

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД
«ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Навчально-науковий інститут фізики та інформаційних технологій

Кафедра інформаційних технологій та систем

Соколов Дмитро Ігорович

**ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОЇ ВЕРСІЇ ГРИ
«МОРСЬКИЙ БІЙ» ДЛЯ ПЛАТФОРМИ ANDROID З ВИКОРИСТАННЯМ
КРОСПЛАТФОРМНОГО ФРЕЙМВОРКУ FLUTTER**

Бакалаврська робота
за напрямом підготовки 121 Інженерія програмного забезпечення

Особистий підпис – _____ Дмитро Соколов

Науковий керівник – _____ кандидат педагогічних наук, доцент
(підпис) М.А. Семенов
(посада, науковий ступінь,
наукове звання, ініціали, прізвище)

Зав. кафедри – _____ зав. кафедри ІТС, кандидат
(підпис) педагогічних наук, доцент,
М.А. Семенов
(посада, науковий ступінь,
наукове звання, ініціали, прізвище)

Полтава – 2025

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”

Факультет (інститут)

Навчально-науковий інститут фізики та
інформаційних технологій

Кафедра, циклова комісія

Інформаційних технологій та систем

Освітній ступень

Бакалавр

Напрямок підготовки
(спеціальність)

121 «Інженерія програмного
забезпечення»

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

М.А. Семенов

(підпис)

(ініціали, прізвище)

“ ” 2025 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Соколов Дмитро Ігорович

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи)

Проектування та розробка
мобільної версії гри

“Морський бій” для платформи Android з використанням кросплатформного
фреймворку flutter

Керівник кваліфікаційної роботи

Семенов М.А.

(прізвище, ім'я, по батькові, науковий
ступінь, вчене звання)

затверджена наказом по університету

2. Строк подання студентом проєкту (роботи)

3. Вихідні дані до роботи (проєкту)

У результаті виконання

роботи була розроблена двомірна мобільна гра «Морський бій» для платформ
Android та iOS із використанням кросплатформного фреймворку Flutter

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно
розробити) Огляд класичних правил гри «Морський бій»

Адаптація гри до мобільних платформ.

Аналіз вимог користувачів до мобільних ігор

Оцінка кросплатформених фреймворків для розробки гри

Вибір технологій для розробки (Flutter, Dart)

Порівняння інструментів розробки для Android та iOS

Розробка ігрової логіки

Створення архітектури застосунку за моделлю MVVM

Проектування інтерфейсу користувача

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Діаграма взаємодії компонентів, життєвий цикл гри, поле під ураженням, відображення перемоги гравця, інтерфейс користувача

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання „_____” _____ 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з / п	Назва етапів дипломного проєкту (роботи)	Строк викона ння етапів проєкту (роботи)	Примітка
	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 15 жовтня	
	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	Другий тиждень листопа да (10 листопа да)	
	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 15 грудня	
	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 28 січня	
	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	Перший тиждень березня	
	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 31 березня	
	Попередній захист роботи на кафедрі	квітень	
	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державн ої атестаці ї	

	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	
--	--	----------------------------------	--

Студент

підпис

Д. І. Соколов

(ініціали, прізвище)

Керівник

проекту (роботи)

М.А. Семенов

АНОТАЦІЯ

Соколов Д. І.

Тема: Проєктування та розробка мобільної версії гри «морський бій» для платформи android з використанням кросплатформного фреймворку flutter

Спеціальність: 121 "Інженерія програмного забезпечення"

Установа: ЛНУ імені Тараса Шевченка, 2021 р.

Бакалаврська робота містить: 69 с., 11 рис., 3 табл., 1 додат., 21 джерел.

Об'єкт дослідження: мобільні ігри, які стали лідерами на ринку розваг, витіснивши за популярністю ігри для персональних комп'ютерів і консолей.

Предмет дослідження: процес створення мобільної гри з урахуванням постійних вимог користувачів до нововведень і якості геймплею, розробленої з використанням кросплатформного фреймворку Flutter [1] для операційних систем Android та IOS.

Мета роботи - розробка двомірної мобільної гри "Морський бій" з використанням сучасного кросплатформного фреймворку Flutter [1].

Результати роботи. У роботі проведено аналіз статистичних даних щодо розвитку мобільних ігор [4], виконано проєктування та реалізацію двомірної гри "Морський бій" для мобільних платформ Android та iOS із використанням Flutter [1].

Ключові слова. МОРСЬКИЙ БІЙ, МОБІЛЬНА ГРА, FLUTTER, DART, КРОСПЛАТФОРМНИЙ, ШТУЧНИЙ ІНТЕЛЕКТ, ІГРОВА ЛОГІКА.

ABSTRACT

Sokolov D. I.

Tema: Development of a mobile game "sea battle" using the cross-platform flutter framework

Specialty: 121 "Software Engineering"

Institution: Luhansk Taras Shevchenko National University, 2021

The bachelor's thesis includes: 69 pages, 11 figures, 3 tables, 1 appendix, 21 references..

Object of Study: Mobile games that have become leaders in the entertainment market, surpassing in popularity games for personal computers and consoles..

Subject of Study: The process of creating a mobile game, taking into account the constant demands of users for innovations and gameplay quality, developed using the cross-platform framework Flutter [1] for Android and iOS operating systems.

Objective of the Work: To develop a 2D mobile game "Sea Battle" using the modern cross-platform framework Flutter [1].

Results of the Work: The work analyzes statistical data on the development of mobile games, and involves the design and implementation of the 2D game "Sea Battle" for mobile platforms Android and iOS using Flutter [1].

Keywords: SEA BATTLE, MOBILE GAME, FLUTTER, DART, CROSS-PLATFORM, ARTIFICIAL INTELLIGENCE, GAME LOGIC.

ІТС.ПІ4.0721-01-ВІІ
ВІДОМІСТЬ ПРОЕКТУ.ПРОЕКТУВАННЯ ТА РОЗРОБКА
МОБІЛЬНОЇ ВЕРСІЇ ГРИ «МОРСЬКИЙ БІЙ» ДЛЯ ПЛАТФОРМИ
ANDROID З ВИКОРИСТАННЯМ КРОСПЛАТФОРМНОГО
ФРЕЙМВОРКУ FLUTTER.

Позначення	Найменування	Кількість прим/стор	Місцезнаходження / Примітка
	Документація проекту		
ІТС.ПІ4.0721-02-ТЗ	Розробка мобільної гри	1/6	Формат А4
	“Морський бій” з		
	використанням фреймворку		
	Flutter		
	Технічне завдання.		
ІТС.ПІ4. 0721-03-ПЗ	Розробка мобільної гри	1/56	Формат А4
	“Морський бій” з		
	використанням фреймворку		
	Flutter		
	Пояснювальна записка.		
ІТС.ПІ4.0721-04-КК	Розробка мобільної гри	1/6	Формат А4
	“Морський бій” з		
	використанням фреймворку		
	Flutter		
	Microsoft Visual Studio		
	Керівництво користувача.		
ІТС.ПІ4.0721-05-МТ	Розробка мобільної гри	1/4	Формат А4
	“Морський бій” з		
	використанням фреймворку		
	Flutter		
	Програма та методика		
	тестування.		

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут фізики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

М.А. Семенов

(підпис)

(ініціали, прізвище)

“ ” 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання програмної розробки (ПР):

"ПРОЕКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОЇ ВЕРСІЇ ГРИ
«МОРСЬКИЙ БІЙ» ДЛЯ ПЛАТФОРМИ ANDROID З
ВИКОРИСТАННЯМ КРОСПЛАТФОРМНОГО ФРЕЙМВОРКУ
FLUTTER"

ІТС.ПЗ4.0721-02-ТЗ

ПОГОДЖЕНО

Керівник кваліфікаційної роботи

Семенов М.А.

“ ” 2025р

ВИКОНАВЕЦЬ

Студент групи 4ПЗ

Соколов Д. І.

“ ” 2025р

Полтава 2025

ЗМІСТ

ВСТУП	3
1. ХАРАКТЕРИСТИКА ОБ'ЄКТА	3
2. ПРИЗНАЧЕННЯ ТОВАРІВ	3
3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ.....	3
4. ТЕХНІКО - ЕКОНОМІЧНІ ВИМОГ ДО КІНЦЕВОГО ПРОДУКТУ	4
5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ.....	4
6. ЕТАПИ ВИКОНАННЯ ПР	5
7. ПРИЙОМ.....	5
8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО.....	6

ВСТУП

1.1 Найменування: Мобільна гра “Морський Бій”.

1.2 Шифр ПР: АДР -1

1.3 Підстава для виконання ПР: Підставою для виконання даної розробки є заява від замовника.

1.4 Терміни розробки:

1.4.1 Початок 30 жовтня 2025 р.

1.4.2 Закінчення 20 квітня 2025 р.

1.5 Фінансується за рахунок коштів замовника. Умови фінансування - за договором № 13 / а і протоколу узгодження ціни № 13 / б.

1. ХАРАКТЕРИСТИКА ОБ'ЄКТА

1.1. Розробляема гра повина мати розважальний характер. **До складу об'єкту**, що створюється повинно входити:

1.1.1. Графічний додаток, що створюється.

1.2. **До вхідної інформації** належать вимоги замовника щодо додатку.

2. ПРИЗНАЧЕННЯ ТОВАРІВ

2.1. **Призначення:** розважальна гра.

2.2. **Основні критерії ефективності**

2.2.1. Зручний інтерфейс.

2.2.1.1. Гравець повинен мати можливість виставляти свої корблі та атакувати ворожі;

2.2.1.2. Керування повинно бути інтуїтивним.

3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Загальні вимоги

3.1.1. Графічний додаток працює на смартфонах на базі ОС Android версії 4.4 або вище;

3.1.2. Вимоги до апаратного забезпечення смартфона - не передбачені і можуть встановлюватися розробником програмного комплексу;

3.1.3. Графічна гра повина мати зручний інтерфейс.

3.2. Додаткові вимоги

3.2.1. Мова програмування Dart, фреймворк LibGDX.

3.2.2. Вимоги до ліцензійного ПЗ не передбачаються і вирішуються замовником

3.3. Вимоги до складу і архітектури

3.3.1. Розробник самостійно вибирає склад і виконує розробку архітектури ПР

3.3.2. Особливих умов до складу та архітектури ПР не передбачено

3.4. Вимоги до якості і надійності

3.4.1. Графічний додаток повинен надійно працювати.

3.4.2. Розробник обирає технічні характеристики персонального комп'ютера, смартфона, налаштовує системне програмне забезпечення.

3.4.3. Розробник гарантує роботу графічного додатку без збоїв та переналаштувань.

3.5. Вимоги до експлуатації

3.5.1. Розробник використовує смартфон, на якому графічний додаток повинен надійно працювати.

4. ТЕХНІКО - ЕКОНОМІЧНІ ВИМОГ ДО КІНЦЕВОГО ПРОДУКТУ

Вартість робіт по розробці даної ПР визначається згідно з договором на розробку. Вартість пропонувананих аналогів повинна забезпечити економічну доцільність їх застосування.

5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ

5.1. Вимоги до екологічної безпеки при експлуатації.

Не пред'являються.

5.2. Спеціальні вимоги до кінцевого продукту.

Не пред'являються.

5.3. Вимоги до безпеки для населення при експлуатації продукції.

Не пред'являються.

6. ЕТАПИ ВИКОНАННЯ ПР

Етапи виконання ПР можуть уточнюватися згідно календарного плану робіт за погодженням між замовником і виконавцем

№	Етапи виконання роботи	Термін виконання і обсяг робіт	Звітні матеріали
1	Аналіз розробки програмного комплексу та розробка першої версії. Аналіз вимог. Розробка структури. Попереднє тестування.		Фрагмент програмного комплексу на ЕОМ замовника, який виконує всі основні функції і звітна документація п.8.2
2	Коригування структури. Розробка допоміжних функцій. Розробка остаточної версії програмного комплексу і його обробки. Тестування.		Готовий програмний комплекс на ЕОМ замовника і звітна документація п.8.2
3	Доопрацювання окремих модулів і навчання користувачів. Розробка звітних матеріалів по п.8 цього ТЗ		Звітні матеріали згідно з пунктом 8.

7. ПРИЙОМ

7.1. Необхідні вимоги для впровадження ПР і завершення робіт.

Оцінка результатів розробки і доцільність її продовження здійснюється замовником за поданням наступних матеріалів:

- встановлено програмний комплекс на ЕОМ замовника;
- перелік файлів на резервному носії;
- короткий опис роботи ПР і опис всіх файлів, які необхідні для роботи

ПР.

- перелік документів
- Технічне завдання
- Пояснювальна записка
- Програма і методика тестування
- Керівництво користувача

7.2. Перелік звітних документів, необхідних для прийняття етапів роботи:

- короткий опис результатів етапу у вигляді анотованого звіту (для 1 та 2 етапів);

- частковий програмний комплекс на ЕОМ замовника згідно календарного плану робіт;

- акт приймання продукції.

Звітні матеріали подаються у вигляді звітів на папері по ГОСТ 7.32-91

7.3. Загальний перелік до прийому звітних документів, макетів, експериментальних зразків.

До прийому пред'являються: акт здачі-приймання продукції, акт впровадження ПР.

7.4. Тестування ПР

Тестування виконується в "Програми і методики тестування", яка розробляється виконавцем і затверджується замовником

8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЗ «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Навчально-науковий інститут математики та інформаційних технологій
(назва факультету, інституту)
Кафедра інформаційних технологій та систем
(назва кафедри)

Пояснювальна записка до кваліфікаційної роботи
за першим (бакалаврським) рівнем освіти

на тему:

Проектування та розробка мобільної версії гри «морський бій» для платформи android з використанням кросплатформного фреймворку flutter

Виконав: здобувач вищої освіти 4 курсу спеціальності
F2 «Інженерія програмного забезпечення»
(шифр і назва напрямку підготовки, спеціальності)

_____ Дмитро СОКОЛОВ
(прізвище та ініціали)

Керівник _____ Микола СЕМЕНОВ
(прізвище та ініціали)

Рецензент _____ Юрій КОЗУБ
(прізвище та ініціали)

Полтава – 2025

ЗМІСТ

ВСТУП	5
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	8
1 АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1 Постановка задачі	9
1.2 Огляд вимог до програмного забезпечення.....	10
1.2.1 Функціональні вимоги	10
1.2.2 Нефункціональні вимоги	11
1.3 Розгляд аналогів.....	12
1.3.1 Sea Battle 2 (BYRIL).....	12
1.3.2 Fleet Battle (Smuttlewerk Interactive).....	13
1.3.3 Battleship (Marmalade Game Studio)	14
1.3.4 Порівняльний аналіз та висновки	16
1.4 Вибір моделі розробки.....	17
1.5 Вибір технологій та інструментів	20
1.5.1 Мова програмування.....	20
1.5.2 Огляд Flutter	22
1.6 Середовище розробки Android Studio	25
1.7 Висновки до першого розділу	28
2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ.....	31
2.1 Архітектура системи.....	31
2.1.1 Обґрунтування вибору архітектурного підходу.....	31
2.1.2 Компонентна структура застосунку.....	31
Таблиця 1	32
2.1.3 Взаємодія основних модулів	32
2.1.4 Життєвий цикл гри.....	34
2.2 Проєктування логіки гри	34
2.2.1 Представлення ігрового поля та кораблів	35
2.2.2 Реалізація класичних правил гри	35
2.2.3 Механізми ручної та автоматичної розстановки кораблів	36
2.2.4 Логіка ходів гравця та комп'ютерного супротивника	36
2.2.5 Алгоритми перевірки стану гри	37
2.3 Модель штучного інтелекту супротивника	37
2.3.1 Вимоги до комп'ютерного супротивника.....	37
2.3.2 Модель поведінки (рівні складності: легкий, середній, складний) ..	37
2.3.3 Алгоритми атаки: випадковий вибір, стратегія “пошуку та добивання”	38

2.3.4 Оптимізація продуктивності та часу обробки ходів	38
2.4 Проєктування інтерфейсу користувача	39
2.5 Структура даних і збереження стану гри.....	40
2.5.1 Представлення ігрового поля	41
2.5.2 Структура кораблів	41
2.5.3 Структура ходу гри	42
2.5.4 Збереження стану гри	42
2.5.5 Потенційне розширення збереження гри.....	43
2.6 Алгоритм обробки дій користувача	44
2.6.1 Основні типи дій користувача.....	44
2.6.2 Алгоритм обробки подій	44
2.6.3 Реактивна модель взаємодії.....	45
2.6.4 Валідація і обробка помилок	45
2.6 Сценарії використання (Use Case Diagram).....	46
2.6.1 Основні сценарії взаємодії користувача з додатком	46
2.6.2 Використання варіантів	47
2.6.3 Послідовність дій при запуску, розстановці та грі.....	47
2.7 Висновки до розділу	48
Розділ 3. Розробка та тестування програмної системи	49
3.1 Архітектурна структура реалізації.....	49
3.2 Головна точка входу в застосунок	49
3.3 Реалізація моделі гри	51
3.3.1 Клітинка (Cell).....	51
3.3.2 Корабель (Ship)	52
3.3.3 Ігрове поле (Board).....	52
3.4 Логіка гри у ViewModel.....	53
3.4.1 Генерація кораблів	53
3.4.2 Обробка пострілу	54
3.4.3 Перевірка завершення гри	55
3.5 Інтерфейс користувача	56
3.6 Тестування програмної системи	58
3.6.1 Модульне тестування логіки гри.....	58
3.6.2 Ручне тестування інтерфейсу	58
3.7 Висновки до розділу	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
Додаток А.....	63

ВСТУП

У сучасному світі мобільні ігри стали невід’ємною частиною цифрової культури, активно розвиваючись на тлі загального зростання ринку мобільних додатків. Завдяки широкій доступності мобільних пристроїв, ігри на смартфонах охоплюють широку аудиторію користувачів різного віку та соціального статусу. Серед усього розмаїття ігрових жанрів особливе місце займають логічні та стратегічні ігри, які не тільки розважають, але й сприяють розвитку мислення, концентрації та стратегічного планування. Однією з таких ігор є «Морський бій» — класична настільна гра, що здобула популярність завдяки простим правилам і водночас високому потенціалу для змагальності.

З розвитком мобільних технологій виникає потреба у перенесенні традиційних ігор у цифровий формат, адаптований до сучасних платформ і користувацьких очікувань. Розробка мобільної версії гри «Морський бій» з використанням кросплатформеного фреймворку Flutter [1] дозволяє створити застосунок, доступний для користувачів Android та iOS без необхідності окремої реалізації для кожної платформи. Такий підхід забезпечує скорочення часу та витрат на розробку, підвищує продуктивність команди та полегшує подальшу підтримку програмного продукту.

Актуальність теми обумовлена кількома факторами. По-перше, збереження інтересу до класичних ігор у нових поколінь користувачів вимагає адаптації таких ігор до мобільного середовища. По-друге, використання сучасних технологій для створення інтуїтивно зрозумілого інтерфейсу та інтерактивного геймплею дозволяє забезпечити високу залученість гравців. По-третє, Flutter [1] як платформа забезпечує стабільну та ефективну реалізацію застосунку з можливістю масштабування, адаптації під різні пристрої та швидкого розгортання нових функцій.

Метою роботи є розробка мобільної гри «Морський бій» з режимом гри проти комп’ютерного супротивника, що реалізована з використанням технології Flutter [1] та мови програмування Dart [2]. Основний акцент зроблено на

реалізації якісного користувацького інтерфейсу, надійної ігрової логіки та адаптивного алгоритму штучного інтелекту [6].

Для досягнення поставленої мети визначено такі основні завдання:

- здійснити аналіз предметної області та функціональних особливостей гри «Морський бій»;
- сформулювати вимоги до функціоналу та архітектури програмного забезпечення;
- розробити структуру даних і модель логіки гри відповідно до класичних правил [10];
- реалізувати адаптивний інтерфейс користувача з підтримкою ручної та автоматичної розстановки кораблів;
- розробити алгоритм дій комп'ютерного гравця з урахуванням різних рівнів складності;
- провести тестування гри на мобільних пристроях з метою перевірки стабільності та коректності функціонування;
- оцінити можливості подальшого розширення функціоналу (наприклад, додавання мережевої гри).

Теоретичну основу роботи становлять методи об'єктно-орієнтованого програмування, принципи архітектури MVVM [8] та підходи до проєктування інтерфейсів у середовищі Flutter [1]. У процесі реалізації використовується мова програмування Dart [2], фреймворк Flutter та засоби автоматизованого тестування для перевірки окремих модулів системи.

Тема роботи є актуальною в контексті цифровізації ігрової культури, розвитку мобільного програмування та забезпечення доступу до якісного розважального контенту. Розробка гри «Морський бій» надає можливість поєднати класичну механіку з сучасними технологіями, створивши інноваційний програмний продукт, що відповідає вимогам ринку та очікуванням користувачів.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД	база даних
ПЗ	програмне забезпечення
ПК	персональний комп'ютер

1 АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Завданням роботи є розробка мобільної гри «Морський бій», яка дозволяє користувачам грати проти комп'ютерного супротивника. Для реалізації застосунку буде використано кросплатформену технологію Flutter [1], що дозволяє охопити аудиторію на різних пристроях під керуванням Android та iOS.

Гра має дотримуватися класичних правил [10] «Морського бою», забезпечувати можливість автоматичного та ручного розташування кораблів на ігровому полі. Ключовою є реалізація зрозумілого та приємного інтерфейсу, який забезпечить комфортну взаємодію з грою для користувачів різного віку.

На основі аналізу предметної області визначено такі задачі для реалізації проєкту:

- Розробка архітектури гри;
- Проєктування логіки гри відповідно до класичних правил [10];
- Реалізація алгоритму поведінки комп'ютерного супротивника;
- Створення зручного та адаптивного інтерфейсу користувача;
- Реалізація механізму автоматичного та ручного розташування кораблів;
- Тестування гри на різних платформах і пристроях.

Основні функції гри:

- Гра проти комп'ютерного суперника;
- Ручне та автоматичне розташування кораблів;
- Графічна індикація ходів (попадання, промахи, знищення кораблів);
- Вибір рівня складності комп'ютерного супротивника;
- Підрахунок результатів гри та статистика перемог;
- Можливість перезапуску гри після її завершення.

Предметна область залишається актуальною завдяки широкій популярності класичних настільних ігор в цифровій реалізації, що дозволяє користувачам різного віку цікаво та продуктивно проводити час. Використання

технології Flutter [1] надає можливість легко масштабувати та розширювати гру в майбутньому, наприклад, додаючи мережеві режими, рейтингові системи та інтеграцію соціальних функцій.

Таким чином, розробка кросплатформної мобільної гри «Морський бій» з єдиним режимом гри проти комп'ютера відповідає сучасним тенденціям у сфері розробки мобільних застосунків, забезпечує широку доступність та створює потенціал для подальшого розвитку проєкту.

1.2 Огляд вимог до програмного забезпечення

1.2.1 Функціональні вимоги

Функціональні вимоги визначають ключові можливості та функції системи, необхідні для забезпечення її роботи згідно із завданнями, які були поставлені у роботі.

Для гри «Морський бій» встановлено наступні функціональні вимоги:

- Гра проти комп'ютерного суперника, що передбачає можливість здійснення ходів по черзі з комп'ютерним гравцем;
- Реалізація класичних правил [10] гри «Морський бій», що включає правильне розташування кораблів, черговість ходів та визначення переможця;
- Автоматична розстановка кораблів для швидкого початку гри;
- Ручна розстановка кораблів користувачем з метою створення персоналізованої стратегії;
- Графічна індикація результатів ходів (попадання, промах, знищення корабля);
- Вибір різних рівнів складності комп'ютерного супротивника (легкий, середній, важкий);
- Ведення статистики перемоги і поразок для аналізу результатів;
- Можливість швидкого перезапуску гри після завершення для повторної гри.

1.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики гри, що забезпечують зручність, продуктивність та надійність роботи застосунку. Серед цих вимог виділено:

- Підтримка роботи застосунку на операційних системах Android (версія 8 та вище) та iOS, забезпечуючи широкий спектр потенційних користувачів;
- Висока швидкодія та стабільність роботи, що передбачає відсутність помітних затримок та збоїв у процесі гри;
- Оптимізація використання ресурсів мобільного пристрою для мінімізації споживання батареї та зручності тривалого використання гри;
- Масштабованість, яка передбачає можливість легко додавати нові функції чи режими гри без значних змін архітектури;
- Інтуїтивно зрозумілий та зручний інтерфейс користувача, який не потребує значних технічних знань для комфортного використання;
- Простота процесу оновлення, що дозволяє легко впроваджувати покращення та виправляти помилки;
- Наявність якісної документації коду, що забезпечує легкість подальшого супроводу, розвитку та масштабування проєкту.

1.3 Розгляд аналогів

У процесі розробки гри «Морський бій» доцільним є проведення аналізу програмних аналогів, представлених на ринку, з метою виявлення їх сильних і слабких сторін, технічних характеристик, а також загальних підходів до реалізації функціональності. Такий аналіз дозволяє не лише оцінити конкурентні рішення, але й сформулювати обґрунтовані технічні вимоги до власного програмного продукту.

1.3.1 Sea Battle 2 [12] (BYRIL)

Один із найвідоміших мобільних аналогів гри «Морський бій» Sea Battle 2 [12] - реалізований студією BYRIL. Програмний продукт підтримує класичні правила гри, а також передбачає декілька режимів: одиночну гру проти штучного інтелекту [6], багатокористувацьку гру через Bluetooth або Інтернет, з

можливістю ведення чату між гравцями. Графічне оформлення витримане в стилі блокнотних ілюстрацій, що формує унікальну візуальну естетику.

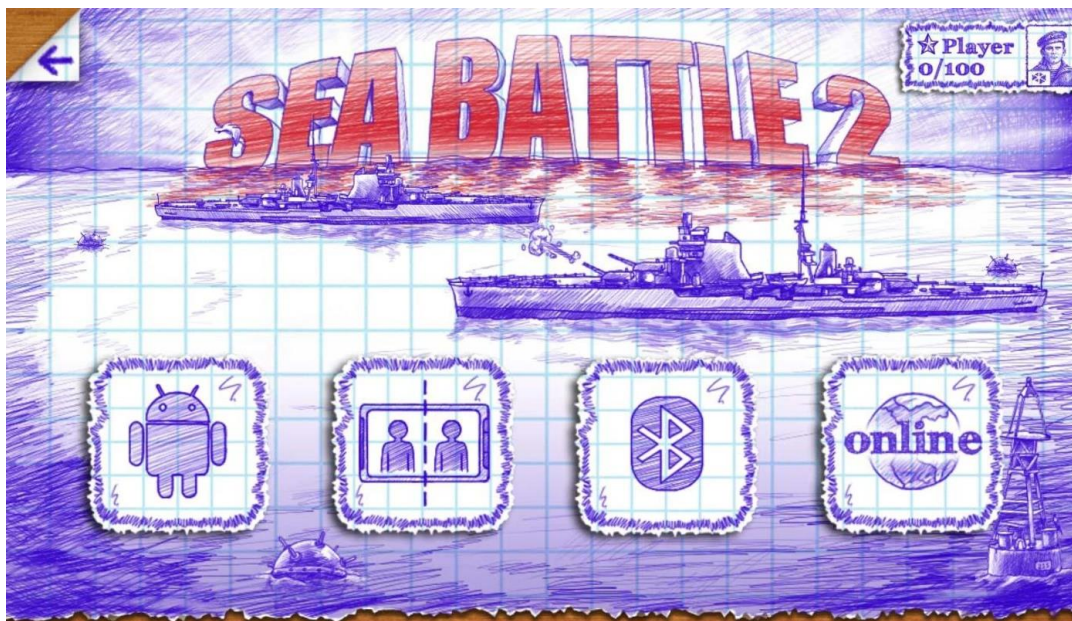


Рисунок 1.1 - Sea Battle 2 [12]

Технічні характеристики:

- Платформи: Android, iOS
- Мови програмування: Java, Kotlin (Android), Swift (iOS)
- Тип графіки: 2D
- Мережеві можливості: TCP/IP, Bluetooth, власна серверна інфраструктура
- Монетизація: реклама, внутрішні покупки

Оцінка: Серед переваг — широкий вибір режимів гри та простий, інтуїтивно зрозумілий інтерфейс. Основним недоліком є надмірна кількість реклами та спрощена логіка штучного інтелекту [6] супротивника.

1.3.2 Fleet Battle [13] (Smuttlewerk Interactive)

Гра Fleet Battle [13] орієнтована на реалізацію класичного морського бою з сучасним візуальним стилем та підтримкою мережевих функцій. Особлива увага приділяється соціальній взаємодії гравців, рейтинговим системам і персоналізації ігрового досвіду.

Технічні характеристики:

- Платформи: Android, iOS
- Технологія розробки: Unity [11] (мова програмування - C#)

- Тип графіки: 2D/3D з анімаційними ефектами
- Мережеві можливості: Unity [11] Multiplayer, REST API
- Монетизація: реклама, покупки всередині додатка

Рисунок 1.2 - Fleet Battle [13]

Перевагою є якісна графіка, кастомізація кораблів, ведення статистики. Серед недоліків - часті комерційні повідомлення та затримки під час онлайн-гри.

1.3.3 Battleship [14] (Marmalade Game Studio)

Battleship [14] - офіційна ліцензована реалізація гри за мотивами класичної настільної версії від Hasbro. Програмний продукт вирізняється високою якістю візуального оформлення та підтримкою декількох режимів, у тому числі тематичних сценаріїв гри.

Технічні характеристики:

- Платформи: Android, iOS, Nintendo Switch
- Технологія розробки: Unity [11] Engine (мова програмування - C#)
- Тип графіки: 2D з анімаціями та візуальними ефектами
- Мережеві можливості: REST API, peer-to-peer, власні сервери
- Монетизація: платна версія без реклами

Рисунок 1.3 - Battleship [14]

Оцінка: До переваг відноситься відповідність офіційним стандартам гри, естетика та мультиплатформність. Недоліком є відсутність безкоштовної базової версії, що обмежує доступ для широкої аудиторії.

1.3.4 Порівняльний аналіз та висновки

Узагальнюючи результати аналізу наведених аналогів, можна сформулювати низку висновків, що мають практичну цінність для реалізації власного проєкту:

1. Переважна більшість ігор реалізована за допомогою ігрового рушія Unity [11], що забезпечує високу продуктивність і гнучкість, проте вимагає глибокого володіння екосистемою C#. У цьому контексті використання Flutter [1] як кросплатформного інструменту з мовою Dart [2] дає змогу

досягти аналогічного рівня кросплатформності при зниженні складності початкової розробки.

2. Наявність режиму гри проти штучного інтелекту [6] є базовою функціональністю більшості реалізацій, тоді як мережеві режими, рейтинги та кастомізація розглядаються як додаткові функції. Це підтверджує доцільність початкової реалізації лише одиночного режиму з перспективою подальшого розширення функціоналу.
3. Інтерфейс та візуальний стиль значно впливають на сприйняття гри користувачем. Варто орієнтуватися на чистий, інтуїтивно зрозумілий дизайн без надмірної кількості реклами або складних елементів управління.
4. Реалізація адаптивної складності штучного інтелекту [6] може стати вагомим конкурентним перевагою. Варіативність логіки дій супротивника дозволяє урізноманітнити ігровий процес і забезпечити залучення гравців із різним рівнем підготовки.
5. Відсутність офіційних реалізацій на Flutter [1] у сегменті подібних ігор відкриває можливості для створення інноваційного кросплатформного продукту, що поєднує сучасні UI [7]-рішення та високий рівень продуктивності.

Отже, проведений аналіз аналогів дозволив ідентифікувати ключові вимоги до програмного продукту, що розробляється, визначити технічні орієнтири та сформулювати уявлення про функціональні очікування цільової аудиторії. Отримані результати будуть використані як підґрунтя для прийняття архітектурних і функціональних рішень у процесі реалізації гри «Морський бій».

1.4 Вибір моделі розробки

У процесі вибору моделі розробки програмного забезпечення для реалізації гри «Морський бій» було проведено аналіз сучасних методологій та визначено найбільш доцільну для специфіки даного проєкту. Зважаючи на особливості розробки ігрових додатків, зокрема високу динаміку змін вимог,

необхідність частого тестування та постійної взаємодії з кінцевим користувачем, оптимальним рішенням є застосування гнучкої моделі розробки програмного забезпечення (Agile [5]).

Agile [5] – це ітеративна модель створення програмних систем, яка характеризується розбиттям процесу розробки на короткі цикли (ітерації), що дозволяють послідовно створювати функціональні компоненти продукту. Після завершення кожної ітерації команда отримує готовий функціональний модуль, придатний для тестування та оцінки. Це забезпечує можливість раннього виявлення недоліків, оперативної адаптації до змін вимог та ефективного управління ризиками.

Процес розробки за гнучкою моделлю складається з наступних основних етапів:

Планування ітерації (спринту):

На початку кожної ітерації визначаються конкретні цілі, складається перелік задач (беклог спринту), які необхідно виконати протягом цього періоду. Визначаються пріоритети завдань відповідно до поточних вимог та ресурсних можливостей проєкту.

Реалізація поставлених завдань:

В процесі виконання ітерації команда здійснює розробку окремих компонентів або функцій, які визначені в беклозі. Кожен компонент проходить стадії проєктування, написання програмного коду та попереднього внутрішнього тестування.

Інтеграція та тестування:

Реалізовані в межах ітерації функції інтегруються в основну систему та підлягають комплексному тестуванню для перевірки їх працездатності, сумісності з іншими компонентами та відповідності поставленим вимогам.

Демонстрація та зворотний зв'язок:

Наприкінці кожної ітерації здійснюється демонстрація створеного продукту замовникам, користувачам або іншим зацікавленим особам з метою

отримання зворотного зв'язку, що дозволяє швидко коригувати подальші етапи розробки відповідно до отриманих рекомендацій.

Ретроспектива:

Після завершення ітерації команда аналізує процес її виконання, виявляє сильні та слабкі сторони поточного підходу, визначає шляхи покращення робочих процесів для підвищення ефективності у майбутніх ітераціях.

До основних переваг застосування гнучкої моделі належать:

- висока адаптивність до змін у вимогах на будь-якому етапі проєкту;
- раннє виявлення та коригування помилок завдяки постійному тестуванню;
- постійна взаємодія із замовником та кінцевими користувачами, що забезпечує максимальну відповідність створюваного продукту їхнім очікуванням;
- покращене управління ризиками завдяки регулярному моніторингу та контролю процесів.

Серед недоліків, які варто враховувати, можна виділити:

- підвищені вимоги до рівня організації робочого процесу та координації роботи команди;
- складність довгострокового планування через часту зміну пріоритетів задач та гнучкість вимог;
- необхідність активної участі всіх учасників проєкту в щоденній комунікації, що може ускладнювати процес у великих командах.

Обґрунтовуючи доцільність вибору саме гнучкої моделі розробки для гри «Морський бій», слід зазначити, що специфіка ігрових додатків, які характеризуються високим ступенем інтерактивності, змінюваністю вимог користувачів та необхідністю ретельного і частого тестування, повністю відповідає основним принципам Agile [5]-методології. Вибір даного підходу дозволяє ефективно створювати якісний програмний продукт з можливістю його подальшого удосконалення та розвитку.

Таким чином, застосування гнучкої моделі розробки програмного забезпечення є обґрунтованим та відповідає сучасним науково-практичним

підходам до реалізації інтерактивних програмних систем, забезпечуючи високий рівень ефективності розробки та якості кінцевого продукту.

1.5 Вибір технологій та інструментів

1.5.1 Мова програмування

Для написання роботи було обрано мову програмування Dart [2], яка є основною мовою розробки для фреймворку Flutter [1]. Dart ідеально підходить для створення кросплатформних мобільних, десктопних та веб-додатків завдяки своїм сучасним характеристикам, високій продуктивності та легкості освоєння.

Dart [2] – це статично типізована мова програмування загального призначення, яка підтримує як об'єктно-орієнтовану, так і функціональну парадигми. Dart була створена компанією Google та вперше анонсована у 2011 році. Основною метою створення Dart було забезпечення високої продуктивності розробки сучасних веб- та мобільних застосунків з комфортним, зрозумілим та лаконічним синтаксисом.

Dart [2] є мовою з відкритим вихідним кодом та активно підтримується великою спільнотою розробників, що дозволяє легко отримувати оновлення, документацію, навчальні матеріали та бібліотеки. Dart набула особливої популярності після появи Flutter [1] – UI [7]-фреймворку від Google, який забезпечує єдину базу коду для мобільних, веб та десктопних застосунків.

Ключові характеристики мови Dart [2] включають:

Сучасний і зрозумілий синтаксис: Dart [2] має чіткий та лаконічний синтаксис, що робить код зрозумілим та легким для читання та підтримки, особливо у великих проєктах.

Статична типізація з динамічними можливостями: Dart [2] підтримує статичну типізацію, що дозволяє виявляти багато помилок ще на етапі компіляції. Водночас Dart має гнучкість, надаючи можливість використовувати динамічні типи, коли це необхідно.

Ефективне керування пам'яттю: Dart [2] має автоматичне керування пам'яттю (Garbage Collection), що значно спрощує роботу розробника та зменшує ймовірність помилок пам'яті.

Ізоляція потоків (Isolates): Dart [2] використовує концепцію ізолятів (ізольованих потоків виконання), що дозволяє писати ефективний багатопоточний код без ризику виникнення конфліктів чи блокувань.

Асинхронне програмування: Вбудована підтримка асинхронних операцій через ключові слова `async` та `await` робить код чистішим, простішим для написання та ефективнішим у виконанні асинхронних завдань.

Висока продуктивність: Dart [2]-компілятор дозволяє компілювати код як у рідний (нативний) код для мобільних і десктопних застосунків, так і в JavaScript для веб-додатків, що забезпечує високу продуктивність та швидкий запуск додатків.

Hot Reload [16] (гаряче перезавантаження): Dart [2] має підтримку гарячого перезавантаження, що дозволяє миттєво бачити результати змін в коді, значно прискорюючи розробку та тестування.

Інтеграція з Flutter [1]: Dart [2] тісно інтегрована з фреймворком Flutter, що дозволяє створювати красиві та адаптивні інтерфейси з єдиною кодовою базою для Android, iOS, веб та настільних платформ.

Мова Dart [2] дозволяє розробникам швидко створювати та розгортати ефективні додатки завдяки простоті, гнучкості та ефективності свого синтаксису. Dart особливо популярна серед команд, що використовують Flutter [1], адже саме ця мова забезпечує зручну розробку та підтримку проєктів різного масштабу та рівнів складності.

Ключові моменти, що забезпечують безпечність та ефективність коду Dart [2], включають:

Null safety (Безпека від null): Dart [2] має вбудовану систему захисту від помилок, пов'язаних із null, що запобігає виникненню помилок типу `NullPointerException`.

Суворі типізація та безпечні перетворення: Мова чітко контролює типи даних, що мінімізує помилки під час виконання програми, дозволяючи використовувати безпечні приведення типів.

Вбудовані колекції: Dart [2] має зручні колекції (List, Set, Map) з потужними методами обробки даних, що підвищують продуктивність і комфорт розробки.

Швидка компіляція та виконання коду: Компіляція Dart [2] до нативного коду або JavaScript забезпечує оптимізацію продуктивності додатків на різних платформах.

Таким чином, вибір Dart [2] для написання роботи є виправданим завдяки сучасності мови, високій ефективності розробки, можливості створення кросплатформних додатків та тісній інтеграції з Flutter [1], що дозволяє легко розвивати, підтримувати та масштабувати розроблюваний програмний продукт.

1.5.2 Огляд Flutter [1]

Для реалізації програмного продукту у роботі було обрано фреймворк Flutter [1], який є одним із найсучасніших інструментів для створення кросплатформних додатків. Flutter забезпечує швидку розробку застосунків для мобільних пристроїв (Android, iOS), веб-додатків та настільних додатків (Windows, macOS, Linux) за допомогою єдиної бази коду.

Flutter [1] - це UI [7]-фреймворк з відкритим вихідним кодом, розроблений компанією Google та вперше представлений у 2017 році. Основна ідея Flutter полягає у можливості створювати красиві, ефективні та адаптивні інтерфейси користувача, використовуючи декларативний підхід до написання UI-компонентів.

Flutter [1] забезпечує повну сумісність із сучасною мовою програмування Dart [2], що дозволяє ефективно використовувати всі переваги цієї мови у процесі розробки.

Ключові характеристики Flutter [1] включають:

- Єдина кодова база (Single codebase): Flutter [1] дозволяє використовувати один набір вихідного коду для створення додатків для різних платформ (мобільні пристрої, веб, десктоп). Це суттєво скорочує час розробки, знижує витрати на підтримку та оновлення додатків.

- Декларативний підхід до створення інтерфейсів: Flutter [1] використовує декларативну парадигму створення UI [7], яка дозволяє розробнику описати, як повинен виглядати інтерфейс, а фреймворк автоматично займається його відображенням і оновленням.
- Висока продуктивність: Flutter [1] використовує власний високопродуктивний графічний рушій (Skia [15]), що дозволяє отримати максимально плавні анімації та швидке відображення інтерфейсу з частотою до 60-120 fps. Це робить Flutter ідеальним вибором для реалізації динамічних додатків і мобільних ігор [4].
- Гаряче перезавантаження (Hot Reload [16]): Ця функція дозволяє бачити результат змін в коді миттєво, без необхідності повної компіляції або перезапуску додатку. Це суттєво прискорює процес розробки та відлагодження.
- Широкий набір готових віджетів: Flutter [1] пропонує багатий набір стандартних віджетів (Material Design для Android та Cupertino для iOS), що значно спрощує створення користувацьких інтерфейсів і робить розробку додатків зручною і швидкою.
- Підтримка кастомних віджетів: Розробники можуть легко створювати власні віджети, налаштовуючи інтерфейс під будь-які потреби, що дозволяє розробляти додатки з унікальним дизайном.
- Глибока інтеграція з Dart [2]: Flutter [1] використовує Dart як основну мову програмування, що забезпечує тісну інтеграцію з її функціональністю, такою як асинхронність, ізоляція потоків, типізація та безпека.
- Висока адаптивність та масштабованість: Додатки на Flutter [1] легко адаптуються до різних розмірів екрану, пристроїв та операційних систем. Це дозволяє створювати універсальні інтерфейси з мінімальними витратами часу.
- Велика та активна спільнота розробників: Flutter [1] має значну спільноту розробників, що дозволяє швидко отримати допомогу, знайти рішення для різноманітних проблем, а також отримати доступ до багатой

екосистеми бібліотек та розширень.

Flutter [1] активно розвивається, отримує регулярні оновлення, покращення продуктивності, додавання нових можливостей, завдяки чому стає все більш привабливим та перспективним для розробки додатків будь-якої складності.

Завдяки цим перевагам Flutter [1] широко застосовується для розробки:

- мобільних додатків (iOS та Android);
- веб-додатків (з використанням Flutter [1] for Web);
- десктопних додатків (Windows, Linux, macOS);
- вбудованих систем і IoT-рішень.

Особливості Flutter [1], які роблять його відмінним вибором для створення якісного та ефективного програмного продукту:

- Покращений досвід користувача (UX): Flutter [1] дозволяє створювати плавні та швидкі інтерфейси, які надають користувачам комфортний та позитивний досвід взаємодії з додатком.
- Прискорення розробки: Використання одного набору коду для кількох платформ значно скорочує час розробки, дозволяючи швидко запускати продукт та перевіряти гіпотези.
- Економія ресурсів: Єдина база коду знижує витрати на підтримку, тестування та оновлення додатків, роблячи процес більш економічним.
- Гнучкість і персоналізація: Flutter [1] дає можливість повністю контролювати кожен елемент інтерфейсу та легко адаптувати дизайн під індивідуальні потреби проєкту.

Таким чином, вибір фреймворку Flutter [1] для розробки гри «Морський бій» є доцільним та обґрунтованим завдяки його високій продуктивності, гнучкості, масштабованості та зручності у створенні ефективних кросплатформних додатків. Використання Flutter дозволить забезпечити якісне виконання завдання роботи та створити продукт, який легко підтримуватиметься та розвиватиметься в майбутньому.

1.6 Середовище розробки Android Studio [3]

Android Studio [3] є офіційним інтегрованим середовищем розробки (IDE) для створення додатків під платформу Android, розробленим компанією Google. Базуючись на потужній платформі IntelliJ IDEA від JetBrains, Android Studio надає розробникам широкий спектр інструментів для ефективного створення, налагодження та оптимізації мобільних додатків. Основними мовами програмування, що підтримуються в Android Studio, є Java та Kotlin, що дозволяє розробникам вибирати найбільш зручну та сучасну мову для реалізації функціональних можливостей додатків.

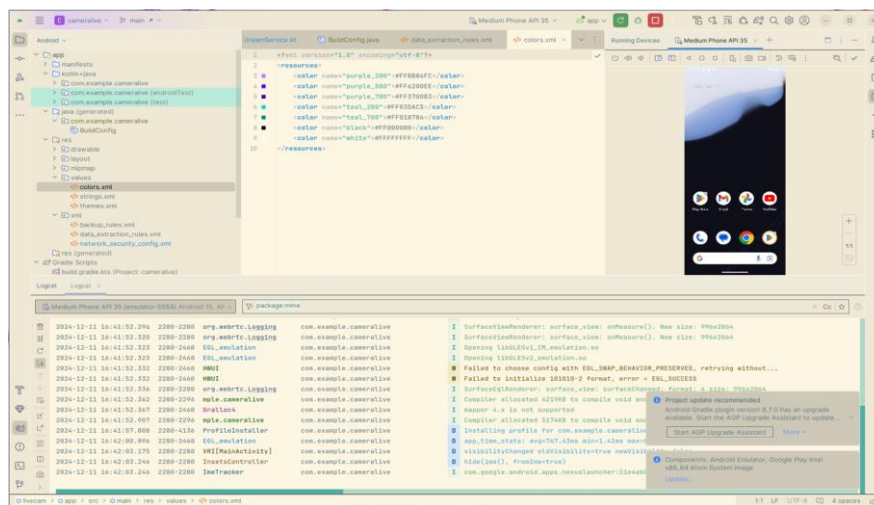


Рисунок 1.4 - Головне вікно Android Studio [3]

Однією з ключових особливостей Android Studio [3] є інтеграція з системою автоматизованої збірки Gradle, яка спрощує керування залежностями проєкту та налаштування процесу збірки. Gradle дозволяє автоматизувати різні етапи розробки, включаючи компіляцію, тестування та розгортання додатків, що значно підвищує продуктивність розробників та забезпечує гнучкість у конфігуруванні проєктів для різних середовищ та варіантів використання.

Вбудовані емулятори Android Studio [3] надають можливість тестування додатків на різних пристроях та версіях операційної системи Android без необхідності фізичного обладнання. Це забезпечує розробникам змогу швидко перевіряти функціональність додатків у різних умовах, що сприяє виявленню та усуненню помилок на ранніх етапах розробки. Крім того, Android Studio

підтримує інтеграцію з популярними системами контролю версій, такими як Git, що полегшує співпрацю в команді та управління змінами в коді проєкту.

Редактор коду в Android Studio [3] пропонує інтуїтивно зрозумілий інтерфейс з підсвічуванням синтаксису, автозаповненням та можливостями рефакторингу, що сприяє написанню чистого та оптимізованого коду. Інструменти для налагодження, включаючи дебаггер та профайлер, дозволяють розробникам детально аналізувати продуктивність додатків, виявляти та виправляти помилки, а також оптимізувати використання ресурсів пристрою.[6]

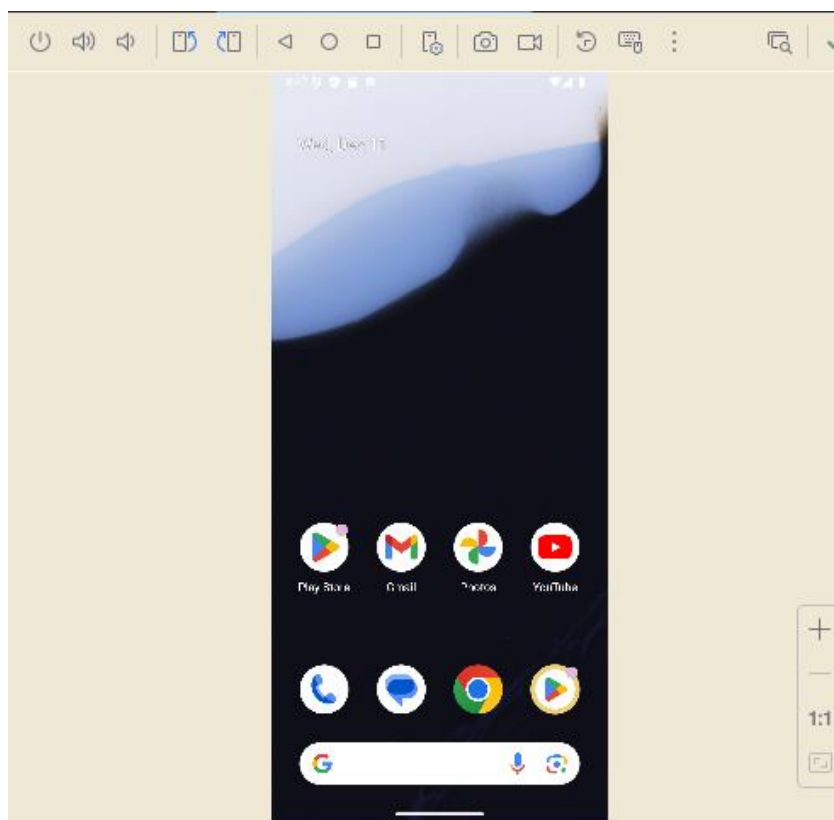


Рисунок 1.5 - Вікно емулятора Android пристрою

Особливою перевагою Android Studio [3] є дизайнер інтерфейсу користувача, який дозволяє створювати та редагувати макети додатків за допомогою візуального редактора з функцією перетягування елементів. Це значно спрощує процес дизайну, дозволяючи розробникам швидко створювати адаптивні та привабливі інтерфейси, що відповідають сучасним стандартам користувацького досвіду.

У контексті розробки системи відеотрансляції з використанням WebRTC, Android Studio [3] забезпечує необхідні інструменти для інтеграції медіа-даних,

оптимізації мережевих з'єднань та забезпечення стабільної роботи додатку на різних пристроях. Завдяки підтримці WebRTC API та можливості використання нативних бібліотек, розробники можуть реалізовувати ефективні алгоритми передачі та обробки відео- та аудіопотоків, що забезпечує високу якість комунікації та мінімальні затримки.

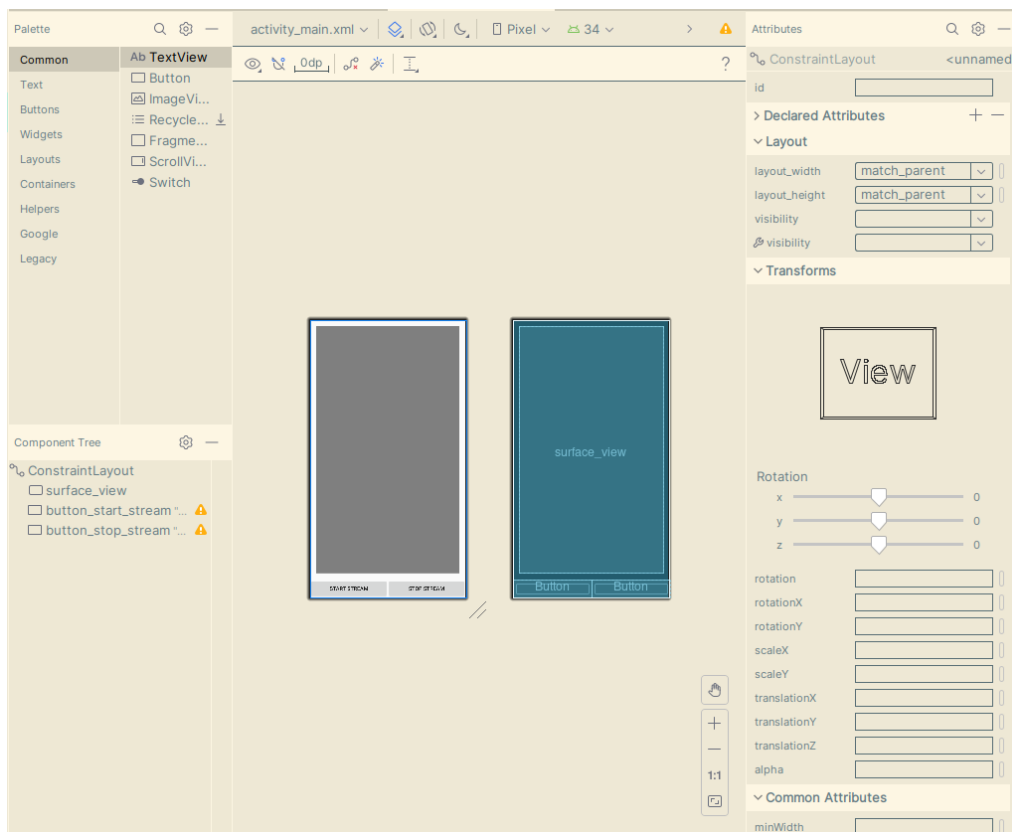


Рисунок 1.7 - Вікно дизайнер інтерфейсу.

1.7 Висновки до першого розділу

У результаті проведеного аналізу в межах першого розділу роботи було визначено концептуальні, функціональні, технологічні та методологічні основи для реалізації мобільної гри «Морський бій». Узагальнені результати цього етапу дозволяють зробити низку висновків, що обґрунтовують прийняті технічні рішення та стратегічні напрямки реалізації програмного продукту.

Передусім, було чітко сформульовано мету та завдання роботи - створення кросплатформної мобільної гри з єдиним режимом взаємодії з комп'ютерним супротивником відповідно до класичних правил [10] гри «Морський бій». Така постановка задачі дозволяє зосередити зусилля на реалізації базової, але

повноцінної функціональності, забезпечуючи при цьому потенціал для майбутнього розвитку програмного продукту - зокрема впровадження мережових режимів, рейтингової системи, збереження прогресу та мультиплеєрної взаємодії.

В межах підрозділу 1.2 сформульовано функціональні та нефункціональні вимоги до програмного забезпечення. Функціональні вимоги охоплюють усі ключові аспекти ігрового процесу — починаючи з логіки обміну ходами між гравцем і супротивником, закінчуючи відображенням результатів та підрахунком статистики. Нефункціональні вимоги визначають якісні характеристики гри, серед яких особливо важливими є кросплатформність, висока продуктивність, енергоефективність, масштабованість архітектури, а також простота використання інтерфейсу.

Проведений у підрозділі 1.3 аналіз існуючих аналогів мобільних ігор [4] дозволив виявити загальні тенденції, технічні підходи та найбільш поширені функціональні можливості. Було встановлено, що більшість рішень реалізовані з використанням рушія Unity [11], орієнтовані на онлайн-гру, та мають розширені функції кастомізації. Водночас відсутність реалізацій на базі Flutter [1] у даному сегменті ринку відкриває можливість створити інноваційний продукт, заснований на сучасних принципах UI [7]/UX-дизайну, з підтримкою єдиної кодової бази для різних платформ.

У підрозділі 1.4 було обґрунтовано вибір гнучкої моделі розробки (Agile [5]), яка, на відміну від каскадних та інкрементальних моделей, забезпечує максимальну адаптивність до змін вимог, дозволяє ефективно планувати роботу в ітераціях (спринтах), сприяє постійному вдосконаленню продукту та активній взаємодії між учасниками процесу розробки.

Обґрунтований вибір мови програмування Dart [2] та фреймворку Flutter [1] (підрозділи 1.5.1 і 1.5.2) базується на таких факторах, як кросплатформність, висока продуктивність, підтримка декларативного UI [7], багата екосистема віджетів, гаряче перезавантаження, а також активна спільнота розробників. Використання саме цих технологій дозволяє мінімізувати витрати часу на

розробку та підтримку, забезпечити легке масштабування функціоналу та створити сучасний інтерфейс, зручний для широкого кола користувачів.

У підрозділі 1.6 було обґрунтовано доцільність використання середовища розробки Android Studio [3], яке забезпечує повну інтеграцію з Flutter [1], підтримує всі етапи створення програмного забезпечення — від написання коду до налагодження, тестування та публікації. Також наголошено на наявності інструментів для емуляції мобільних пристроїв, що дозволяє ефективно проводити тестування гри на різних конфігураціях без залучення фізичного обладнання.

Узагальнюючи вищенаведене, можна зробити висновок, що:

- було сформовано чітку концепцію гри та визначено її функціональні й нефункціональні характеристики;
- здійснено аналіз предметної області та аналогів, що дозволив виявити ключові тенденції і можливості диференціації;
- обрано ефективну модель розробки, що відповідає динамічним умовам і специфіці ігрових проєктів;
- прийнято технічно обґрунтовані рішення щодо вибору мови програмування, фреймворку, середовища розробки.

Таким чином, результати першого розділу створюють надійне підґрунтя для практичної реалізації проєкту, формують чітке бачення його подальшого розвитку та забезпечують методологічну базу для наступних етапів розробки мобільної гри «Морський бій».

2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Архітектура системи

2.1.1 Обґрунтування вибору архітектурного підходу

Для реалізації мобільної гри «Морський бій» було обрано архітектурну модель Model–View–ViewModel (MVVM [8]), яка є однією з найбільш придатних для розробки мобільних застосунків у Flutter [1] завдяки чіткому розподілу

відповідальностей між компонентами та підтримці реактивної парадигми програмування.

В основі MVVM [8] лежить концепція односпрямованого потоку даних, коли кожна зміна у стані програми ініціюється через ViewModel, що взаємодіє з моделлю (Model), обробляє логіку, а потім оновлює відображення (View). Це дозволяє забезпечити масштабованість, полегшує тестування окремих модулів, знижує зв'язність компонентів та підвищує стабільність системи в цілому.

Компоненти моделі MVVM [8]:

- Model – відповідає за зберігання та управління даними. У грі це інформація про стан клітинок, розміщення кораблів, результати ходів тощо.
- ViewModel – реалізує логіку взаємодії. Отримує запити від View, обробляє їх, змінює стан Model і передає змінені дані назад до View.
- View – представляє інтерфейс користувача. Відображає стан даних, отриманий від ViewModel, і реагує на взаємодію користувача (тапи, натискання кнопок тощо).

Такий підхід дозволяє мінімізувати залежність між UI [7] та логікою гри, що особливо важливо при розробці багатоплатформного застосунку з подальшою можливістю масштабування функціональності.

2.1.2 Компонентна структура застосунку

Застосунок побудовано за модульним принципом, де кожен компонент відповідає за чітко визначену функціональність. Це дозволяє розробляти, тестувати та оновлювати частини системи незалежно одна від одної, зберігаючи цілісність архітектури.

Компонентна структура забезпечує можливість гнучкого керування логікою гри, станом, інтерфейсом і взаємодією користувача. Наприклад, окремі сервіси дозволяють у майбутньому вдосконалити штучний інтелект без необхідності змінювати структуру View або ViewModel.

GameModel	Зберігає дані про стан гри, розташування кораблів, результати ходів
GameViewModel	Управляє логікою гри, обробляє

	взаємодію користувача, оновлює стан
GameBoardWidget	Відображає ігрове поле, кораблі, промахи та попадання
ShipPlacementService	Реалізує алгоритм ручної та автоматичної розстановки кораблів
BotPlayerService	Здійснює логіку роботи комп'ютерного супротивника
StatsService	Відповідає за зберігання статистики гри та перемоги
NavigationController	Керує переходами між екранами (головне меню, гра, результати)
UI Components	Створює окремі елементи інтерфейсу: кнопки, діалоги, панелі

Таблиця 1 Компоненти гри

2.1.3 Взаємодія основних модулів

Між усіма компонентами гри реалізовано односпрямований потік даних, що означає, що дані змінюються лише у напрямку: $\text{View} \rightarrow \text{ViewModel} \rightarrow \text{Model}$, після чого зміни передаються у зворотному порядку для оновлення інтерфейсу. Така схема дозволяє централізовано управляти логікою додатку і уникати проблем, пов'язаних із непередбачуваними змінами стану.

Компоненти не взаємодіють безпосередньо між собою (наприклад, View не змінює Model), що виключає ризик неконтрольованої модифікації даних. Всі дії користувача опрацьовуються ViewModel , яка, в свою чергу, викликає відповідні сервіси або змінює стан Model , після чого новий стан відображається у View .

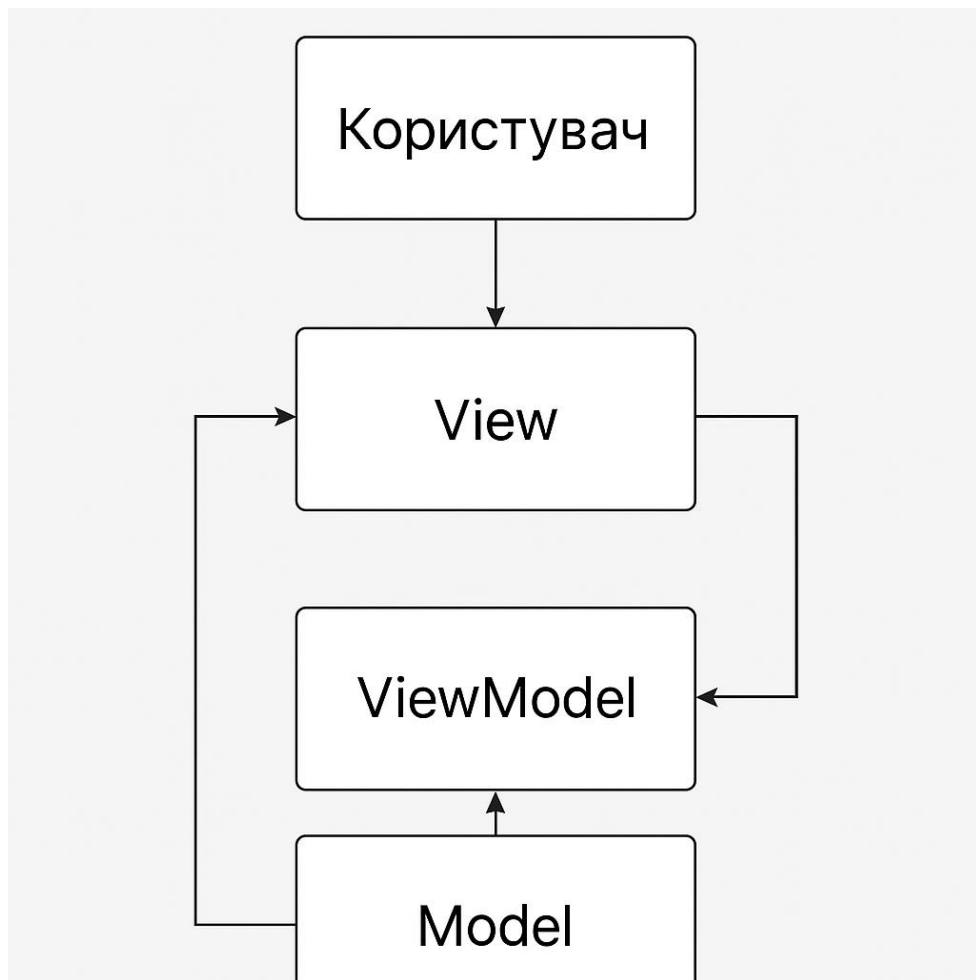


Рисунок 2.1 Діаграма взаємодії компонентів

View – елементи інтерфейсу (екрани, віджети);

ViewModel – координує логіку (напр., обробка кліків);

Model – дані гри (розташування кораблів, статус клітинок).

2.1.4 Життєвий цикл гри

Гра проходить через чітко визначену послідовність етапів, що забезпечує структурованість процесу взаємодії користувача із застосунком. Кожен з етапів відповідає окремому стану системи, у межах якого активуються відповідні компоненти, ViewModel змінює поточну логіку, а інтерфейс оновлюється відповідно до контексту.

На початку гри користувач потрапляє до головного меню. Після вибору режиму розпочинається етап підготовки: ініціалізується поле, користувач розміщує кораблі вручну або за допомогою автоматичного генератора. Далі гра переходить до основного циклу: обробка ходів гравця та супротивника,

перевірка на перемогу/поразку, оновлення стану гри. Після завершення гри користувачу пропонується результат, і надається можливість почати нову сесію.

Етап	Опис
Завантаження інтерфейсу	Ініціалізація графічного інтерфейсу, відображення головного меню
Розміщення кораблів	Гравець або алгоритм виконує розстановку кораблів на ігровому полі
Основна гра	Користувач та бот здійснюють по черзі ходи, ведеться візуалізація
Завершення гри	Відображення результату гри, оновлення статистики
Перезапуск	Очищення стану, повернення до етапу розміщення кораблів

Таблиця 2 Етапи взаємодії користувача

Кожен із цих етапів є логічно відокремленим і реалізується у вигляді станів, які керуються через ViewModel, що дозволяє зручно контролювати перехід між фазами гри, уникаючи дублювання коду та забезпечуючи надійність системи.

2.2 Проектування логіки гри

Проектування логіки гри «Морський бій» передбачає реалізацію формалізованих правил ігрового процесу, які ґрунтуються на класичній настільній грі. Основна мета полягає у створенні стабільної, передбачуваної та розширюваної архітектури поведінки гри, яка керується єдиним джерелом істини — моделлю стану гри (GameModel). Усі дії користувача чи комп'ютерного супротивника повинні уніфіковано оброблятися ViewModel та узгоджено відображатися через View.

2.2.1 Представлення ігрового поля та кораблів

Ігрове поле моделюється як двовимірний масив типу `List<List<Cell>>`, де `Cell` є структурою, що містить інформацію про координати клітинки, її статус (наприклад, порожня, влучення, промах, частина корабля) та інші допоміжні дані (наприклад, індикатори знищення корабля).

Кораблі представлені об'єктами класу `Ship`, який зберігає:

- тип корабля (від 1 до 4 клітинок),
- координати розміщення на полі,
- орієнтацію (горизонтальна або вертикальна),
- список ушкоджених сегментів.

Кожен корабель може мати методи, які дозволяють:

- визначити, чи потрапляння по клітинці спричиняє його пошкодження;
- перевірити, чи корабель повністю знищено;
- повідомити ViewModel про зміну стану корабля.

Ці об'єкти формують колекції `playerShips` та `botShips`, які `ViewModel` використовує під час обробки ігрових подій.

2.2.2 Реалізація класичних правил [10] гри

Гра реалізує всі стандартні правила класичного «Морського бою»:

- Усі кораблі мають фіксовану кількість і розміри, які не можуть змінюватися під час гри.
- Кораблі не можуть торкатися один одного, навіть по діагоналі. Це досягається за допомогою алгоритму валідації сусідніх клітинок під час розміщення.
- Хід передається гравцям по черговому, за винятком ситуацій, коли хід був влучним — тоді гравець може ходити ще раз.
- Після влучання всі суміжні клітинки навколо знищеного корабля автоматично маркуються як промахи (візуалізація «силуету»).
- Гра завершується, коли всі кораблі одного з учасників повністю знищені.

Ця логіка реалізована як набір внутрішніх сервісів у `ViewModel`, які викликаються при кожному ході. Крім того, забезпечено можливість програмного перезапуску гри, скидання стану та оновлення інтерфейсу.

2.2.3 Механізми ручної та автоматичної розстановки кораблів

Користувач має змогу обрати між ручним та автоматичним режимом розміщення кораблів. Це реалізовано як дві взаємовиключні гілки логіки:

- Ручне розташування: реалізується через інтерфейс drag-and-drop або tap-to-place. При кожному розміщенні система перевіряє: чи не перетинається корабель з іншими, чи не виходить за межі поля, чи дотримано допустимих відстаней до інших кораблів. Користувач має можливість повертати корабель (змінювати орієнтацію).
- Автоматичне розташування: запускається генератор, який послідовно перебирає варіанти розміщення і кораблів відповідно до заданої конфігурації. Якщо генерація неуспішна — цикл повторюється до отримання валідної розстановки. Алгоритм використовує випадкові координати та напрямки, з перевіркою сусідніх клітинок.

Ці підходи дозволяють врахувати як потребу користувача у гнучкості, так і необхідність швидкого старту гри.

2.2.4 Логіка ходів гравця та комп'ютерного супротивника

Кожен хід реалізується у вигляді події (event), що активує метод ViewModel. Користувач натискає на клітинку, яка ще не була обстріляна, після чого ViewModel викликає перевірку:

- Чи є корабель у цій клітинці?
- Якщо так — чи знищено весь корабель?
- Якщо ні — чи потрібно повторити хід?

Відповідно, UI [7] оновлює стан поля: колір клітинки, анімації, індикатори знищення. У разі знищення корабля відображаються суміжні промахи.

Хід комп'ютера ініціюється після завершення ходу гравця. Бот обирає координати згідно з логікою складності (див. 2.3). Результати передаються назад у Model і відображаються у View аналогічно діям гравця.

2.2.5 Алгоритми перевірки стану гри

Після кожного ходу виконується серія перевірок:

- Сканування усіх кораблів для визначення, чи були повністю знищені всі елементи (isSunk()).
- Якщо playerShips або botShips повністю виведено з ладу — ініціюється завершення гри.

- Після завершення гри система відображає підсумкове вікно з результатом, та додає запис до історії (перемога/поразка).

Також підтримується логіка перезапуску гри, яка очищує поля, перевантажує ViewModel і повертає гравця на екран розміщення кораблів.

2.3 Модель штучного інтелекту [6] супротивника

Комп'ютерний супротивник у грі діє як автономний агент, що імітує поведінку реального гравця. Його модель включає алгоритми пошуку цілей, реагування на попадання та адаптації до ситуації на полі.

2.3.1 Вимоги до комп'ютерного супротивника

Штучний інтелект повинен:

- Дотримуватися правил гри так само, як і гравець.
- Не мати доступу до розташування кораблів гравця (тобто не «читати» поле).
- Реагувати на результат своїх попередніх ходів.
- Поводитись варіативно, особливо на високих рівнях складності.
- Забезпечувати затримки у діях (імітація мислення) для поліпшення UX.

Ці вимоги формують основу для модульного підходу до реалізації логіки поведінки комп'ютера.

2.3.2 Модель поведінки (рівні складності: легкий, середній, складний)

Передбачено три рівні ІІ:

- Легкий: вибирає клітинки випадково. Не враховує попередніх влучань. Рівень ідеально підходить для ознайомлення та новачків.
- Середній: використовує просту евристику: якщо було влучення, перевіряє суміжні клітинки в усіх напрямках (стратегія добивання).
- Складний (потенційно): реалізовує ймовірнісну карту поля, аналізує шаблони, пріоритетність клітинок. Така поведінка наближена до гри проти людини.

2.3.3 Алгоритми атаки: випадковий вибір, стратегія “пошуку та добивання”

Алгоритм випадкового вибору:

1. Створюється список доступних клітинок.
2. Обирається випадкова клітинка.
3. Якщо попадання - викликається стратегія добивання.

Стратегія добивання:

1. При попаданні зберігаються координати.
2. Послідовно перевіряються клітинки вгору, вниз, вліво та вправо.
3. Після знаходження другого попадання визначається орієнтація корабля.
4. Бот продовжує стріляти в обраному напрямі до повного знищення.

Ці алгоритми інкапсульовані в окремому сервісі (BotPlayerService), що забезпечує чітку ізоляцію логіки П.

2.3.4 Оптимізація продуктивності та часу обробки ходів

Для забезпечення плавності гри застосовується затримка між ходами комп'ютера (наприклад, `Future.delayed(Duration(seconds: 1))`), яка дозволяє уникнути надто швидких реакцій. Це покращує відчуття реалістичності та створює ефект справжнього протистояння.

Обчислення координат атаки виконується асинхронно, що запобігає блокуванню UI [7]-потoku та забезпечує стабільну продуктивність на всіх пристроях.

2.4 Проєктування інтерфейсу користувача

Інтерфейс користувача є ключовим компонентом будь-якого мобільного застосунку, оскільки він визначає якість взаємодії користувача із програмною системою. У випадку з грою «Морський бій», яка реалізується на платформі Flutter [1], особливу увагу було приділено створенню інтуїтивно зрозумілого, адаптивного та функціонального графічного інтерфейсу. Основною метою проєктування інтерфейсу є забезпечення максимальної зручності користувача при мінімальному когнітивному навантаженні.

Розробка інтерфейсу базується на принципах декларативного програмування та компонувальної моделі віджетів, притаманних фреймворку Flutter [1]. Кожен елемент інтерфейсу реалізується як окремий віджет, що

дозволяє досягти високого рівня гнучкості, масштабованості та повторного використання компонентів. Логіка інтерфейсу ізольована від бізнес-логіки гри шляхом використання архітектурної моделі MVVM [8], що забезпечує чіткий розподіл відповідальностей між шарами.

У межах застосунку реалізовано декілька основних екранів: головне меню, ігрове поле, екран завершення гри та модулі налаштувань. Головне меню виконує роль стартового інтерфейсу, надаючи користувачу можливість обрати рівень складності гри, переглянути статистику попередніх сесій або розпочати нову гру. Структура головного меню реалізована за допомогою вертикального компонування елементів (Column) з автоматичною адаптацією до розмірів екрана.

Ігровий екран містить дві сітки 10×10 клітинок, які відображають стан гри: розташування кораблів, результати атак, індикацію попадань та промахів. Окремий віджет GameBoard відповідає за візуалізацію поля та обробку подій взаємодії. Інтерактивність реалізовано через обробники подій, які надсилають інформацію до ViewModel для обробки логіки гри та оновлення інтерфейсу. Додатково на екрані відображається статусна панель, яка містить інформацію про поточний хід, результат пострілу та загальний прогрес гри.

Після завершення гри відображається спеціальний інтерфейс, який містить підсумкову інформацію: перемогу або поразку, загальну статистику та можливість перезапуску гри. Екран переходу реалізовано з використанням анімацій PageRouteBuilder та FadeTransition, що забезпечує плавну зміну між етапами гри та покращує візуальне сприйняття.

Адаптивність інтерфейсу до розмірів екрана є однією з ключових вимог. Застосування MediaQuery, Flexible, Expanded, LayoutBuilder та інших інструментів Flutter [1] дозволило забезпечити коректне відображення інтерфейсу як на смартфонах, так і на планшетах. Додатково враховано підтримку портретної та альбомної орієнтації пристрою. При зміні орієнтації або розміру екрана компоненти інтерфейсу автоматично перебудовуються з урахуванням нових умов, не порушуючи логіки розміщення.

Особливу увагу приділено реалізації візуальних ефектів, які підсилюють емоційне сприйняття гри. Анімації попадання, знищення корабля, індикація промахів та переходи між екранами реалізовано за допомогою вбудованих механізмів Flutter [1] (AnimatedContainer, AnimatedOpacity, TweenAnimationBuilder). Окрім цього, передбачено звукові ефекти, що супроводжують ключові дії (влучення, промах, завершення гри), з можливістю їх вимкнення у налаштуваннях користувача.

Важливою складовою інтерфейсу є доступність. Враховано контрастність шрифтів, розміри інтерактивних елементів, чітку індикацію станів гри, що забезпечує зручність користування навіть на пристроях із невеликим екраном або при поганих умовах освітлення. Застосунок підтримує українську мову інтерфейсу та потенційно допускає реалізацію мультимовності через механізми локалізації Flutter [1].

Таким чином, проектування інтерфейсу користувача гри «Морський бій» забезпечує оптимальний баланс між естетичністю, функціональністю та ергономічністю. Застосовані підходи до побудови UI [7] дозволяють забезпечити ефективну взаємодію з користувачем, знижуючи поріг входу в гру та підвищуючи загальний рівень задоволення від користування застосунком.

2.5 Структура даних і збереження стану гри

У процесі реалізації програмної системи гри «Морський бій» важливою складовою є коректна організація внутрішніх структур даних. Оптимальне представлення даних забезпечує ефективну роботу логіки гри, можливість збереження проміжних результатів та подальшу розширюваність системи. У цьому підрозділі розглянуто структури, що використовуються для зберігання ігрового стану, а також підходи до їхнього збереження під час та після завершення гри.

2.5.1 Представлення ігрового поля

Ігрове поле для кожного гравця реалізовано як двовимірний масив об'єктів Cell, розміром 10×10. Кожен об'єкт Cell містить інформацію про координати клітинки, її поточний стан (порожня, містить корабель, промах, попадання) та

допоміжні поля для візуалізації. Масив забезпечує швидкий доступ до клітинки за індексами, що дозволяє здійснювати ефективні перевірки під час ходу гравця чи супротивника.

Інформація про стан поля представлена переліком enum-типів (наприклад, `CellState.empty`, `CellState.ship`, `CellState.hit`, `CellState.miss`), що забезпечує зрозумілу логіку обробки у `ViewModel` та мінімізує можливі помилки під час реалізації логіки гри.

2.5.2 Структура кораблів

Кожен корабель представлений окремим об'єктом класу `Ship`, який містить такі поля:

- розмір корабля (кількість клітинок),
- список координат (`List<Offset>` або власні `Position`-класи),
- орієнтація (горизонтальна або вертикальна),
- статус пошкодження кожного сегмента,
- ідентифікатор корабля.

Кораблі зберігаються у вигляді списку (`List<Ship>`) для кожного з гравців. Завдяки цьому можливе швидке оновлення стану конкретного корабля, перевірка на повне знищення, а також відображення корабля у візуальному інтерфейсі.

Під час гри об'єкти `Ship` використовуються не тільки для логічної перевірки попадань, але й для формування графічного інтерфейсу, що забезпечує повну відповідність між логікою та візуальним станом.

2.5.3 Структура ходу гри

Для відстеження дій гравців використовується окремий список об'єктів `Move`, які містять таку інформацію:

- координати цілі,
- результат ходу (попадання, промах, знищення),
- індекс гравця (гравець або комп'ютер),
- часову мітку ходу.

Ці об'єкти використовуються для побудови статистики, аналізу сесії гри та відображення в журналі подій (наприклад, у розділі статистики).

Завдяки інкапсуляції ходу в окремий клас можливо реалізувати функціональність скасування або перегляду попередніх кроків у майбутніх версіях застосунку.

2.5.4 Збереження стану гри

Збереження ігрового стану реалізується за допомогою механізмів локального зберігання даних. У рамках реалізації використано Flutter [1] SharedPreferences — легкий інструмент для зберігання простих даних у вигляді ключ-значення.

Інформація, яка зберігається:

- кількість перемог та поразок,
- останній вибраний рівень складності,
- налаштування користувача (наприклад, вимкнення звуку),
- (опційно) дані для відновлення незавершеної гри.

Збереження здійснюється у форматі JSON, що дозволяє легко серіалізувати об'єкти Dart [2], а також забезпечує зручність розширення структури у разі необхідності. Наприклад, структура збереження статистики може виглядати наступним чином:

```
{  
  "wins": 12,  
  "losses": 7,  
  "lastDifficulty": "medium",  
  "soundEnabled": true  
}
```

Після запуску програми відбувається ініціалізація ViewModel, яка зчитує наявні дані та оновлює відповідні елементи інтерфейсу. У разі відсутності попередніх даних ініціалізуються стандартні значення.

2.5.5 Потенційне розширення збереження гри

У майбутньому можлива реалізація повного збереження сесії гри, що дозволить користувачу вийти із застосунку та пізніше відновити гру з того самого моменту. Для цього передбачено створення структури об'єкта GameState, який інкапсулює:

- розташування всіх кораблів обох гравців;
- стан кожної клітинки поля;
- поточний хід;
- історію попередніх дій.

Такий об'єкт може бути серіалізований у JSON або збережений через інші формати (наприклад, SQLite), залежно від складності та обсягу даних. Вибір формату збереження здійснюється з урахуванням балансу між продуктивністю, надійністю та обсягом зберезуваної інформації.

Таким чином, проектування структури даних у грі «Морський бій» реалізовано з дотриманням принципів модульності, ефективності та гнучкості. Використані підходи дозволяють забезпечити узгодженість логіки, стабільність роботи програми та потенціал для розширення функціоналу без суттєвих змін у структурі програмного коду. Ретельне планування збереження та серіалізації ігрового стану дозволяє підтримувати стабільний досвід користувача та забезпечує цілісність даних упродовж усього життєвого циклу застосунку.

2.6 Алгоритм обробки дій користувача

Ефективна обробка дій користувача є одним із ключових аспектів взаємодії людини з програмною системою, особливо у контексті інтерактивних застосунків, таких як мобільна гра. У грі «Морський бій» дії користувача ініціюють ігрові події, що потребують негайного аналізу, зміни стану внутрішніх структур та оновлення інтерфейсу. Реалізація такого механізму потребує побудови чітко визначеного алгоритму, що базується на реактивній парадигмі програмування, підтримуваний фреймворком Flutter [1].

2.6.1 Основні типи дій користувача

У межах гри користувач може здійснювати низку взаємодій, серед яких виділяються основні:

- вибір рівня складності гри;
- ручне розташування кораблів на полі;
- перехід до автоматичної розстановки;
- натискання на клітинку поля супротивника для здійснення пострілу;
- перезапуск гри після завершення;
- перехід до головного меню;
- перегляд статистики перемог/поражок.

Кожна з перелічених дій інтерпретується як подія, яка активує відповідний метод у ViewModel. Це забезпечує централізовану обробку логіки ігрового процесу, ізоляцію інтерфейсу від бізнес-логіки та спрощує тестування.

2.6.2 Алгоритм обробки подій

Усі події користувача обробляються за допомогою шаблону «подія – реакція». Загальний принцип полягає в тому, що кожна дія користувача ініціює виклик функції у ViewModel, яка виконує:

1. Валідацію дії (перевірку її допустимості в поточному стані гри).
2. Оновлення стану Model (наприклад, зміну клітинки поля, оновлення списку кораблів).
3. Генерацію нового стану ViewModel.
4. Нотифікацію інтерфейсу про зміни (через notifyListeners() або механізм setState()).
5. У разі необхідності — ініціацію дій супротивника або зміни екрана.

Наприклад, обробка дії «натискання на клітинку супротивника» передбачає такі етапи:

- перевірка, чи клітинка ще не була обстріляна;
- фіксація координат цілі;
- визначення результату (попадання, промах, знищення);
- оновлення моделі стану поля;

- ініціація ходу комп'ютера у разі промаху.

Такий підхід дозволяє забезпечити детерміновану поведінку гри та мінімізувати можливість помилок через некоректні переходи станів.

2.6.3 Реактивна модель взаємодії

Flutter [1] підтримує реактивну модель побудови UI [7], у якій стан інтерфейсу безпосередньо залежить від даних. Зміни у ViewModel автоматично призводять до оновлення інтерфейсу через механізми сповіщення (ChangeNotifier [18], ValueNotifier, Stream, Riverpod або Provider [17]).

У межах гри обробка дій реалізована за допомогою ChangeNotifier [18], що забезпечує простий і зрозумілий механізм сповіщення про зміну стану. Після зміни внутрішніх структур (наприклад, GameBoard або списку кораблів) ViewModel викликає notifyListeners(), і всі відповідні елементи інтерфейсу оновлюються без необхідності ручного втручання.

Це дозволяє досягти високої реактивності застосунку, знизити кількість помилок і забезпечити плавну взаємодію з користувачем.

2.6.4 Валідація і обробка помилок

Особливу увагу приділено перевірці коректності введених даних та запобіганню невалідним станам. Наприклад:

- користувач не може розмістити корабель на зайнятій клітинку;
- повторний вибір тієї самої клітинки для стрільби заблоковано;
- недопущення виконання пострілу до завершення фази розміщення кораблів.

Усі ці перевірки реалізовано безпосередньо у ViewModel, що гарантує узгодженість логіки та інтерфейсу. При виникненні помилкових дій користувач отримує візуальні або текстові повідомлення, які не переривають процес гри, а лише сповіщають про необхідність зміни дії.

Таким чином, алгоритм обробки дій користувача у мобільній грі «Морський бій» побудований на основі реактивного підходу до взаємодії, централізованої обробки подій та ізоляції логіки у ViewModel. Це дозволяє

забезпечити стабільну та передбачувану поведінку гри, високу продуктивність інтерфейсу та ефективну обробку помилок, що значно покращує користувацький досвід.

2.6 Сценарії використання (Use Case Diagram)

2.6.1 Основні сценарії взаємодії користувача з додатком

Під час проєктування функціональної моделі гри «Морський бій» важливим кроком є аналіз можливих сценаріїв використання застосунку з боку кінцевого користувача. Такий підхід дозволяє визначити основні цілі користувача, структурувати функціональні вимоги, а також описати взаємодію між актором (користувачем) та системою. Сценарії використання визначають набір дій, які може виконувати користувач, та реакцію системи на ці дії.

У контексті гри «Морський бій» основним актором є гравець, який взаємодіє з додатком через інтерфейс користувача. Основні сценарії включають:

- запуск гри;
- вибір рівня складності;
- перегляд статистики;
- ручна або автоматична розстановка кораблів;
- здійснення ходу (пострілу);
- отримання результатів гри;
- перезапуск гри;
- повернення до головного меню.

Кожен зі сценаріїв відповідає конкретному бізнес-випадку, що реалізується у застосунку через відповідну логіку та графічний інтерфейс.

2.6.2 Використання варіантів

Основні варіанти використання:

- «Запустити гру»;
- «Вибрати складність»;
- «Розташувати кораблі» (підрозділяється на «Автоматично» / «Ручно»);
- «Здійснити хід»;

- «Переглянути результат»;
- «Перезапустити гру»;
- «Перейти в головне меню»;
- «Переглянути статистику».

Діаграма допомагає формалізувати інтерфейс між користувачем та функціональністю системи, що є необхідним етапом при проектуванні архітектури.

2.6.3 Послідовність дій при запуску, розстановці та грі

Для детальнішого опису взаємодії користувача із додатком було змодельовано типовий сценарій використання, який охоплює повний життєвий цикл ігрової сесії — від запуску гри до її завершення.

Алгоритм дій:

1. Користувач відкриває додаток.
2. В головному меню обирає рівень складності.
3. Система переходить до етапу розміщення кораблів.
4. Користувач самостійно розміщує кораблі або обирає автоматичну розстановку.
5. Починається ігрова сесія.
6. Користувач по черзі здійснює ходи, натискаючи на клітинки поля супротивника.
7. Система реагує на кожен хід, відображаючи результат (промах, попадання, знищення).
8. Гра завершується при досягненні перемоги або поразки.
9. Відображається результат гри та оновлюється статистика.
10. Користувач може обрати «Перезапустити гру» або «Повернутися до меню».

Такий сценарій є основним для реалізації логіки гри, на його основі реалізовано внутрішні механізми переходів станів у ViewModel.

2.7 Висновки до розділу

У розділі 2 було розглянуто проектування програмної системи мобільної гри «Морський бій» з урахуванням сучасних підходів до архітектури, модульності, обробки дій користувача та збереження стану гри.

Було обґрунтовано вибір архітектурної моделі MVVM [8], яка забезпечує чітке розмежування відповідальностей між компонентами, полегшує супровід та тестування застосунку. Проведено моделювання структури даних, що охоплює представлення ігрового поля, логіку кораблів, хід гри та механізми збереження результатів, що є необхідним для стабільної роботи системи та її масштабованості.

У межах проектування інтерфейсу користувача реалізовано адаптивний, інтуїтивно зрозумілий та естетично привабливий UI [7], що базується на компонентному підході Flutter [1]. Обґрунтовано алгоритми обробки дій користувача з урахуванням реактивного підходу, що дозволяє забезпечити високу швидкодію та чітку реакцію інтерфейсу на дії гравця.

Також було проведено аналіз сценаріїв використання, змодельовано основні взаємодії користувача з системою за допомогою діаграм варіантів використання та послідовностей дій. Отримані моделі забезпечують формалізовану основу для подальшої реалізації логіки гри та забезпечують узгодженість функціональних вимог із проектними рішеннями.

Таким чином, результати проектування, описані в даному розділі, сформували надійну концептуальну та технічну основу для реалізації програмного продукту, який відповідає сучасним вимогам до мобільних ігор [4] у сфері кросплатформної розробки.

Розділ 3. Розробка та тестування програмної системи

3.1 Архітектурна структура реалізації

Реалізація гри «Морський бій» здійснена з використанням архітектурного патерну Model–View–ViewModel (MVVM [8]), який є доцільним у межах платформи Flutter [1] завдяки чіткому поділу логіки, інтерфейсу та управління

станом. Основні компоненти реалізовано через `ChangeNotifier` [18]-провайдер, що дозволяє `ViewModel` реагувати на зміну стану та автоматично оновлювати інтерфейс.

Основні структурні частини:

- `Model: Cell, Ship, Board` — описують сутності гри.
- `ViewModel: GameProvider` [17] — реалізує логіку гри, алгоритми, стан.
- `View: game_screen.dart` — інтерфейс гри, віджети.

Ця архітектура дозволяє легко масштабувати застосунок, забезпечуючи високий рівень підтримки коду та зручність тестування окремих модулів.

3.2 Головна точка входу в застосунок

Файл `main.dart` виконує функцію ініціалізації Flutter [1]-застосунку, підключаючи `GameProvider` [17] як глобальний менеджер стану:

```
void main() {  
  runApp(  
    MultiProvider [17](  
      providers: [  
        ChangeNotifier [18]Provider [17](create: (_) => GameProvider()),  
      ],  
      child: const MyApp(),  
    ),  
  );  
}
```

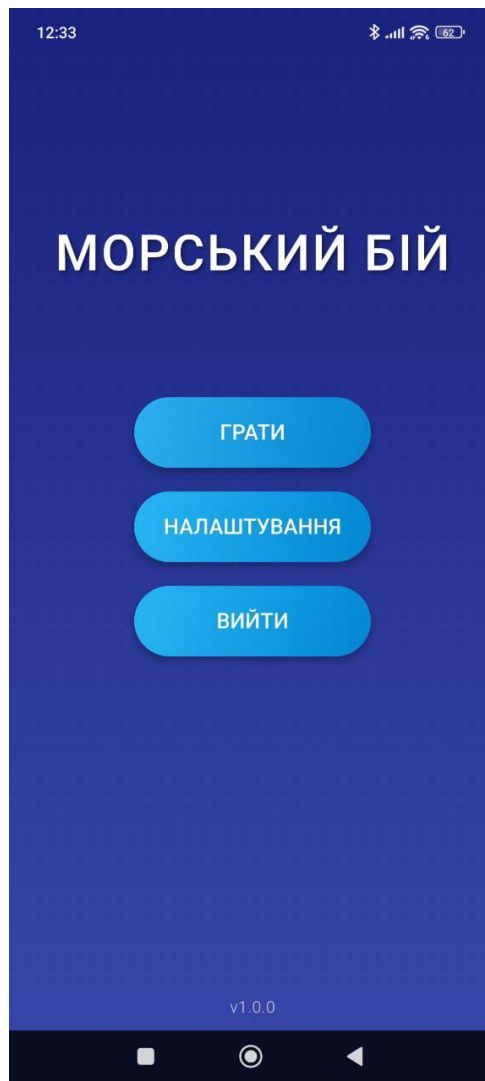


Рисунок 3.1 - Головне меню

Це дозволяє зробити об'єкт `GameProvider` [17] доступним для всіх нащадків віджетів у дереві, забезпечуючи централізовану обробку подій та змін стану.

3.3 Реалізація моделі гри

3.3.1 Клітинка (Cell)

Кожна клітинка поля реалізована як об'єкт класу `Cell`, який зберігає координати, стан ураження та наявність корабля:

```
class Cell {  
    final int x;  
    final int y;  
    bool hasShip;  
    bool isHit;
```

```
Cell({required this.x, required this.y, this.hasShip = false, this.isHit = false});  
}
```

Завдяки цій моделі здійснюється відстеження кожної дії гравця та збереження результатів пострілу.

3.3.2 Корабель (Ship)

Корабель — це структура, що складається з набору клітинок певного розміру:

```
dart  
class Ship {  
  final int size;  
  List<Cell> cells;  
  
  Ship({required this.size, required this.cells});  
}
```

Це дозволяє відслідковувати, які частини корабля уражено, і визначати, коли корабель знищено.

3.3.3 Ігрове поле (Board)

Поле реалізоване як двовимірна матриця клітинок, створена методом `generateBoard()`:

```
dart  
class Board {  
  final List<List<Cell>> cells;  
  
  Board() : cells = List.generate(10, (x) => List.generate(10, (y) => Cell(x: x, y: y)));  
}
```

Доступ до клітинки здійснюється через метод `getCell(x, y)` — це спрощує звернення у `ViewModel`.

3.4 Логіка гри у `ViewModel`

Основна логіка реалізована у класі `GameProvider` [17], який відповідає за:

- генерацію кораблів;
- зміну стану клітинок;
- обробку пострілів;
- перемикання ходу;
- завершення гри.

3.4.1 Генерація кораблів

Функція `placeShipsRandomly()` розміщує кораблі на полі у випадковому порядку, з урахуванням правил (без перетину):

```
dart
void placeShipsRandomly() {
  playerShips.clear();
  // Для кожного розміру корабля шукається допустима позиція
  for (var size in [4, 3, 3, 2, 2, 2, 1, 1, 1, 1]) {
    // Перевірка на допустимість та генерація
  }
  notifyListeners();
}
```

3.4.2 Обробка пострілу

Після натискання на клітинку користувачем, викликається метод `shootAt()`:

```
dart
void shootAt(int x, int y) {
  final cell = enemyBoard.getCell(x, y);
  if (!cell.isHit) {
    cell.isHit = true;
    if (cell.hasShip) {
      // обробка попадання
    }
  }
}
```

```

    } else {
        switchTurn();
    }
    notifyListeners();
}
}

```

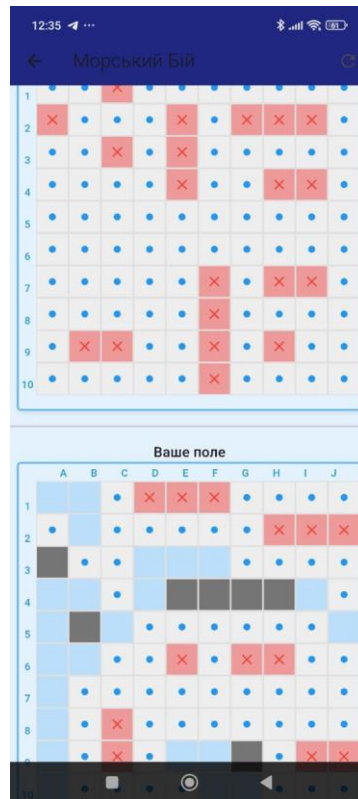


Рисунок 3.2 - Поле під ураженням

Обробка влучання змінює стан клітинки та визначає, чи знищено весь корабель.

3.4.3 Перевірка завершення гри

Кожен раз після ходу перевіряється умова завершення гри:

dart

```

bool isGameOver() {
    return enemyShips.every((ship) => ship.cells.every((cell) => cell.isHit));
}

```

Якщо всі кораблі знищені, користувачу показується повідомлення про перемогу або поразку.

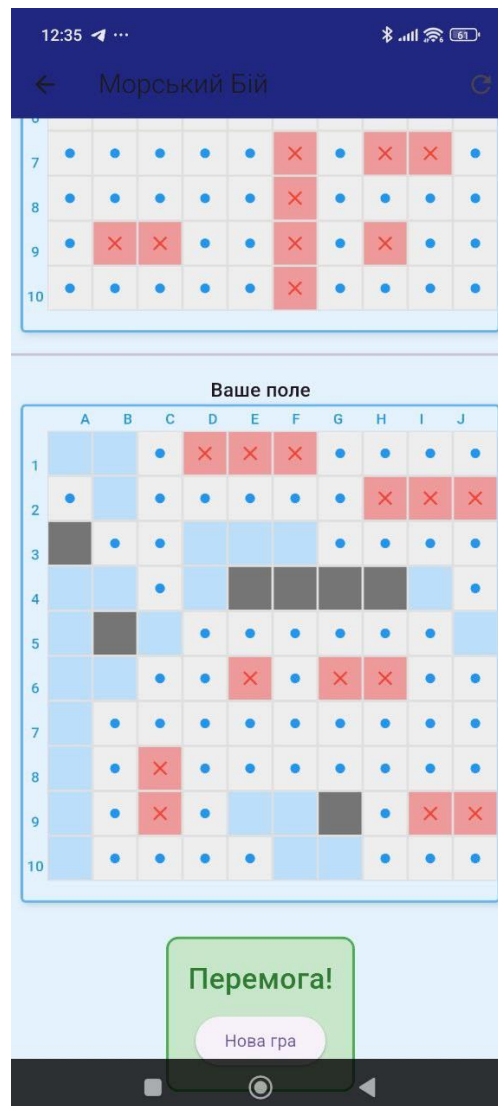


Рисунок 3.3 - Відображення перемоги гравця

3.5 Інтерфейс користувача

Інтерфейс реалізовано у файлі `game_screen.dart`. Основне поле гри створюється за допомогою `GridView`:

```
GridView.builder( gridDelegate: const  
SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 10),  
    itemCount: 100,  
    itemBuilder: (context, index) {
```

```

final x = index % 10;
final y = index ~/ 10;
final cell = gameProvider [17].enemyBoard.getCell(x, y);
return GestureDetector(
  onTap: () => gameProvider [17].shootAt(x, y),
  child: CellWidget(cell: cell),
);
},
),

```

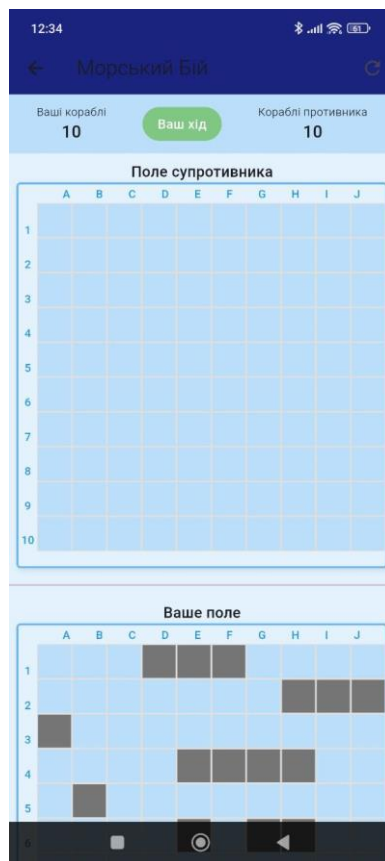


Рисунок 3.4 - Поле гравця

Кожен CellWidget змінює колір або відображає графічний елемент залежно від стану (hit, miss, hasShip).

3.6 Тестування програмної системи

3.6.1 Модульне тестування логіки гри

Модульні тести реалізовані з використанням пакету test. Наприклад, перевірка зміни стану клітинки:


```

dart
test('Cell should be hit after shoot', () {
  final cell = Cell(x: 0, y: 0);
  expect(cell.isHit, false);
  cell.isHit = true;
  expect(cell.isHit, true);
});

```

Інші тести перевіряють:

- правильну генерацію кораблів;
- відсутність перетинів;
- коректне завершення гри;
- зміну черги ходу.

3.6.2 Ручне тестування інтерфейсу

Здійснювалося у середовищі Android Studio [3] із використанням емулятора. Перевірено:

- перехід між екранами;
- відображення попадань і промахів;
- візуальна реакція на натискання;
- правильне оновлення статистики;
- плавність відображення анімацій.

3.7 Висновки до розділу

У даному розділі було розглянуто повний цикл реалізації мобільної гри «Морський бій»: від архітектурного підходу та побудови моделі до логіки гри, інтерфейсу та тестування. Розробка виконана з урахуванням принципів чистої архітектури, що забезпечило високу якість коду, стабільність і масштабованість системи.

Застосування MVVM [8]-архітектури, розділення відповідальностей та використання механізмів Flutter [1]-платформи дозволили реалізувати надійний ігровий застосунок із сучасним інтерфейсом, який задовольняє вимоги користувачів та дозволяє легко його розвивати в майбутньому.

Висновки

У ході виконання дипломної роботи було успішно розроблено кросплатформну мобільну гру «Морський бій» на основі фреймворку Flutter [1] із застосуванням архітектурного підходу MVVM [8] та мови програмування Dart [2]. Здійснено повний цикл розробки—від аналізу предметної області та вимог (розділ 1), через проєктування архітектури й ігрової логіки (розділ 2), до реалізації та тестування програмної системи (розділ 3). Результати свідчать про відповідність програмного продукту поставленим функціональним і нефункціональним вимогам, а також про його готовність до практичного використання.

Проведений аналіз аналогів дозволив виявити основні тренди ринку мобільних «морських боїв»: широке застосування рушія Unity [11] для забезпечення мережових режимів і кастомізації, а також домінування ігор із рекламними та внутрішніми покупками. Водночас відсутність Flutter [1]-реалізацій відкриває нішу для продуктивного та гнучкого продукту з єдиною кодовою базою. Обраний підхід із фокусом на одноособову гру проти комп'ютера забезпечив реалізацію стабільного ігрового процесу без надмірної складності серверної частини.

Проєктування та імплементація логіки гри охопили як класичну механіку розташування кораблів, перевірку стану гри й алгоритми ходів, так і три рівні складності штучного інтелекту [6]: від випадкового вибору клітинок до стратегії «пошуку й добивання». Затримки між діями бота й асинхронне виконання обчислень забезпечили плавність інтерфейсу та реалістичність ігрового процесу. Модульне тестування й ручна перевірка інтерфейсу підтвердили коректність роботи основних компонентів та стабільність застосунку.

Інтерфейс користувача реалізовано з урахуванням принципів адаптивності, доступності й мінімального когнітивного навантаження: два 10×10 поля, чітка індикація попадань і промахів, плавні анімації переходів і звукові ефекти. Використання декларативного підходу Flutter [1] і виджетної композиції

спростило підтримку локалізації та масштабування дизайну під різні розміри екранів і орієнтації пристрою.

Отримані результати демонструють практичну придатність розробленого програмного продукту й створюють міцну основу для подальшого розвитку проєкту. Передбачено розширення функціональності через:

- реалізацію мережевих режимів гри з підтримкою мультиплеєра;
- додавання рейтингових таблиць і соціальної взаємодії між гравцями;
- оптимізацію продуктивності та підтримку нових платформ (desktop, web);
- впровадження аналітики користувацької поведінки для покращення UX.

Таким чином, виконана робота відповідає сучасним вимогам цифрових ігор і підтверджує ефективність застосування Flutter [1] у створенні кросплатформних мобільних продуктів із високим рівнем продуктивності та зручності користування

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація Dart [2]. – [Електронний ресурс] - 2024 Режим доступу:
2. Офіційна документація Android Studio [3]. – [Електронний ресурс] - 2024 Режим доступу:
3. JetBrains. Офіційна документація IntelliJ IDEA. – [Електронний ресурс] - 2024 Режим доступу: <https://www.jetbrains.com/idea/documentation/>
4. Martin Fowler. Patterns of Enterprise Application Architecture. – [Книга] – 2002.
5. Read, J. (2024). Communication Patterns (pp. 170-178). O'Reilly Media. ISBN-13: 978-1-098-14054-0.
6. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – [Книга] – p 126-127 - 2017.
7. Martin Fowler. Patterns of Enterprise Application Architecture. – [Книга] – 2002.
8. Read, J. Communication Patterns (pp. 170–178). O'Reilly Media, 2024. ISBN-13: 978-1-098-14054-0.
9. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – [Книга] – p. 126–127 – 2017.
10. Google. Flutter [1] Documentation. – [Електронний ресурс]. – Режим доступу:
11. Google. Dart [2] Language Tour. – [Електронний ресурс]. – Режим доступу:
12. JetBrains. Kotlin Programming Language Documentation. – [Електронний ресурс]. – Режим доступу: <https://kotlinlang.org/docs/home.html>
13. Android Developers. Android Studio [3] User Guide. – [Електронний ресурс]. – Режим доступу:
14. MVVM [8] Design Pattern in Flutter [1] – [Електронний ресурс] – Medium.com. – Режим доступу:
15. Kivy Project. Official Documentation. – [Електронний ресурс]. – Режим доступу: <https://kivy.org/doc/stable/>

- 16.WebRTC Project. WebRTC API Reference. – [Электронный ресурс]. – Режим доступа: <https://webrtc.org>
- 17.Flutter [1]Dev. Provider [17] Package – State Management [19]. – [Электронный ресурс]. – Режим доступа:
- 18.Raywenderlich.com. Flutter [1] Testing Tutorial: Unit, Widget, and Integration Tests. – [Электронный ресурс] – Режим доступа:
- 19.Hasbro. Battleship [14] – Official Game Rules. – [Электронный ресурс]. – Режим доступа:
- 20.Smuttlewerk Interactive. Fleet Battle [13] – Game Description and Features. – [Электронный ресурс]. – Режим доступа:

ДОДАТКИ

Додаток А

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:provider/provider.dart';
import 'screens/home_screen.dart';
import 'screens/game_screen.dart';
import 'screens/settings_screen.dart';
import 'providers/game_provider.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  // Фіксація орієнтації екрану на портретний режим
  SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
  ]);
  runApp(const Battleship [14]Game());
}

class Battleship [14]Game extends StatelessWidget {
  const Battleship [14]Game({ Key? key }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return ChangeNotifier [18]Provider [17](
      create: (context) => GameProvider [17](),
      child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'Морський Бій',
        theme: ThemeData(
          primarySwatch: Colors.blue,
          visualDensity: VisualDensity.adaptivePlatformDensity,
          fontFamily: 'Roboto',
          appBarTheme: const AppBarTheme(
            backgroundColor: Color(0xFF1A237E),
            elevation: 0,
          ),
        ),
      ),
    );
  }
}
```

```

    scaffoldBackgroundColor: const Color(0xFFE3F2FD),
  ),
  initialRoute: '/',
  routes: {
    '/': (context) => const HomeScreen(),
    '/game': (context) => const GameScreen(),
    '/settings': (context) => const SettingsScreen(),
  },
),
);
}

```

```

import 'dart:math';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

```

```

class Utils {
  // Згенерувати випадкове число в діапазоні
  static int getRandomInt(int min, int max) {
    return min + Random().nextInt(max - min + 1);
  }

  // Конвертація координат з буквено-числового формату (A1, B5, ...) в індекси
  static List<int> coordinatesFromString(String coord) {
    if (coord.length < 2 || coord.length > 3) {
      throw Exception('Неправильний формат координат: $coord');
    }

    // Перша літера - стовпець (A-J)
    String columnStr = coord[0].toUpperCase();
    int column = columnStr.codeUnitAt(0) - 'A'.codeUnitAt(0);

    // Решта - рядок (1-10)
    String rowStr = coord.substring(1);
    int row = int.parse(rowStr) - 1;

    if (column < 0 || column > 9 || row < 0 || row > 9) {
      throw Exception('Координати поза межами поля: $coord');
    }
  }
}

```



```

        return [row, column];
    }

    // Конвертація індексів у буквено-числовий формат
    static String coordinatesToString(int row, int column) {
        String columnStr = String.fromCharCode('A'.codeUnitAt(0) + column);
        return '$columnStr${row + 1}';
    }

    // Вібрація при певних діях
    static Future<void> vibrate({int duration = 100}) async {
        HapticFeedback.mediumImpact();
        if (duration > 100) {
            await Future.delayed(Duration(milliseconds: 50));
            HapticFeedback.mediumImpact();
        }
    }

    // Стандартні розміри кораблів для гри
    static List<int> getDefaultShipSizes() {
        return [4, 3, 3, 2, 2, 2, 1, 1, 1, 1]; // [1x4, 2x3, 3x2, 4x1]
    }

    // Отримати колір для стану клітинки
    static Color getCellColor(String state, {bool isEnemyBoard = false}) {
        switch (state) {
            case 'empty':
                return Colors.blue.shade100;
            case 'ship':
                return isEnemyBoard ? Colors.blue.shade100 : Colors.grey.shade700;
            case 'hit':
                return Colors.red.shade400;
            case 'miss':
                return Colors.grey.shade300;
            case 'adjacent':
                return Colors.blue.shade50;
            default:
                return Colors.blue.shade100;
        }
    }

```

```

    }
}

// Отримати іконку для стану клітинки
static IconData? getCellIcon(String state) {
    switch (state) {
        case 'ship':
            return Icons.directions_boat;
        case 'hit':
            return Icons.close;
        case 'miss':
            return Icons.circle_outlined;
        default:
            return null;
    }
}

// Форматування часу для таймера
static String formatTime(int seconds) {
    final int minutes = seconds ~/ 60;
    final int remainingSeconds = seconds % 60;
    return '${minutes.toString().padLeft(2,
'0')}:${remainingSeconds.toString().padLeft(2, '0')}';
}

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/game_provider.dart';
import '../widgets/board_widget.dart';

class GameScreen extends StatelessWidget {
    const GameScreen({ Key? key }) : super(key: key);

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: const Text('Морський Бій'),

```



```

    ),

    // Повідомлення про стан гри
    if (gameProvider [17].gameStatus != GameStatus.inProgress)
      _buildGameOverMessage(context, gameProvider [17]),
  ],
),
);
},
),
);
}

Widget _buildInfoPanel(BuildContext context, GameProvider [17] gameProvider) {
  return Container(
    padding: const EdgeInsets.all(10),
    color: Colors.blue.shade100,
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceAround,
      children: [
        _buildStatColumn('Ваші кораблі', '${gameProvider
[17].playerShipsRemaining}'),
        _buildTurnIndicator(gameProvider [17]),
        _buildStatColumn('Кораблі противника', '${gameProvider
[17].enemyShipsRemaining}'),
      ],
    ),
  );
}

Widget _buildStatColumn(String title, String value) {
  return Column(
    children: [
      Text(title, style: const TextStyle(fontSize: 12)),
      Text(value, style: const TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
    ],
  );
}

```

```

Widget _buildTurnIndicator(GameProvider [17] gameProvider) {
  final isPlayerTurn = gameProvider [17].isPlayerTurn;
  return Container(
    padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
    decoration: BoxDecoration(
      color: isPlayerTurn ? Colors.green.shade300 : Colors.red.shade300,
      borderRadius: BorderRadius.circular(20),
    ),
    child: Text(
      isPlayerTurn ? 'Ваш хід' : 'Хід противника',
      style: const TextStyle(
        color: Colors.white,
        fontWeight: FontWeight.bold,
      ),
    ),
  );
}

```

```

Widget _buildBoardSection(BuildContext context, String title, Widget board) {
  return Padding(
    padding: const EdgeInsets.all(8.0),
    child: Column(
      children: [
        Text(
          title,
          style: const TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.bold,
          ),
        ),
        SizedBox(
          height: MediaQuery.of(context).size.width,
          child: board,
        ),
      ],
    ),
  );
}

```

```

Widget _buildGameOverMessage(BuildContext context, GameProvider [17]
gameProvider) {
  final isVictory = gameProvider [17].gameStatus == GameStatus.playerWon;
  return Container(
    margin: const EdgeInsets.all(20),
    padding: const EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: isVictory ? Colors.green.shade100 : Colors.red.shade100,
      borderRadius: BorderRadius.circular(10),
      border: Border.all(
        color: isVictory ? Colors.green : Colors.red,
        width: 2,
      ),
    ),
    child: Column(
      children: [
        Text(
          isVictory ? 'Перемога!' : 'Поразка!',
          style: TextStyle(
            fontSize: 24,
            fontWeight: FontWeight.bold,
            color: isVictory ? Colors.green.shade800 : Colors.red.shade800,
          ),
        ),
        const SizedBox(height: 10),
        ElevatedButton(
          onPressed: () {
            gameProvider [17].startNewGame();
          },
          child: const Text('Нова гра'),
        ),
      ],
    ),
  );
}

void _showRestartDialog(BuildContext context) {
  showDialog(

```

```

context: context,
builder: (context) => AlertDialog(
  title: const Text('Перезапустити гру?'),
  content: const Text('Весь прогрес буде втрачено.'),
  actions: [
    TextButton(
      onPressed: () => Navigator.pop(context),
      child: const Text('Скасувати'),
    ),
    ElevatedButton(
      onPressed: () {
        Navigator.pop(context);
        Provider [17].of<GameProvider>(context, listen: false).startNewGame();
      },
      child: const Text('Перезапустити'),
    ),
  ],
),
);
}
}

```

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут фізики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

КЕРІВНИЦТВО КОРИСТУВАЧА
на виконання програмної розробки (ПР):
«ПРОЕКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОЇ ВЕРСІЇ ГРИ
«МОРСЬКИЙ БІЙ» ДЛЯ ПЛАТФОРМИ ANDROID З
ВИКОРИСТАННЯМ КРОСПЛАТФОРМНОГО ФРЕЙМВОРКУ
FLUTTER ЗАСОБАМИ MICROSOFT VISUAL STUDIO"

ІТС.ПЗ4.0721-03-КК

ПОГОДЖЕНО
Керівник кваліфікаційної роботи

Семенов М.А.

“ ” 2025р

ВИКОНАВЕЦЬ
Студент групи 4ПЗ

Соколов Д.І.

“ ” 2025р

Старобільськ 2025

ЗМІСТ

1.1. Підготовка до роботи з додатком	3
1.2. Робота з додатком	3

1.1. Підготовка до роботи з додатком

Смартфони на базі ОС Android

Для коректної роботи додатку на смартфонах повинна бути встановлена операційна система Android версії 4.4 або вище, або iOS версії 10.0 або вище. Якщо ви не знаєте версію вашої ОС, це можна перевірити в налаштуваннях смартфона в розділі «Про телефон» для Android або в «Налаштуваннях» -> «Основні» -> «Інформація про пристрій» для iOS.

Якщо версія операційної системи нижча за необхідну, перевірте, чи підтримує ваш пристрій оновлення до цієї версії. Для цього можна виконати наступні дії:

Для Android: Оновлення можна завантажити через Wi-Fi за допомогою функції «повітряного завантаження» в налаштуваннях смартфона в розділі «Про телефон». Просто слідуйте вказівкам на екрані.

Для iOS: Оновлення ОС можна завантажити безпосередньо через налаштування пристрою в розділі «Оновлення програмного забезпечення».

Для Android через комп'ютер: Якщо оновлення не доступне безпосередньо через смартфон, ви можете завантажити нову версію ОС за допомогою спеціальної програми, що постачається з вашим телефоном. Для цього підключіть пристрій до комп'ютера через USB, запустіть програму і виберіть опцію «Завантажити нову версію ОС Android». Потім слідуйте інструкціям програми.

1.2. Робота з додатком

Ключові стани розробленої гри представлені на рисунках 1.1 - 1.5. Головне меню (рис. 1.1) складається з трьох кнопок: Грати, налаштування і вийти.

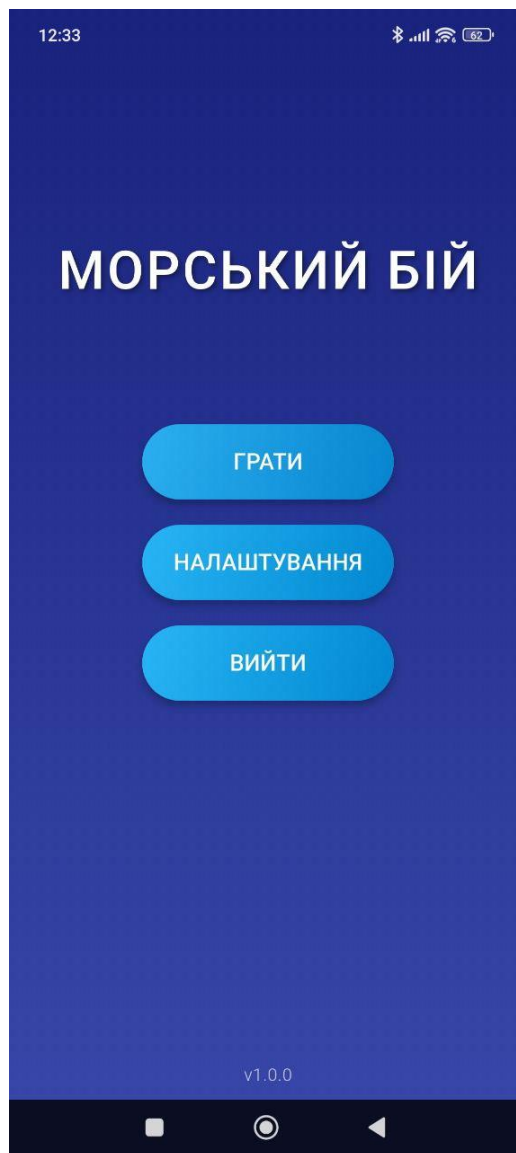


Рисунок 1.1. Вигляд головного меню

Кнопка «Вийти» надає можливість вийти з гри. У кнопці «Налаштування» гравцю надається вибір складності між Легким, Середнім та Складним, та можливість ввімкнути вібрацію та автоматичне розміщення кораблів(рис 1.2).

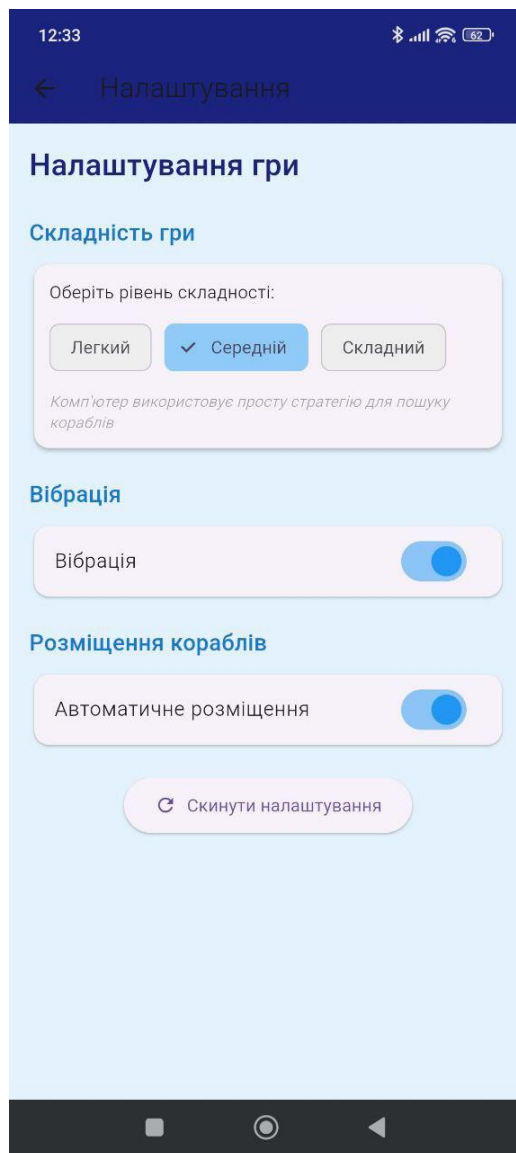


Рисунок 1.2. Меню налаштувань

Кнопка «Грати» дозволяє перейти на наступну сцену виставу кораблів(якщо опція автоматичного виставлення вимкнена) і потім початок самої гри(рис 1.3).

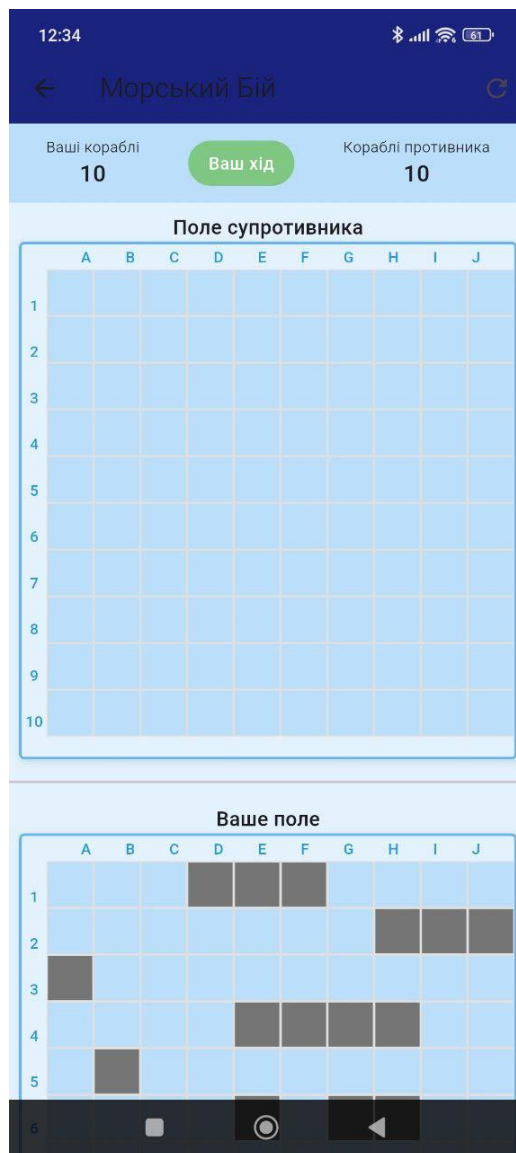


Рисунок 1.3. Меню гри

Натискаючи пальцем на клітинки від А-І та 1-10 користувач обирає яку клітинку він буде атакувати, при влучному попаданні надається ще одна спроба влучити це продовжується до поки користувач не промаже. Після цього настає хід комп'ютера, який користується тим же правилами як і користувач.(рис 1.4).

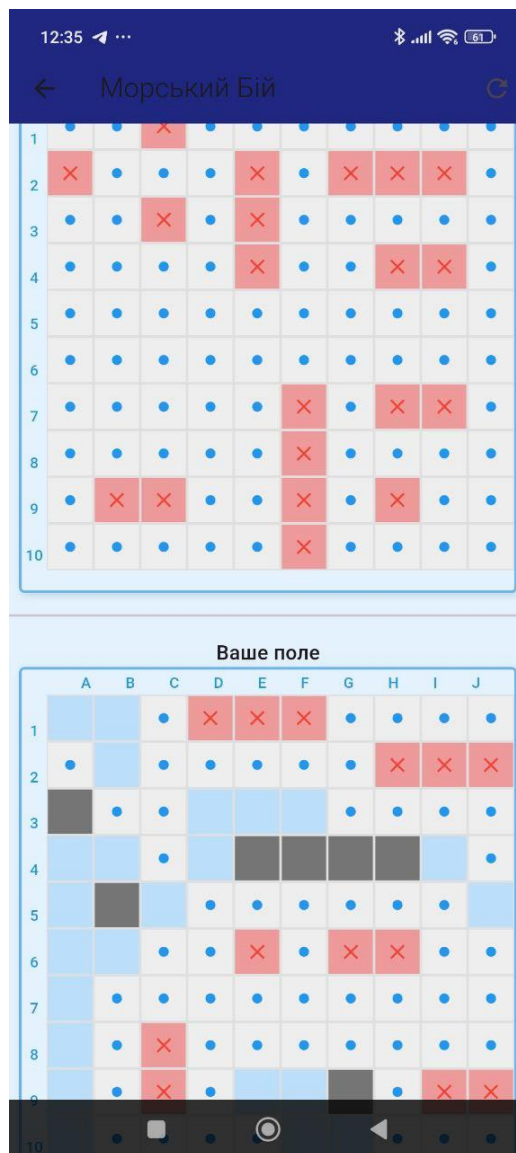


Рисунок 1.4. Скріншот процесу гри

Після перемоги або поразки з'являється вікно з словами «Перемога!» або «Поразка»(рис 1.5). Після цього з'являється можливість почати нову гру або вийти в головне меню стрілочкою зверху зліва.

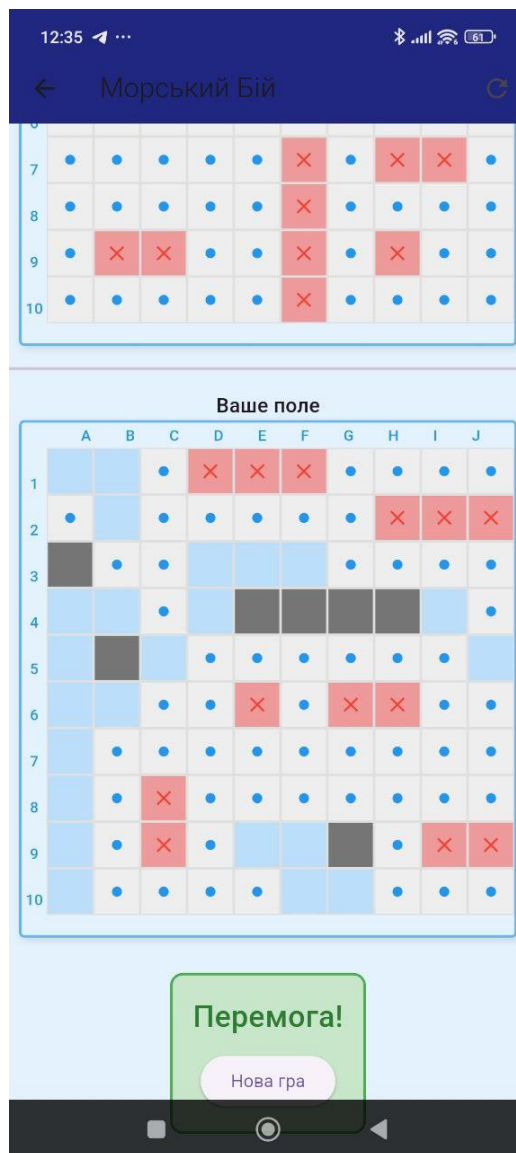


Рисунок 1.5. Скріншот після перемоги