**

**for peer\_id in get\_multiplayer().get\_peers():**

**rpc\_id(peer\_id, "rpc\_send\_ships", player\_ships\_data)**

**else:**

**# Если мы клиент, шлём хосту (rpc\_id=1)**

**rpc\_id(1, "rpc\_send\_ships", player\_ships\_data)**

**\_try\_start\_battle()**

Хост отправляет всем клиентам (в нашем случае — ровно одному), а клиент шлёт хосту по rpc\_id(1, "rpc\_send\_ships", ...).

Сразу после отправки вызываем \_try\_start\_battle(), чтобы проверить, может быть, данные противника уже поступили раньше.

**3.5.3** Старт сетевого боя и управление ходами

func \_try\_start\_battle() -> void:

if state == GameState.WAITING\_FOR\_OPPONENT and got\_enemy\_ships:

wait\_label.visible = false

state = GameState.BATTLE

if is\_host:

player\_turn = true

status\_label.text = "Оба готовы. Игра началась! Ваш ход (хост)."

\_enable\_enemy\_buttons(true)

else:

player\_turn = false

status\_label.text = "Оба готовы. Игра началась! Ждём хода хоста."

\_enable\_enemy\_buttons(false)

Проверяем, что мы в состоянии ожидания и что пришли координаты врага. Если мы хост, сразу даём себе ход (player\_turn = true) и активируем кнопки (вражеские поля). Если мы клиент, ход ещё не наш, поэтому окна для стрельбы остаются заблокированы, и ждём RPC от хоста.

**3.5.4** Выстрелы через RPC

Выстрелы происходят по нажатию кнопок (по вражескому полю). Обработчик события вызывает метод player\_shoot(x: int, y: int). Код данного метода приведен ниже.

func player\_shoot(x: int, y: int) -> void:

if state != GameState.BATTLE or game\_over or not player\_turn:

return

var btn = get\_node("EnemyGrid/Button\_%d\_%d" % [y, x]) as Button

if not btn:

return

if game\_mode == MODE\_NETWORK:

# Проверяем, что в enemy\_grid\_logical ранее не было ни попаданий (2), ни промахов (3)

if enemy\_grid\_logical[y][x] in [2, 3]:

return

# Сразу блокируем эту кнопку — чтобы дважды не нажали

btn.disabled = true

# Забираем ход

player\_turn = false

\_enable\_enemy\_buttons(false)

# Посылаем RPC “выстрел” тому, чей интересует: хост стреляет клиенту, клиент — хосту

if is\_host:

var peers = get\_multiplayer().get\_peers()

if peers.size() > 0:

var client\_id = peers[0]

rpc\_id(client\_id, "rpc\_player\_shot", x, y)

else:

rpc\_id(1, "rpc\_player\_shot", x, y)

}

Таким образом, протекает цепочку событий сетевой игры. Это обеспечивает полноценную сетевую игру между двумя игроками игры «Морской бой».

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Основная затруднительная ситуация, с которой мне пришлось столкнулся в процессе разработки модуля ручной расстановки, заключается в том, что при попытке повернуть корабль он либо всегда ставится горизонтально, либо всегда вертикально, и выбор ориентации не меняется. Вместо того чтобы реально переворачиваться в режиме редактирования (т. е. «отражаться» относительно горизонтали), корабли всегда расходятся по одной оси — фактически всегда остаются фиксированными.

Мне не удалось решить эту проблему, поскольку механизм обработки клика правой кнопкой и присвоения переменной current\_direction либо не срабатывает, либо не передаётся дальше в метод can\_place\_ship/place\_ship. В итоге при клике «повернуть» ориентация остаётся прежней, а при ручном размещении штатно используется не переменная current\_direction, а всегда выбирается либо «0» (горизонтально), либо «1» (вертикально) по умолчанию.

Из-за этого игрок может вручную поставить корабли в случайном направлении — поэтому периодически приходится полагаться на кнопку «Рандом», которая автоматически выставляет корабли как горизонтально, так и вертикально в подходящих позициях. Работа над тем, чтобы отладить логику правого клика и корректно изменить current\_direction до передачи в place\_ship, не дала результата. В текущей версии проекта лодки при ручной расстановке ориентируются только по умолчанию, а попытка изменить ориентацию не даёт эффекта.

Проблема возникла из-за недочетов на стадии проектирования программного средства.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Интерфейс программного средства

5.1.1 Главное меню

Главное меню игры состоит из компонентов которые предлагают пользователю дальнейшее взаимодействие с программным средстов. Внешний вид главного меню морского боя представлен на рисунке 5.1.

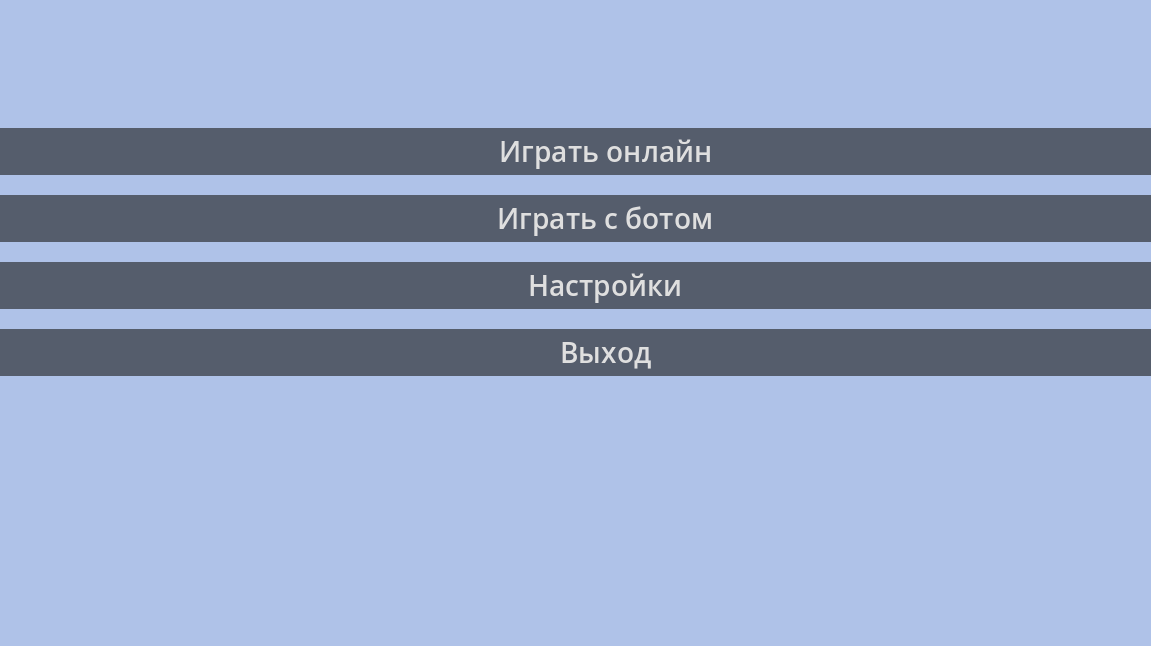


Рисунок 5.1 – Главное меню игры

5.1.2 Настройки

Окно настроек содержит в себе ползунок громкости, который определяет уровень громкости внутриигровой музыки, Внешний вид окна настроек представлен на рисунке 5.1.



Рисунок 5.2 – Окно настроек

5.1.3 Окно расстановки кораблей

При нажатии кнопки играть с ботом или играть онлайн, игра перекидывает пользователя в окно, в котором игрок может расставить билеты сам либо же нажать кнопку рандом и они расставятся в случайном порядке. Внешний вид расстановки кораблей представлен на рисунке 5.3.

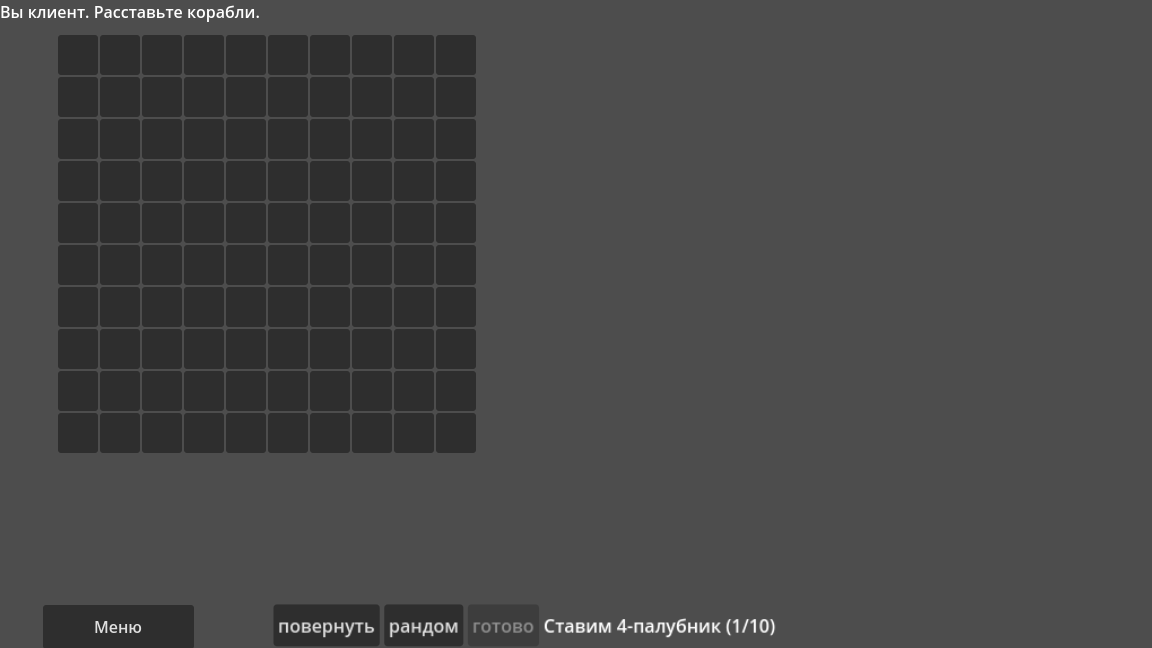


Рисунок 5.3 – Окно расстановки кораблей

5.1.4 Начало игры

После того как игрок расставил корбали начинается бой, в случае попадания по вражескому полю клетка перекрашивается в красный цвет и на нем рисуется крестик, в случае промаха, она перекрашивается в серый и рисуется нолик. Если корабль был убит то вокруг него автоматически рисуются кружки. Внешний вид игры морской бой представлен на рисунке 5.4.

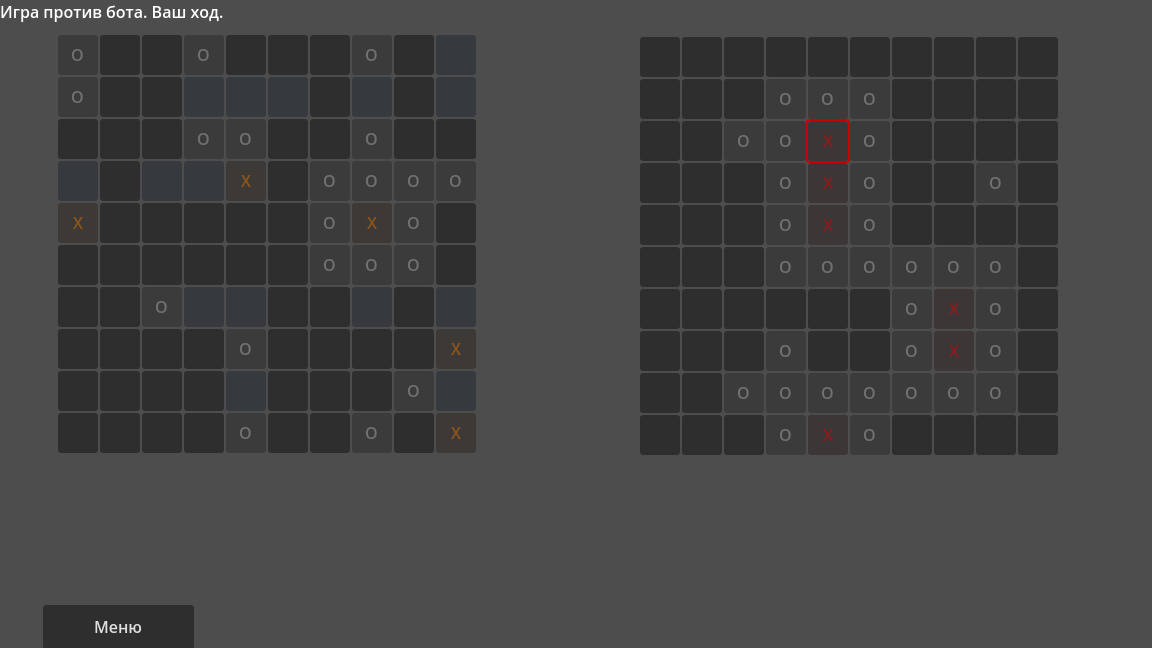


Рисунок 5.4 – Внешний вид игры морской бой

5.1.5 Середина игры

После того как вы начали играть вы можете нажать кнопку меню после чего вас перекинет в меню, где вы можете выйти из игры, выйти в меню или вернуться в нее если. Меню во время игры морской бой представлен на рисунке 5.5.

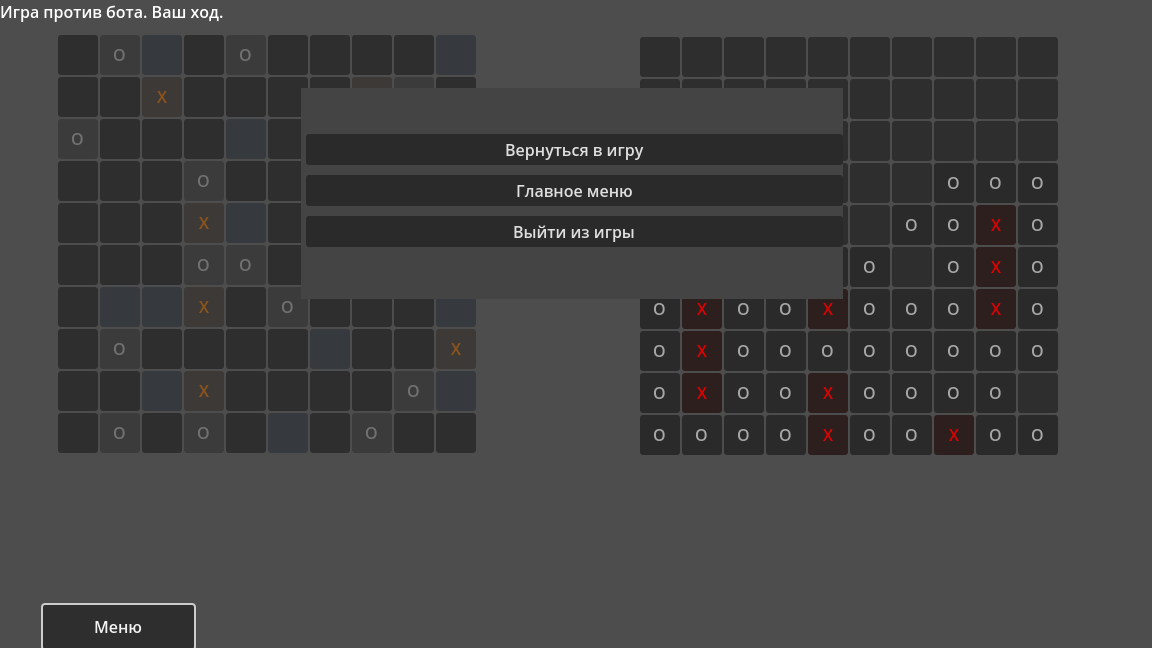


Рисунок 5.5 – Меню во время игры морской бой

## 5.2 Управление программным средством

5.2.1 Элементы управления процессом

Игровой интерфейс содержит основные элементы управления, позволяющие пользователю интуитивно взаимодействовать с программой. Интерфейс включает в себя следующие компоненты:

– Игровое поле пользователя;

– Игровое поле противника (бота или другого игрока);

– Панель управления (кнопки запуска новой игры, выхода и т. д.);

– Информационные сообщения (чей сейчас ход, результат выстрела);

– Визуальные индикаторы попаданий, промахов и потопленных кораблей.

При наведении курсора на элементы интерфейса отображаются всплывающие подсказки с кратким описанием их назначения. В ходе использования приложения пользователю доступны следующие функции:

– Размещение кораблей перед началом игры (вручную или автоматически);

– Режим одиночной игры с ИИ (ботом);

– Режим сетевой игры (по локальной сети или интернету);

– Ход по очереди: выбор клетки для выстрела по полю противника;

– Отображение результатов хода: промах, попадание, потоплен корабль;

– Отслеживание оставшихся кораблей;

– Завершение игры при победе одного из участников;

– Запуск новой игры;

– Выход в главное меню;

– Завершение работы приложения.

5.2.2 Управление настройками игры

Настройки игры доступны в отдельном окне или меню. Пользователь может изменять параметры, влияющие на игровой процесс и отображение интерфейса:

– Выбор режима игры: одиночная игра или сетевая;

– Изменение внешнего вида клеток (цветов при попадании и промахов);

– Включение/отключение музыкального фона;

– Настройка сетевого соединения (IP-адрес, порт и режим сервера/клиента);

# ЗАКЛЮЧЕНИЕ

На сегодняшний день игры остаются одними из самых популярных видов программного обеспечения как для развлечения, так и для тренировки логического мышления. Одной из таких классических игр является «Морской бой», известный многим с детства.

В рамках данного курсового проекта было разработано программное средство «Морской бой», позволяющее пользователям запускать игру в режиме одиночного боя с компьютером, а также в сетевом режиме с другим игроком. Программа реализована с использованием современного фреймворка (Godot Engine), и предлагает пользователю удобный и интуитивно понятный интерфейс.

В процессе разработки были выполнены все поставленные задачи:

– Создан графический интерфейс с двумя игровыми полями;

– Реализован алгоритм расстановки кораблей вручную и автоматически;

– Обеспечена логика поочерёдных ходов игроков;

– Внедрена система отображения результатов каждого выстрела;

– Разработан режим одиночной игры против ИИ с базовым уровнем сложности;

– Реализован сетевой режим с возможностью подключения по IP-адресу;

– Добавлены основные настройки и функции управления игрой.

Программа «Морской бой» может быть расширена и улучшена в будущем:

– Усовершенствование ИИ (например, внедрение алгоритма вероятностного анализа);

– Расширение настроек внешнего вида (темы, размеры поля);

– Поддержка рейтинговых матчей и статистики побед/поражений;

– Добавление анимаций и звукового сопровождения;

– Возможность игры через интернет с другими пользователями;

– Портирование на мобильные платформы.

Разработка игры «Морской бой» позволила закрепить навыки программирования, проектирования интерфейсов и реализации сетевого взаимодействия. Программное средство демонстрирует стабильную работу, удовлетворяет базовым требованиям жанра.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Godot Docs – master branch — Документация Godot Engine, 2021. 378

с.

[2] Симон Р. Windows 95 Win32 Programming Api Bible, 1996. — 1378с.

[3] Godot Docs – html master –2021. — 600 с.

[4] Алгоритмы. Теория и практические применение / Род Стивенс. —

Москва, 2016 — 544 с.

[5] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека

программиста. — СПб.: Питер, 2018. — 464 с.

[6] Microsoft. ASP.NET Core Documentation [Электронный ресурс].

Режим доступа: https://learn.microsoft.com/aspnet/core/, свободный.

# ПРИЛОЖЕНИЕ А. Исходный код программы

“Game.gd”

extends Node2D

# -------------------------------------------------------------------------

# Подключаем логику расстановки и класс бота

# -------------------------------------------------------------------------

const ShipPlacement = preload("res://ShipPlacement.gd")

const BotClass = preload("res://Bot.gd")

# -------------------------------------------------------------------------

# Режимы игры

# -------------------------------------------------------------------------

const MODE\_SINGLEPLAYER = 0

const MODE\_NETWORK = 1

# -------------------------------------------------------------------------

# Размер поля (10×10)

# -------------------------------------------------------------------------

const FIELD\_SIZE: int = 10

# -------------------------------------------------------------------------

# Состояния игры

# -------------------------------------------------------------------------

enum GameState { PLACEMENT, WAITING\_FOR\_OPPONENT, BATTLE }

var state: int = GameState.PLACEMENT

# -------------------------------------------------------------------------

# Логические карты 10×10:

# 0 = пусто

# 1 = палуба

# 2 = подбитая палуба

# 3 = промах

# -------------------------------------------------------------------------

var player\_grid: Array = []

var enemy\_grid\_logical: Array = []

var bot\_grid: Array = []

# -------------------------------------------------------------------------

# Флаг: чей ход (true = игрок/хозяин, false = бот/противник)

# -------------------------------------------------------------------------

var player\_turn: bool = true

var game\_over: bool = false

# -------------------------------------------------------------------------

# Режим игры: singleplayer или network

# -------------------------------------------------------------------------

var game\_mode: int = MODE\_SINGLEPLAYER

# -------------------------------------------------------------------------

# Для ручной расстановки игрока (последовательность кораблей: 4,3,3,2,2,2,1,1,1,1)

# -------------------------------------------------------------------------

var manual\_ships: Array = [4, 3, 3, 2, 2, 2, 1, 1, 1, 1]

var current\_ship\_index: int = 0

# -------------------------------------------------------------------------

# Данные по кораблям игрока и противника

# Каждый элемент: { "x": int, "y": int, "size": int, "dir": int, "hits": int }

# -------------------------------------------------------------------------

var player\_ships\_data: Array = []

var bot\_ships\_data: Array = []

var enemy\_ships\_data: Array = []

# -------------------------------------------------------------------------

# Сетевой флаг и peer

# -------------------------------------------------------------------------

var is\_host: bool = false

var is\_singleplayer: bool = false

var network\_peer: ENetMultiplayerPeer = null

var got\_enemy\_ships: bool = false # флаг, чтобы понять, что мы получили данные противника

# -------------------------------------------------------------------------

# onready‐ссылки на узлы (пути должны совпадать с деревом сцены Game.tscn)

# -------------------------------------------------------------------------

@onready var placement\_panel: Control = $PlacementPanel

@onready var btn\_random: Button = $PlacementPanel/HBoxContainer/RandomButton

@onready var btn\_done: Button = $PlacementPanel/HBoxContainer/DoneButton

@onready var lbl\_place: Label = $PlacementPanel/HBoxContainer/PlacementLabel

@onready var player\_grid\_node: GridContainer = $PlayerGrid

@onready var enemy\_grid\_node: GridContainer = $EnemyGrid

@onready var wait\_label: Label = $WaitLabel

@onready var status\_label: Label = $StatusLabel

@onready var btn\_menu: Button = $MenuButton

@onready var endgame\_layer: CanvasLayer = $EndGameLayer

@onready var lbl\_result: Label = $EndGameLayer/ColorRect/VBoxContainer/ResultLabel

@onready var btn\_rematch: Button = $EndGameLayer/ColorRect/VBoxContainer/RematchButton

@onready var btn\_exit\_to\_menu: Button = $EndGameLayer/ColorRect/VBoxContainer/ExitToMenuButton

@onready var pause\_menu: CanvasLayer = $PauseMenu

@onready var music\_player: AudioStreamPlayer2D = $MusicPlayer

var bot\_instance: Node = null

func \_ready() -> void:

randomize()

# 1) Инициализируем пустые карты 10×10

var sp = ShipPlacement.new()

player\_grid = sp.initialize\_grid()

bot\_grid = sp.initialize\_grid()

enemy\_grid\_logical = sp.initialize\_grid()

# 2) Проверяем наличие сетевого peer → если нет, играем против бота

network\_peer = get\_multiplayer().get\_multiplayer\_peer() as ENetMultiplayerPeer

if network\_peer == null:

is\_singleplayer = true

is\_host = true

game\_mode = MODE\_SINGLEPLAYER

status\_label.text = "Игра против бота. Расставьте корабли."

else:

is\_singleplayer = false

is\_host = get\_multiplayer().is\_server()

game\_mode = MODE\_NETWORK

if is\_host:

status\_label.text = "Вы хост. Расставьте корабли."

else:

status\_label.text = "Вы клиент. Расставьте корабли."

# Сетевые сигналы

get\_multiplayer().peer\_connected.connect(\_on\_peer\_connected)

get\_multiplayer().peer\_disconnected.connect(\_on\_peer\_disconnected)

get\_multiplayer().connection\_failed.connect(\_on\_connection\_failed)

get\_multiplayer().server\_disconnected.connect(\_on\_server\_disconnected)

# 3) На старте фаза PLACEMENT: показываем панель расстановки

placement\_panel.visible = true

btn\_done.disabled = true

wait\_label.visible = false

enemy\_grid\_node.visible = false

endgame\_layer.visible = false

pause\_menu.hide()

# 4) Подключаем кнопки панели расстановки

btn\_random.connect("pressed", Callable(self, "\_on\_RandomButton\_pressed"))

btn\_done.connect("pressed", Callable(self, "\_on\_DoneButton\_pressed"))

# 5) Подключаем gui\_input ко всем кнопкам PlayerGrid

\_connect\_player\_grid\_buttons()

# 6) Подключаем сигналы EnemyGrid один раз и отключаем их

\_connect\_enemy\_grid\_buttons\_once()

# 7) Кнопка «Меню»

btn\_menu.connect("pressed", Callable(self, "\_on\_MenuButton\_pressed"))

# 8) Кнопки конца игры

btn\_rematch.connect("pressed", Callable(self, "\_on\_RematchButton\_pressed"))

btn\_exit\_to\_menu.connect("pressed", Callable(self, "\_on\_ExitToMenuButton\_pressed"))

# 9) Если одиночная игра, создаём бота и добавляем в дерево

if game\_mode == MODE\_SINGLEPLAYER:

bot\_instance = BotClass.new()

add\_child(bot\_instance)

# 10) Запускаем музыку (если есть)

if music\_player:

music\_player.play()

# 11) Обновляем лейбл расстановки

\_update\_placement\_label()

print("Game.gd: PHASE=PLACEMENT — Расставьте корабли вручную или нажмите «Рандом».")

# -------------------------------------------------------------------------

# 1) PHASE: РАССТАНОВКА КОРАБЛЕЙ (PLACEMENT)

# -------------------------------------------------------------------------

func \_connect\_player\_grid\_buttons() -> void:

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

var path = "PlayerGrid/Button\_%d\_%d" % [y, x]

var btn = get\_node\_or\_null(path)

if btn is Button:

btn.text = ""

btn.modulate = Color.WHITE

btn.disabled = false

btn.connect("gui\_input", Callable(self, "\_on\_PlayerGrid\_gui\_input").bind(x, y))

func \_on\_PlayerGrid\_gui\_input(event: InputEvent, x: int, y: int) -> void:

if state != GameState.PLACEMENT or game\_over:

return

# Левый клик — попытка поставить текущий корабль

if event is InputEventMouseButton and event.button\_index == MOUSE\_BUTTON\_LEFT and event.pressed:

if current\_ship\_index >= manual\_ships.size():

return # все корабли расставлены

var size = manual\_ships[current\_ship\_index]

var sp = ShipPlacement.new()

var dir = randi() % 2 # случайная ориентация: 0 = горизонтально, 1 = вертикально

if sp.can\_place\_ship(player\_grid, x, y, size, dir):

# 1) Обозначаем «1» в логической карте

sp.place\_ship(player\_grid, x, y, size, dir)

# 2) «Окрашиваем» кнопки, отображая корабль

for i in range(size):

var dx = x + (i if dir == 0 else 0)

var dy = y + (i if dir == 1 else 0)

var btn\_path = "PlayerGrid/Button\_%d\_%d" % [dy, dx]

var deck\_btn = get\_node\_or\_null(btn\_path)

if deck\_btn is Button:

deck\_btn.modulate = Color(0, 0.5, 1) # синий

deck\_btn.disabled = true

# 3) Добавляем в player\_ships\_data

player\_ships\_data.append({

"x": x,

"y": y,

"size": size,

"dir": dir,

"hits": 0

})

# 4) Переходим к следующему кораблю

current\_ship\_index += 1

\_update\_placement\_label()

if current\_ship\_index >= manual\_ships.size():

btn\_done.disabled = false

lbl\_place.text = "Все корабли расставлены!"

else:

print("Game.gd: Нельзя разместить корабль здесь.")

return

func \_on\_RandomButton\_pressed() -> void:

# Сброс логики и UI PlayerGrid

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

player\_grid[y][x] = 0

var btn\_path = "PlayerGrid/Button\_%d\_%d" % [y, x]

var b = get\_node\_or\_null(btn\_path)

if b is Button:

b.text = ""

b.modulate = Color.WHITE

b.disabled = false

# Рандомная расстановка всего набора кораблей

var sp = ShipPlacement.new()

sp.place\_all\_ships\_randomly(player\_grid)

# Извлекаем и сохраняем данные о всех кораблях

player\_ships\_data = \_extract\_ships\_from\_grid(player\_grid)

# Окрашиваем кнопки в синий и блокируем их

for ship\_info in player\_ships\_data:

var sx = ship\_info["x"]

var sy = ship\_info["y"]

var sz = ship\_info["size"]

var sd = ship\_info["dir"]

for i in range(sz):

var dx = sx + (i if sd == 0 else 0)

var dy = sy + (i if sd == 1 else 0)

var btn\_path = "PlayerGrid/Button\_%d\_%d" % [dy, dx]

var deck\_btn = get\_node\_or\_null(btn\_path)

if deck\_btn is Button:

deck\_btn.modulate = Color(0, 0.5, 1)

deck\_btn.disabled = true

current\_ship\_index = manual\_ships.size()

btn\_done.disabled = false

lbl\_place.text = "Все корабли расставлены!"

func \_update\_placement\_label() -> void:

if current\_ship\_index < manual\_ships.size():

var size = manual\_ships[current\_ship\_index]

var placed = current\_ship\_index

var total = manual\_ships.size()

lbl\_place.text = "Ставим %d-палубник (%d/%d)" % [size, placed + 1, total]

else:

lbl\_place.text = "Готово!"

func \_on\_DoneButton\_pressed() -> void:

# Завершили расстановку → скрываем панель, показываем EnemyGrid

placement\_panel.visible = false

enemy\_grid\_node.visible = true

if game\_mode == MODE\_SINGLEPLAYER:

# Одиночная игра против бота

state = GameState.BATTLE

player\_turn = true

status\_label.text = "Игра против бота. Ваш ход."

# Бот рандомно расставляет свои корабли

var sp = ShipPlacement.new()

sp.place\_all\_ships\_randomly(bot\_grid)

bot\_ships\_data = \_extract\_ships\_from\_grid(bot\_grid)

# Активируем кнопки EnemyGrid

\_enable\_enemy\_buttons(true)

else:

# Сетевой режим: отправляем данные и ждём противника

state = GameState.WAITING\_FOR\_OPPONENT

wait\_label.visible = true

status\_label.text = "Вы готовы. Ждём, пока противник тоже будет готов..."

# --- Отправляем свои данные друг другу ---

if is\_host:

for peer\_id in get\_multiplayer().get\_peers():

rpc\_id(peer\_id, "rpc\_send\_ships", player\_ships\_data)

else:

rpc\_id(1, "rpc\_send\_ships", player\_ships\_data)

# --- Если уже получили расстановку противника, стартуем бой сразу ---

\_try\_start\_battle()

# -------------------------------------------------------------------------

# 2) PHASE: СЕТЕВАЯ ЧАСТЬ

# -------------------------------------------------------------------------

func \_on\_peer\_connected(id: int) -> void:

if is\_host:

status\_label.text = "Клиент подключился (id=%d). Ждём «Готово»." % id

else:

status\_label.text = "Подключены к хосту. Ждём «Готово»."

func \_on\_peer\_disconnected(id: int) -> void:

status\_label.text = "Игрок %d отключился." % id

func \_on\_connection\_failed() -> void:

status\_label.text = "Не удалось подключиться к серверу."

func \_on\_server\_disconnected() -> void:

status\_label.text = "Сервер разорвал соединение."

@rpc("any\_peer")

func rpc\_send\_ships(ships\_data: Array) -> void:

# Любой (хост или клиент) получит здесь расстановку врага

enemy\_ships\_data = ships\_data.duplicate()

\_fill\_enemy\_logical(enemy\_ships\_data)

got\_enemy\_ships = true

# Если локально уже нажали «Готово» → стартуем бой

\_try\_start\_battle()

func \_try\_start\_battle() -> void:

# Запускаем бой, когда оба игрока нажали «Готово» и получили данные друг друга

if state == GameState.WAITING\_FOR\_OPPONENT and got\_enemy\_ships:

wait\_label.visible = false

state = GameState.BATTLE

if is\_host:

player\_turn = true

status\_label.text = "Оба готовы. Игра началась! Ваш ход (хост)."

\_enable\_enemy\_buttons(true)

else:

player\_turn = false

status\_label.text = "Оба готовы. Игра началась! Ждём хода хоста."

\_enable\_enemy\_buttons(false)

func \_fill\_enemy\_logical(ships: Array) -> void:

# Заполняет enemy\_grid\_logical на основе списка кораблей

for ship in ships:

var sx: int = ship["x"]

var sy: int = ship["y"]

var sz: int = ship["size"]

var sd: int = ship["dir"]

for i in range(sz):

var dx = sx + (i if sd == 0 else 0)

var dy = sy + (i if sd == 1 else 0)

enemy\_grid\_logical[dy][dx] = 1

# -------------------------------------------------------------------------

# 3) PHASE: БОЙ (BATTLE)

# -------------------------------------------------------------------------

func \_connect\_enemy\_grid\_buttons\_once() -> void:

# Подключаем сигналы ровно один раз

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

var path = "EnemyGrid/Button\_%d\_%d" % [y, x]

var btn = get\_node\_or\_null(path)

if btn is Button:

btn.text = ""

btn.modulate = Color.WHITE

btn.disabled = true

btn.connect("pressed", Callable(self, "player\_shoot").bind(x, y))

func \_enable\_enemy\_buttons(enable: bool) -> void:

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

var btn = get\_node("EnemyGrid/Button\_%d\_%d" % [y, x]) as Button

btn.disabled = not (enable and player\_turn)

func player\_shoot(x: int, y: int) -> void:

if state != GameState.BATTLE or game\_over or not player\_turn:

return

var btn = get\_node("EnemyGrid/Button\_%d\_%d" % [y, x]) as Button

if not btn:

return

if game\_mode == MODE\_SINGLEPLAYER:

# — Игра против бота —

if bot\_grid[y][x] in [2, 3]:

return # уже стреляли сюда

if bot\_grid[y][x] == 1:

# Попадание

bot\_grid[y][x] = 2

btn.text = "X"

btn.modulate = Color.RED

btn.disabled = true

# Обновляем bot\_ships\_data

for ship in bot\_ships\_data:

var sx = ship["x"]

var sy = ship["y"]

var sz = ship["size"]

var sd = ship["dir"]

for i in range(sz):

if sx + (i if sd == 0 else 0) == x and sy + (i if sd == 1 else 0) == y:

ship["hits"] += 1

if ship["hits"] == sz:

\_mark\_perimeter(ship, bot\_grid, enemy\_grid\_node)

break

# Проверяем победу

if \_check\_win(bot\_grid):

\_show\_endgame("Вы выиграли!")

# Если попали, игрок остаётся в своём ходе

return

else:

# Промах

bot\_grid[y][x] = 3

btn.text = "O"

btn.modulate = Color.GRAY

btn.disabled = true

player\_turn = false

\_enable\_enemy\_buttons(false)

await get\_tree().create\_timer(1.0).timeout

if bot\_instance:

bot\_instance.bot\_move()

return

else:

# — Сетевая игра —

if enemy\_grid\_logical[y][x] in [2, 3]:

return # уже стреляли сюда

btn.disabled = true

player\_turn = false

\_enable\_enemy\_buttons(false)

if is\_host:

var peers = get\_multiplayer().get\_peers()

if peers.size() > 0:

var client\_id = peers[0]

rpc\_id(client\_id, "rpc\_player\_shot", x, y)

else:

rpc\_id(1, "rpc\_player\_shot", x, y)

# -------------------------------------------------------------------------

# Обработка выстрела противника (RPC)

# -------------------------------------------------------------------------

@rpc("any\_peer")

func rpc\_player\_shot(x: int, y: int) -> void:

var shooter\_id = multiplayer.get\_remote\_sender\_id()

var hit: bool = false

if player\_grid[y][x] == 1:

hit = true

player\_grid[y][x] = 2

else:

player\_grid[y][x] = 3

var btn\_host = get\_node("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button

if btn\_host:

if hit:

btn\_host.text = "X"

btn\_host.modulate = Color(1, 0.5, 0)

else:

btn\_host.text = "O"

btn\_host.modulate = Color.GRAY

btn\_host.disabled = true

# Если промах, сразу даём ход стреляющему

if not hit:

player\_turn = true

\_enable\_enemy\_buttons(true)

rpc\_id(shooter\_id, "rpc\_receive\_shot\_result", x, y, hit)

if hit:

for ship in player\_ships\_data:

var sx = ship["x"]

var sy = ship["y"]

var sz = ship["size"]

var sd = ship["dir"]

for i in range(sz):

if sx + (i if sd == 0 else 0) == x and sy + (i if sd == 1 else 0) == y:

ship["hits"] += 1

if ship["hits"] == sz:

\_mark\_perimeter(ship, player\_grid, player\_grid\_node)

break

if \_check\_win(player\_grid):

if is\_host:

rpc("rpc\_game\_over", true)

else:

rpc("rpc\_game\_over", false)

\_show\_endgame("Вы проиграли!")

# -------------------------------------------------------------------------

# Обработка результата своего выстрела (RPC)

# -------------------------------------------------------------------------

@rpc("any\_peer")

func rpc\_receive\_shot\_result(x: int, y: int, hit: bool) -> void:

var btn = get\_node("EnemyGrid/Button\_%d\_%d" % [y, x]) as Button

if not btn:

return

if hit:

btn.text = "X"

btn.modulate = Color.RED

enemy\_grid\_logical[y][x] = 2

for ship in enemy\_ships\_data:

var sx = ship["x"]

var sy = ship["y"]

var sz = ship["size"]

var sd = ship["dir"]

for i in range(sz):

if sx + (i if sd == 0 else 0) == x and sy + (i if sd == 1 else 0) == y:

ship["hits"] += 1

if ship["hits"] == sz:

\_mark\_perimeter(ship, enemy\_grid\_logical, enemy\_grid\_node)

break

player\_turn = true

\_enable\_enemy\_buttons(true)

else:

btn.text = "O"

btn.modulate = Color.GRAY

enemy\_grid\_logical[y][x] = 3

# Ожидаем следующего rpc\_your\_turn

@rpc("any\_peer")

func rpc\_your\_turn() -> void:

player\_turn = true

\_enable\_enemy\_buttons(true)

if is\_host:

status\_label.text = "Ваш ход (хост)."

else:

status\_label.text = "Ваш ход (клиент)."

@rpc("any\_peer")

func rpc\_game\_over(client\_won: bool) -> void:

if is\_host:

if client\_won:

\_show\_endgame("Вы проиграли!")

else:

\_show\_endgame("Вы выиграли!")

else:

if client\_won:

\_show\_endgame("Вы выиграли!")

else:

\_show\_endgame("Вы проиграли!")

# -------------------------------------------------------------------------

# ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

# -------------------------------------------------------------------------

func process\_shot(x: int, y: int) -> bool:

# Этот метод нужен для одиночной игры (бот вызывает его напрямую)

var hit = false

if player\_grid[y][x] == 1:

hit = true

player\_grid[y][x] = 2

var btn\_host = get\_node("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button

if btn\_host:

btn\_host.text = "X"

btn\_host.modulate = Color(1, 0.5, 0)

btn\_host.disabled = true

for ship in player\_ships\_data:

var sx = ship["x"]

var sy = ship["y"]

var sz = ship["size"]

var sd = ship["dir"]

for i in range(sz):

if sx + (i if sd == 0 else 0) == x and sy + (i if sd == 1 else 0) == y:

ship["hits"] += 1

if ship["hits"] == sz:

\_mark\_perimeter(ship, player\_grid, player\_grid\_node)

break

if \_check\_win(player\_grid):

\_show\_endgame("Вы проиграли!")

else:

player\_grid[y][x] = 3

var btn\_host = get\_node("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button

if btn\_host:

btn\_host.text = "O"

btn\_host.modulate = Color.GRAY

btn\_host.disabled = true

# После промаха бот должен вернуть ход игроку

player\_turn = true

\_enable\_enemy\_buttons(true)

return hit

func \_check\_win(grid: Array) -> bool:

for row in grid:

for cell in row:

if cell == 1:

return false

return true

func \_show\_endgame(message: String) -> void:

game\_over = true

lbl\_result.text = message

endgame\_layer.visible = true

func \_on\_RematchButton\_pressed() -> void:

# Сброс состояния → фаза PLACEMENT

game\_over = false

player\_turn = true

state = GameState.PLACEMENT

current\_ship\_index = 0

player\_ships\_data.clear()

bot\_ships\_data.clear()

enemy\_ships\_data.clear()

got\_enemy\_ships = false

var sp = ShipPlacement.new()

player\_grid = sp.initialize\_grid()

bot\_grid = sp.initialize\_grid()

enemy\_grid\_logical = sp.initialize\_grid()

placement\_panel.visible = true

endgame\_layer.visible = false

# Сброс UI PlayerGrid

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

var pb = get\_node\_or\_null("PlayerGrid/Button\_%d\_%d" % [y, x]) as Button

if pb:

pb.text = ""

pb.modulate = Color.WHITE

pb.disabled = false

# Сброс UI EnemyGrid

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

var eb = get\_node\_or\_null("EnemyGrid/Button\_%d\_%d" % [y, x]) as Button

if eb:

eb.text = ""

eb.modulate = Color.WHITE

eb.disabled = true

btn\_done.disabled = true

\_update\_placement\_label()

func \_on\_ExitToMenuButton\_pressed() -> void:

get\_tree().change\_scene\_to\_file("res://MainMenu.tscn")

func \_on\_MenuButton\_pressed() -> void:

get\_tree().paused = true

pause\_menu.show()

func \_extract\_ships\_from\_grid(grid: Array) -> Array:

var ships\_arr: Array = []

var visited: Array = []

for y in range(FIELD\_SIZE):

visited.append([])

for x in range(FIELD\_SIZE):

visited[y].append(false)

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

if grid[y][x] == 1 and not visited[y][x]:

var dir = 0

if x + 1 < FIELD\_SIZE and grid[y][x + 1] == 1:

dir = 0

elif y + 1 < FIELD\_SIZE and grid[y + 1][x] == 1:

dir = 1

var length = 1

if dir == 0:

var cx = x + 1

while cx < FIELD\_SIZE and grid[y][cx] == 1:

length += 1

cx += 1

else:

var cy = y + 1

while cy < FIELD\_SIZE and grid[cy][x] == 1:

length += 1

cy += 1

for i in range(length):

var dx = x + (i if dir == 0 else 0)

var dy = y + (i if dir == 1 else 0)

visited[dy][dx] = true

ships\_arr.append({ "x": x, "y": y, "size": length, "dir": dir, "hits": 0 })

return ships\_arr

func \_mark\_perimeter(ship: Dictionary, grid: Array, grid\_node: GridContainer) -> void:

var sx = ship["x"]

var sy = ship["y"]

var sz = ship["size"]

var sd = ship["dir"]

var min\_x = sx - 1

var max\_x = sx + (sz if sd == 0 else 1)

var min\_y = sy - 1

var max\_y = sy + (sz if sd == 1 else 1)

for yy in range(min\_y, max\_y + 1):

for xx in range(min\_x, max\_x + 1):

if xx < 0 or xx >= FIELD\_SIZE or yy < 0 or yy >= FIELD\_SIZE:

continue

var on\_ship = false

for i in range(sz):

if sx + (i if sd == 0 else 0) == xx and sy + (i if sd == 1 else 0) == yy:

on\_ship = true

break

if on\_ship:

continue

if grid[yy][xx] in [2, 3]:

continue

grid[yy][xx] = 3

var btn = grid\_node.get\_node\_or\_null("Button\_%d\_%d" % [yy, xx]) as Button

if btn:

btn.text = "O"

btn.modulate = Color.GRAY

btn.disabled = true

“Bot.gd”

extends Node

const FIELD\_SIZE: int = 10

var bot\_targets: Array = []

func \_ready() -> void:

randomize()

\_initialize\_bot\_targets()

func \_initialize\_bot\_targets() -> void:

bot\_targets.clear()

for y in range(FIELD\_SIZE):

for x in range(FIELD\_SIZE):

bot\_targets.append(Vector2(x, y))

func bot\_move() -> void:

var game\_scene = get\_tree().get\_current\_scene() as Node

if game\_scene == null or not game\_scene.has\_method("process\_shot"):

push\_error("Bot.gd: Не найден метод process\_shot в Game!")

return

# Фильтруем уже посещённые клетки (2 = подбитая, 3 = промах)

var new\_targets: Array = []

for coord in bot\_targets:

var x = int(coord.x)

var y = int(coord.y)

var grid\_val = game\_scene.player\_grid[y][x]

if grid\_val == 0 or grid\_val == 1:

new\_targets.append(coord)

bot\_targets = new\_targets.duplicate()

if bot\_targets.size() == 0:

return

var rnd\_idx = randi() % bot\_targets.size()

var target = bot\_targets[rnd\_idx]

var tx = int(target.x)

var ty = int(target.y)

bot\_targets.remove\_at(rnd\_idx)

print("Bot.gd: Ход бота в клетку [%d, %d]" % [tx, ty])

# Вызываем process\_shot и получаем, попал ли бот

var was\_hit: bool = game\_scene.process\_shot(tx, ty)

if was\_hit:

# Если попал, ждём 0.5 секунды и стреляет снова

await get\_tree().create\_timer(0.5).timeout

bot\_move()

# Если промах – process\_shot внутри Game вернёт игроку ход, и бот остановится

“Canvasyer.gd”

extends CanvasLayer

func \_ready():

hide() # Скрываем меню при запуске

func \_on\_resume\_button\_pressed():

hide()

get\_tree().paused = false

func \_on\_main\_menu\_button\_pressed():

get\_tree().paused = false

get\_tree().change\_scene("res://MainMenu.tscn") # Замените на путь к вашей сцене главного меню

func \_on\_quit\_button\_pressed():

get\_tree().quit()

“Globals.gd”

extends Node

# Флаг: true = мы хостим (сервер), false = мы клиент

var is\_host: bool = true

# IP и порт сервера (для клиента будет заполнено из Lobby)

var server\_ip: String = ""

var server\_port: int = 7777

“lobby.gd”

extends Control

@onready var info\_label: Label = $VBoxContainer/InfoLabel

@onready var ip\_field: LineEdit = $VBoxContainer/IPField

@onready var port\_field: LineEdit = $VBoxContainer/PortField

@onready var start\_button: Button = $VBoxContainer/StartButton

@onready var back\_button: Button = $VBoxContainer/BackButton

@onready var status\_label: Label = $VBoxContainer/StatusLabel

func \_ready() -> void:

# 1) Подсказочный текст и порт по умолчанию

info\_label.text = "Если поле IP пустое — мы запустим сервер.\nЕсли указать IP, подключаемся как клиент."

port\_field.text = "7777"

# 2) Подключаем сигналы

start\_button.connect("pressed", Callable(self, "\_on\_StartButton\_pressed"))

back\_button.connect("pressed", Callable(self, "\_on\_BackButton\_pressed"))

# 3) Сетевые сигналы

var mp = get\_tree().get\_multiplayer()

mp.peer\_connected.connect(\_on\_peer\_connected)

mp.connection\_failed.connect(\_on\_connection\_failed)

mp.server\_disconnected.connect(\_on\_server\_disconnected)

mp.peer\_disconnected.connect(\_on\_peer\_disconnected)

# 4) Пока не нажали «Start», все поля активны:

ip\_field.editable = true

port\_field.editable = true

start\_button.disabled = false

back\_button.disabled = false

func \_on\_StartButton\_pressed() -> void:

var port: int = int(port\_field.text)

var typed\_ip: String = ip\_field.text.strip\_edges()

if typed\_ip == "":

# ====== Старт сервера ======

var peer := ENetMultiplayerPeer.new()

var err := peer.create\_server(port, 2)

if err != OK:

status\_label.text = "Не удалось запустить сервер на порту %d." % port

return

get\_tree().get\_multiplayer().multiplayer\_peer = peer

status\_label.text = "Сервер запущен на порту %d.\nЖдём подключения клиента..." % port

# Блокируем всё:

ip\_field.editable = false

port\_field.editable = false

start\_button.disabled = true

back\_button.disabled = true

else:

# ====== Старт клиента ======

var peer := ENetMultiplayerPeer.new()

var err := peer.create\_client(typed\_ip, port)

if err != OK:

status\_label.text = "Не удалось подключиться к %s:%d." % [typed\_ip, port]

return

get\_tree().get\_multiplayer().multiplayer\_peer = peer

status\_label.text = "Подключаемся к %s:%d..." % [typed\_ip, port]

# Блокируем всё:

ip\_field.editable = false

port\_field.editable = false

start\_button.disabled = true

back\_button.disabled = true

func \_on\_BackButton\_pressed() -> void:

# Возвращаемся в MainMenu

get\_tree().change\_scene\_to\_file("res://MainMenu.tscn")

func \_on\_peer\_connected(id: int) -> void:

# Когда peer действительно подключился (у хоста: клиент, у клиента: он сам).

status\_label.text = "Соединение установлено (peer id=%d). Переход в игру..." % id

await get\_tree().create\_timer(0.5).timeout

get\_tree().change\_scene\_to\_file("res://Game.tscn")

func \_on\_connection\_failed() -> void:

status\_label.text = "Ошибка подключения."

ip\_field.editable = true

port\_field.editable = true

start\_button.disabled = false

back\_button.disabled = false

func \_on\_server\_disconnected() -> void:

status\_label.text = "Сервер разорвал соединение."

ip\_field.editable = true

port\_field.editable = true

start\_button.disabled = false

back\_button.disabled = false

func \_on\_peer\_disconnected(id: int) -> void:

status\_label.text = "Игрок %d отключился." % id

“Mainmenu.gd”

# res://MainMenu.gd

extends Control

func \_ready() -> void:

$VBoxContainer/PlayBotButton.connect("pressed", Callable(self, "\_on\_PlayBotButton\_pressed"))

$VBoxContainer/PlayOnlineButton.connect("pressed", Callable(self, "\_on\_PlayOnlineButton\_pressed"))

$VBoxContainer/SettingsButton.connect("pressed", Callable(self, "\_on\_SettingsButton\_pressed"))

$VBoxContainer/ExitButton.connect("pressed", Callable(self, "\_on\_ExitButton\_pressed"))

if has\_node("MusicPlayer"):

$MusicPlayer.play()

func \_on\_PlayBotButton\_pressed() -> void:

# В этом случае мы просто идём в Game.tscn, и там играем против бота (если у вас есть алгоритм бота)

get\_tree().change\_scene\_to\_file("res://Game.tscn")

func \_on\_PlayOnlineButton\_pressed() -> void:

# Переходим в Lobby

get\_tree().change\_scene\_to\_file("res://Lobby.tscn")

func \_on\_SettingsButton\_pressed() -> void:

get\_tree().change\_scene\_to\_file("res://Settings.tscn")

func \_on\_ExitButton\_pressed() -> void:

get\_tree().quit()

“mainmenuonline.gd”

extends Control

func \_ready() -> void:

$VBoxContainer/HostButton.connect("pressed", Callable(self, "\_on\_HostButton\_pressed"))

$VBoxContainer/JoinButton.connect("pressed", Callable(self, "\_on\_JoinButton\_pressed"))

func \_on\_HostButton\_pressed() -> void:

get\_tree().change\_scene\_to\_file("res://Lobby.tscn")

func \_on\_JoinButton\_pressed() -> void:

get\_tree().change\_scene\_to\_file("res://Lobby.tscn")

“Pausemenu.gd”

extends CanvasLayer

func \_ready():

hide() # Скрываем меню при старте

# Подключаем кнопки: обратите внимание на путь

$ColorRect/VBoxContainer/ResumeButton.connect("pressed", Callable(self, "\_on\_resume\_button\_pressed"))

$ColorRect/VBoxContainer/MainMenuButton.connect("pressed", Callable(self, "\_on\_main\_menu\_button\_pressed"))

$ColorRect/VBoxContainer/QuitButton.connect("pressed", Callable(self, "\_on\_quit\_button\_pressed"))

func \_on\_resume\_button\_pressed() -> void:

hide()

get\_tree().paused = false

func \_on\_main\_menu\_button\_pressed() -> void:

get\_tree().paused = false

get\_tree().change\_scene\_to\_file("res://MainMenu.tscn")

func \_on\_quit\_button\_pressed() -> void:

get\_tree().quit()

“Settings.gd”

# res://Settings.gd

extends Control

@onready var volume\_slider: HSlider = $VBoxContainer/VolumeSlider

func \_ready() -> void:

var bus\_idx = AudioServer.get\_bus\_index("Master")

volume\_slider.value = db\_to\_linear(AudioServer.get\_bus\_volume\_db(bus\_idx))

volume\_slider.connect("value\_changed", Callable(self, "\_on\_VolumeSlider\_value\_changed"))

$VBoxContainer/BackButton.connect("pressed", Callable(self, "\_on\_BackButton\_pressed"))

if has\_node("MusicPlayer"):

$MusicPlayer.play()

func \_on\_VolumeSlider\_value\_changed(val: float) -> void:

var bus\_idx = AudioServer.get\_bus\_index("Master")

AudioServer.set\_bus\_volume\_db(bus\_idx, linear\_to\_db(val))

func \_on\_BackButton\_pressed() -> void:

get\_tree().change\_scene\_to\_file("res://mainmenu.tscn")

“Ship.gd”

# res://Ship.gd

extends Node2D

@export var length: int = 1 # длина корабля (1..4)

var orientation: int = 0 # 0 = горизонтально, 1 = вертикально

func \_ready() -> void:

var texture\_path = "res://sprites/ship\_%d.png" % length

if ResourceLoader.exists(texture\_path):

$Sprite2D.texture = load(texture\_path)

if orientation == 1:

$Sprite2D.rotation\_degrees = 90

else:

push\_error("Ship.gd: Не удалось загрузить спрайт: %s" % texture\_path)

“ShipPlacement.gd”

# res://ShipPlacement.gd

extends Node

var ships = [

{ "size": 4, "count": 1 },

{ "size": 3, "count": 2 },

{ "size": 2, "count": 3 },

{ "size": 1, "count": 4 }

]

func initialize\_grid() -> Array:

var grid: Array = []

for y in range(10):

var row: Array = []

for x in range(10):

row.append(0)

grid.append(row)

return grid

func can\_place\_ship(grid: Array, x: int, y: int, size: int, direction: int) -> bool:

for i in range(size):

var dx = x + (i if direction == 0 else 0)

var dy = y + (i if direction == 1 else 0)

if dx < 0 or dx >= 10 or dy < 0 or dy >= 10:

return false

if grid[dy][dx] != 0:

return false

for i in range(size):

var dx = x + (i if direction == 0 else 0)

var dy = y + (i if direction == 1 else 0)

for ny in range(dy - 1, dy + 2):

for nx in range(dx - 1, dx + 2):

if nx >= 0 and nx < 10 and ny >= 0 and ny < 10:

if grid[ny][nx] == 1:

return false

return true

func place\_ship(grid: Array, x: int, y: int, size: int, direction: int) -> void:

for i in range(size):

var dx = x + (i if direction == 0 else 0)

var dy = y + (i if direction == 1 else 0)

grid[dy][dx] = 1

func place\_all\_ships\_randomly(grid: Array) -> void:

for ship in ships:

var to\_place = ship["count"]

var size = ship["size"]

var placed = 0

while placed < to\_place:

var x = randi() % 10

var y = randi() % 10

var dir = randi() % 2

if can\_place\_ship(grid, x, y, size, dir):

place\_ship(grid, x, y, size, dir)

placed += 1