Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity”

Виконав здобувач вищої освіти
IV курсу, групи КІ-22мб
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Голубєв В. О.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Голубєву Віктору Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№48-02 від 17.01.2025 року*

2. Строк подання студентом роботи до захисту *24.05.2025 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання 17.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	22.05.2025 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Голубєв В. О. Програмне забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity. 123 Комп’ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для моделювання роботи комп’ютерних мереж.

Метою роботи є створення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

Результат роботи – програмна реалізація комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування C# та у середовищі розробки Unity.

Ключові слова: комп’ютерна інженерія, комп’ютерні ігри, ігровий рушій Unity, симулятор

ABSTRACT

Holubiev V. O. Software for a Computer-Game Simulator Based on the Unity Game Engine. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2025.

This bachelor's qualification work presents software designed to model the operation of computer networks.

The objective of the study is to create a computer game in the simulator genre using the Unity game engine.

The outcome is a fully functioning software implementation of a simulator-genre computer game built on Unity.

During development, existing methods, algorithms, and software tools were analysed. Custom software was then designed and implemented, with all components documented in detail.

A user-friendly interface was created, and instructions for working with the software tools are provided.

The program can be run on IBM PC-compatible personal computers under Windows 10/11.

Development was carried out in the C# programming language within the Unity environment.

Keywords: computer engineering, computer games, Unity game engine, simulator

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Canvas – система побудови UI в Unity, яка дозволяє створювати меню, панелі, кнопки тощо.

CI (Continuous Integration) – підхід до розробки, при якому кожна зміна в коді автоматично перевіряється й збирається. GitHub Actions – приклад системи CI.

CSV (Comma Separated Values) – простий формат зберігання табличних даних у текстовому вигляді, який підтримується більшістю редакторів.

CustomRenderTexture – компонент Unity, який дозволяє створювати текстури, що оновлюються на основі шейдерів у реальному часі. Використовується для симуляції хвиль.

DockZone / DockTrigger – логічна зона на сцені, що сигналізує про завершення рівня при вході човна в неї.

EULA (End-User License Agreement) – ліцензійна угода з кінцевим користувачем, що регулює права використання програмного продукту.

Game Loop – цикл гри, в межах якого кожен кадр оновлюється ввід, фізика, логіка та рендеринг.

GPU (Graphics Processing Unit) – графічний процесор, що використовується для паралельних обчислень, включно з рендерингом зображень і симуляцією фізичних процесів.

HP (Health Points) – очки здоров’я об’єкта, які зменшуються при зіткненнях.

HUD (Heads-Up Display) – графічний інтерфейс, накладений на основну сцену, що містить інформацію (таймер, очки, стан).

InputAdapter – скрипт, що уніфікує обробку вводу з різних платформ (клавіатура, сенсорний екран, ґеймпад).

Joystick (MobileJoystick) – елемент керування на екрані мобільного пристрою, що імітує аналоговий контролер.

JSON (JavaScript Object Notation) – формат зберігання даних у вигляді тексту, зручний для серіалізації й передачі через мережу.

LTS (Long Term Support) – версія програмного забезпечення з тривалим терміном підтримки. Unity 2025 LTS гарантує стабільність API та сумісність протягом кількох років.

PhysX – фізичний рушій, який використовується Unity для обчислення руху, зіткнень, сили, тяжіння тощо.

PlayerPrefs – вбудована система Unity для збереження невеликих обсягів даних (налаштувань, прогресу) у вигляді ключ-значення.

Prefab – шаблон об'єкта в Unity, який можна повторно використовувати у сценах.

Rigidbody – фізичний компонент Unity, який додає об’єктові фізичні властивості, такі як маса, сила, інерція.

SaveLoadSystem – модуль, відповідальний за збереження й завантаження прогресу гравця.

Score / Points / Бали – числова оцінка досягнень гравця у грі.

ScriptableObject – спеціальний тип об'єкта в Unity, який дозволяє створювати незалежні від сцени дані (наприклад, налаштування).

Shader / Шейдер – програма, яка виконується на GPU для обробки візуальних ефектів, наприклад, текстурування, освітлення, хвиль.

Timer – лічильник часу, який відстежує тривалість сеансу гри.

Unity – ігровий рушій, що підтримує 2D і 3D-графіку, фізику, звук, анімацію, UI та крос-платформенну збірку. Дозволяє створювати інтерактивні програми для Windows, Android, iOS, macOS, Linux тощо.

URP (Universal Render Pipeline) – шаблон рендерингу в Unity, що оптимізує баланс між якістю графіки та продуктивністю, особливо на мобільних пристроях.

ВСТУП

Актуальність теми. У рамках обраної теми кваліфікаційної роботи було вирішено розробляти симулятор морського човна.

Упродовж останнього десятиріччя симулятори посіли вагомий сегмент ринку інтерактивних розваг і навчальних програм. На відміну від традиційних аркадних ігор, де головну роль відіграє швидкість реагування, симуляційні продукти відтворюють фізичні закони та реальні сценарії з максимально можливою точністю. Такий підхід робить їх актуальними не лише для дозвілля, а і як інструмент початкової підготовки операторів складної техніки, пілотів чи судноводіїв. Водночас поява доступних, крос-платформних рушіїв – насамперед Unity – суттєво знизила вхідний бар'єр для розробників, дозволивши реалізовувати фізично достовірні моделі на споживчому обладнанні.

Актуальність обраної теми зумовлена двома взаємопов'язаними чинниками. По-перше, морські та внутрішні водні перевезення демонструють стабільну динаміку зростання, а отже, збільшується потреба у швидкій та економічній підготовці шкіперів маломірних суден. Практичні заняття на реальній акваторії пов'язані з високими витратами пального, ризиком технічних пошкоджень та залежністю від погодних умов. По-друге, на ринку інтерактивних розваг спостерігається сталий попит на «casual-sim» – ігри, що поєднують навчальну складову з доступним ігровим процесом і працюють як на настільних, так і на мобільних платформах.

Створений у межах дипломної роботи проєкт належить до жанру навігаційних симуляторів із тактичним керуванням. Об'єктом моделювання є процес швартування невеликого моторного човна в умовах гавані з нерівномірним хвилюванням та штучними перешкодами – пірсами, буями, понтонами. На відміну від глобальних морських симуляторів, де основним викликом є тривала навігація у відкритому морі, запропонований застосунок

зосереджується на маневрах у вузьких водах, коли ключове значення мають інерція, дія гідродинамічного гвинта та відбиті хвилі. Таким чином, продукт позиціонується як «мікросимулятор» – вузькоспеціалізований тренажер коротких епізодів, що тривають від однієї до п’яти хвилин і завершуються оцінкою точності та безпечності швартування.

Мета й завдання дослідження. Метою роботи є створення програмного забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для реалізації комп’ютерних ігор у жанрі симуляторів на основі ігрового рушія Unity.

– Проектування системи для реалізації комп’ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

– Програмна реалізація системи для реалізації комп’ютерної гри у жанрі симулятора морського човна на основі ігрового рушія Unity.

Платформою реалізації обрано ігровий рушій Unity 2025 LTS, який надає готову фізичну підсистему, розвинуті можливості GPU-шейдерів і крос-платформний інструментарій збірки, що є критично важливим для одночасного випуску на Windows, macOS, Linux, Android та iOS. Використання C# як основної мови скриптів забезпечує швидку ітерацію та легку інтеграцію з бібліотеками штучного інтелекту і модулем серіалізації.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно здійснювати реалізацію комп’ютерної гри у жанрі симулятора морського човна на основі ігрового рушія Unity.

Таким чином, дипломний проєкт «Програмне забезпечення комп’ютерної гри у жанрі симулятора на основі ігрового рушія Unity» спрямований на розроблення багатоплатформного тренажера швартування з достовірною гідродинамікою, дружнім користувацьким інтерфейсом і системою збереження прогресу. Результати роботи

можуть бути безпосередньо використані у навчальному процесі морських шкіл, а також слугувати демонстраційною платформою для досліджень у галузі процедурної симуляції води й оптимізації мобільної графіки.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Симулятор швартування маломірного судна, розроблений у межах кваліфікаційної роботи, призначений для двох взаємодоповнювальних цілей. По-перше, це інтерактивний навчальний тренажер, який відтворює основи керування човном у гавані: подавання ходу, маневрування в обмеженому просторі, корекцію курсу з урахуванням інерції, хвиль і підводних перешкод, а також точну постановку судна до понтона. Програма дає користувачеві можливість без ризику пошкоджень освоїти базові навички морського пілотування, відчути вплив водної динаміки та оцінити наслідки помилок, що у реальних умовах можуть спричинити матеріальні збитки чи аварійну ситуацію.

По-друге, симулятор виконує роль дослідницької платформи для апробації алгоритмів реалістичної водної поверхні, фізики плавучих тіл і процедурного аналізу зіткнень. Завдяки поєднанню GPU-симуляції хвиль, скриптової обробки колізій і системи збору статистики застосунок дозволяє експериментувати з параметрами середовища, перевіряти стійкість алгоритмів керування та вимірювати їхню ефективність на різних апаратних конфігураціях, зокрема на мобільних пристроях. Таким чином, продукт слугує як навчальним інструментом для майбутніх шкіперів, так і прикладом сучасних технологій комп'ютерного моделювання для інженерів-розробників.

1.2 Область застосування

Область застосування створеного симулятора охоплює одразу декілька сегментів.

У першу чергу це освітній напрям – морські та річкові училища, яхтові

школи, центри підготовки судноводіїв, де потрібно відпрацьовувати базові маневри без витрат пального й ризику пошкодження реального плавзасобу. Симулятор доречно інтегрувати в теоретичні курси як інтерактивну лабораторну роботу, доповнюючи заняття з навігації й основ гідродинаміки.

Друга сфера застосування – дослідницька: платформа дає можливість розробникам і фахівцям з комп'ютерної графіки тестувати шейдери води, алгоритми плавучості та методи виявлення зіткнень у контрольованому середовищі; результати можна використовувати для академічних публікацій або оптимізації ігрових рушіїв.

Третій сегмент використання – комерційні й інді-проєкти в галузі мобільних та десктопних ігор: модульна архітектура дозволяє швидко адаптувати ядро симуляції до різних жанрів – від казуальних аркад до серйозних VR-тренажерів портових операцій. Нарешті, продукт корисний як демонстраційний стенд у навчальних лабораторіях ІТ-факультетів: студенти можуть досліджувати патерни проєктування Unity, роботу фізичного рушія та принципи побудови крос-платформних застосунків, що поєднують високонавантажену графіку та інтерактивний UX.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Існуючі симулятори водного транспорту на сучасному ринку цифрових ігор охоплюють широкий спектр жанрів – від аркадних перегонів до повноцінних тренажерів з фізично достовірною гідродинамікою. У межах огляду аналізуватимуться найближчі до обраної теми приклади, що реалізують схожі ідеї або використовують ті самі рушії, зокрема Unity та Unreal Engine. Це дозволяє оцінити переваги та обмеження кожного з підходів, виявити наявні шаблони й визначити нішу для створеного у дипломній роботі симулятора.

Ship Simulator Extremes (VStep)

Це високореалістичний симулятор морських суден, який включає широкий спектр кораблів (від круїзних лайнерів до арктичних криголамів). Реалізована точна фізика, погодні ефекти, справжні географічні локації.

Рисунок 2.1 – Ship Simulator Extremes (VStep)

Ship Simulator Extremes (VStep) – це одна з найвідоміших ігор у жанрі морських симуляторів, створена нідерландською компанією VStep, що спеціалізується також на розробці професійних тренажерів для морських академій. Гра дає змогу користувачам керувати різними типами суден у реалістичних умовах, відчути вплив хвиль, вітру та течій, а також виконувати завдання у складних навігаційних умовах, таких як арктичні води чи порти в умовах шторму. В основу геймплею покладено реалістичну модель гідродинаміки, а рівні базуються на реальних портах і маршрутах.

Гра орієнтована на користувачів, які прагнуть максимального занурення в морське середовище, включаючи як любителів, так і професійних моряків, що використовують подібні симулятори для ознайомлення з типами суден і поведінкою на воді. Водночас вона може використовуватися як демонстраційний інструмент у навчальних закладах. Завдяки підтримці сценаріїв із місіями (наприклад, евакуація пасажирів, боротьба з пожежею), гра також знайомить користувача з критичними ситуаціями, які можуть трапитися на морі.

Переваги – глибока фізика й деталізація.

Недоліки – дуже складний інтерфейс для новачків, а також відсутність мобільної версії.

Boat Master: Parking & Driving Simulator (Unity Mobile)

Аркадний симулятор з паркуванням човнів у порту. Просте керування, зрозумілий інтерфейс, легка вага гри.

Boat Master: Parking & Driving Simulator (Unity Mobile) – це мобільна гра, розроблена на рушії Unity, яка належить до жанру аркадних симуляторів і фокусується на точному паркуванні водного транспорту в обмежених просторах. Гравець отримує контроль над різними типами човнів і повинен пришвартувати їх у порту, уникаючи зіткнень з іншими об'єктами. Основна увага зосереджена не на реалістичній симуляції фізики води, а на виконанні завдань з обмеженим часом, що робить гру швидкою та динамічною.

Гра розрахована насамперед на широку аудиторію мобільних

користувачів, тому має інтуїтивно зрозуміле управління, оптимізовану графіку і не потребує високої продуктивності пристрою. Завдяки легкості освоєння вона є популярною серед гравців, які шукають просту розвагу з елементами навичкової гри, але без ускладнень, пов'язаних із навігацією чи фізикою води. Boat Master зосереджена на поступовому проходженні рівнів з різним рівнем складності, що дозволяє гравцю тренувати просторове мислення та точність керування.

Рисунок 2.2 – Boat Master: Parking & Driving Simulator (Unity Mobile)

Переваги – доступність для мобільних пристроїв.

Недоліки – спрощена фізика, відсутність хвиль і гідродинамічного впливу.

Sailaway – The Sailing Simulator

Мультиплатформна гра для шанувальників вітрильного спорту. Модель динаміки вітру, точне відображення земної кулі через OpenStreetMap.

Sailaway – The Sailing Simulator – це реалістичний симулятор вітрильного спорту, створений для ентузіастів морської навігації, які прагнуть максимально наблизитися до умов справжнього вітрильного плавання. Гра використовує глобальну карту Землі з інтеграцією OpenStreetMap, що дозволяє відтворити реальні океани, узбережжя та порти. Усі вітри, припливи та погодні умови відповідають даним, отриманим із реального часу, що робить ігровий процес ще

більш занурюючим.

Однією з ключових особливостей Sailaway є складна система керування вітрилами, урахування кута нахилу вітру, натягу снастей, зміни напрямку повітряних потоків та взаємодії з хвилями. Гравець має змогу налаштовувати кожен елемент керування судном вручну, що вимагає як базових морехідних знань, так і терпіння для опанування. Гра працює як онлайн-платформа, де гравці можуть організовувати регати або пливти навколо світу у повільному, але максимально реалістичному темпі.

Рисунок 2.3 – Sailaway – The Sailing Simulator

Переваги – реалістичність вітрил і навігації.

Недоліки – складність у засвоєнні, орієнтована на вузьку нішу гравців.

MarineVerse Cup (VR, Mobile)

Симулятор вітрильного спорту у віртуальній реальності з підтримкою мобільних платформ і VR-шоломів.

MarineVerse Cup (VR, Mobile) – це інноваційний симулятор вітрильного спорту, який поєднує геймплей із можливостями віртуальної реальності та

мобільних технологій. Розроблений для платформ VR (наприклад, Meta Quest, HTC Vive) та мобільних пристроїв, він надає користувачам змогу взяти участь у регатах, тренуваннях і навчальних місіях у форматі повного занурення. Завдяки системі контролю, що реагує на рухи гравця, керування вітрилами та стерном відбувається максимально природно.

Гра орієнтована як на початківців, так і на досвідчених вітрильників, пропонуючи широкий спектр змагань і викликів. Освітній компонент MarineVerse Cup дозволяє вивчити основи вітрильного спорту в ігровій формі, що робить її корисною не лише як розвагу, а й як допоміжний інструмент навчання. Завдяки мультиплатформності, користувачі з різних пристроїв можуть змагатися між собою онлайн. Графіка витримана в стилізованому, але приємному стилі, що оптимізує продуктивність без шкоди для атмосфери.

Рисунок 2.4 – MarineVerse Cup (VR, Mobile)

Переваги – сучасна іммерсивність та інтуїтивне керування.

Недоліки – потреба у додатковому обладнанні (VR) для повноцінного досвіду, відсутність фізики зіткнень.

World Ship Simulator (Excalibur Games)

Гра з великим відкритим світом і завданнями для транспортних суден. Можна перевозити вантажі, рятувати людей тощо.

World Ship Simulator (Excalibur Games) – це симулятор морських перевезень із відкритим світом, орієнтований на виконання різноманітних завдань із керуванням цивільними суднами. Гравцеві надається можливість управляти вантажними кораблями, буксирами, рятувальними суднами та іншими видами транспорту. У грі передбачено кар'єрний режим, в якому гравець поступово розвиває власний флот, заробляючи кошти на виконанні місій із перевезення вантажів, евакуації постраждалих, буксирування та швартування в портах.

Світ гри побудований за принципом sandbox – гравець може вільно пересуватися між портами, взаємодіяти з різними типами завдань і модернізувати кораблі. Графіка відповідає середньому рівню свого часу, із досить реалістичним відображенням моря та узбережжя. Однак, попри різноманіття сценаріїв, користувачі часто стикалися з технічними проблемами – зависаннями, помилками в симуляції хвиль та нестабільною роботою системи пошкоджень.

Рисунок 2.5 – World Ship Simulator (Excalibur Games)

Переваги – великий світ і сюжетні місії.

Недоліки – технічні проблеми, критика щодо неякісної оптимізації та слабкої фізики.

Ship Handling Simulator (Android, Free)

Мобільна гра, зосереджена на тренуванні навичок пришвартування у вузькому порту.

Ship Handling Simulator (Android, Free) – це мобільна гра-симулятор, яка орієнтована на навчання базовим навичкам маневрування та швартування суден у складних портових умовах. Гравцеві пропонується керувати різними типами кораблів – від невеликих катерів до великих контейнеровозів – із завданням точно зупинити судно в зазначеній зоні, уникаючи зіткнень і перевищення швидкості.

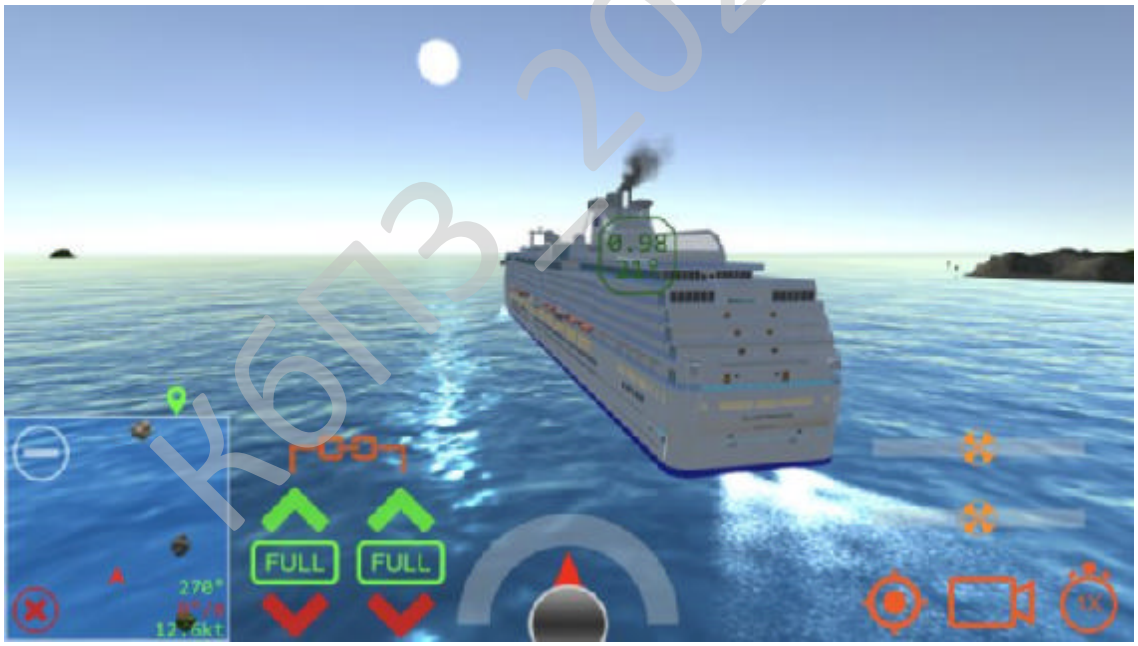

Рисунок 2.6 – Ship Handling Simulator (Android, Free)

Головний акцент гри зроблено на тренування просторового орієнтування та точного контролю у вузьких акваторіях. Це робить її потенційно корисною як для ентузіастів морської справи, так і як ознайомлювальний інструмент для

початківців. Проте, як безкоштовна мобільна гра, вона має значні обмеження в плані графіки, фізичної моделі води та загальної реалістичності. Анімація хвиль, взаємодія корпусу з середовищем і система пошкоджень реалізовані спрощено, що знижує загальний рівень занурення.

Переваги – корисність для базового навчання маневрування.

Недоліки – відсутність графічної деталізації, неякісна анімація хвиль, спрощене керування.

Таким чином, аналіз показує, що попри велику кількість водних ігор, вони здебільшого належать до двох полярних типів: або складні професійні симулятори для вузьких задач, або аркадні додатки з обмеженою фізикою. Існує дефіцит середньої категорії – навчально-ігрових симуляторів із спрощеним, але реалістичним відчуттям динаміки руху, які можна запускати як на ПК, так і на мобільних платформах. Створена в межах цієї дипломної роботи гра може заповнити цю нішу, запропонувавши фізичну модель хвиль, пошкоджуваність корпусу та змагальний елемент у компактному та доступному форматі.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Вибір технологічного стека для реалізації симулятора визначався сукупністю технічних, організаційних та економічних міркувань. У ролі основної мови програмування було обрано **C# 12**. Ця мова входить до стандартного набору інструментів рушія Unity, має сувору статичну типізацію та сучасні засоби підвищення безпеки коду – nullable-аналіз, records, pattern-matching, статичні інтерфейсні методи. Порівняно з C++ C# знижує ризик помилок керування пам’яттю й скорочує час розробки, а проти інтерпретованих мов на кшталт Python забезпечує помітно вищу продуктивність завдяки JIT-компіляції у віртуальній машині .NET CLR. Додатковою перевагою є багата екосистема бібліотек: LINQ спрощує обробку колекцій при підрахунку статистики, а

доступність NuGet-пакетів дозволяє без зусиль інтегрувати JSON-серіалізацію або алгоритми штучного інтелекту.

Як середовище розробки вибрано **Unity 2025 LTS** – довгострокову підтримувану редакцію, що гарантує стабільність API упродовж щонайменше двох років. Unity надає готові модулі 3-D фізики (PhysX 5), систему частинок, пост-процесинг, а також Scriptable Render Pipeline, що дає змогу точно контролювати графічний конвеєр. Для симулятора хвиль використано компонент CustomRenderTexture, який виконує чисельну інтеграцію прямо на GPU й істотно розвантажує процесор. Об'єкти сцени описуються через prefab-підхід, що спрощує повторне використання конфігурацій пірсів та буйків. Вбудований пакет **Input System 1.8** дозволяє в єдиному коді підтримати клавіатуру, ґеймпад і сенсорний джойстик, чим вирішується задача мультиплатформності.

Візуальна частина побудована на **Universal Render Pipeline**. URP забезпечує баланс між якістю та продуктивністю: на ПК активується фізично-коректне освітлення з тінями, а на мобільних профіль «Mobile Optimized» відключає дорогоцінні ефекти (Screen Space Shadows, HDR), що дозволяє утримувати 40–60 fps навіть на пристроях середнього класу. Для low-poly-моделей використано Blender 4.1 LTS – безплатний редактор із експортом у формат FBX, який підтримується Unity без конвертації.

У ролі середовища кодування застосовано **JetBrains Rider 2024.1**, оскільки цей IDE інтегрується з Unity Editor через плагін, підтримує перевірку коду на льоту й має профайлер пам'яті CLR. На альтернативних робочих місцях використовується Visual Studio 2022 Community, що важливо для сумісності в команді. Система контролю версій **Git** у поєднанні з GitHub Actions забезпечує безперервну інтеграцію: кожен пуш у гілку main автоматично збирає проєкт під Windows і Android, запускає юніт-тести та завантажує артефакти релізу.

Аудіо обробляється вбудованими компонентами AudioSource/Audioclip; для майбутніх розширень передбачена інтеграція FMOD Studio, але на етапі дипломної роботи базових можливостей Unity достатньо. Збереження прогресу

виконується через JSON-серіалізацію у **PlayerPrefs**, що не потребує зовнішніх баз і працює однаково на всіх цільових ОС. У перспективі онлайн-лідерборди можуть бути перенесені у Firebase Realtime DB без зміни ядра застосунку.

Таким чином, обраний стек дозволяє розробити фізично достовірний, візуально привабливий і водночас легкий за системними вимогами симулятор, який можна розгорнути на настільних платформах і мобільних пристроях, не дублюючи код та не залучаючи дорогі пропрієтарні бібліотеки.

2.3 Розгорнута постановка завдання

Метою роботи є створення програмного забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

У рамках обраної теми кваліфікаційної роботи було вирішено розробляти симулятор морського човна.

Основна задача: реалізувати багатоплатформну комп’ютерну гру-симулятор «Sea Adventures» (тренажер швартування маломірного судна). Використати ігровий рушій Unity 2025 LTS та мову скриптів – C# 12. Нербхідно створити інтерактивний тренажер, який відтворює маневрування човна на закритій акваторії з достовірною поведінкою водної поверхні та зіткнень. Програма має одночасно виконувати навчальну, дослідницьку й розважальну функції, забезпечуючи користувача стислими сесіями тривалістю від 1 до 5 хвилин із підсумковою оцінкою точності швартування й безпечності проходження маршруту.

Платформи та цільові середовища:

- **ПК** – Windows 10/11 ×64, macOS 12+, Ubuntu 22.04+ зі сховищем Vulkan/Metal.
- **Мобільні пристрої** – Android 11+, iOS 15+ (ARM64). Збірки повинні формуватися однією гілкою коду без ручної модифікації сцен.

Основний ігровий процес:

1. Завантаження рівня, відмальовування стартової камери й HUD.

2. Прийом керових впливів (клавіатура / ґеймпад / сенсорний джойстик).

3. Розрахунок сил тяги, крутного моменту, опору води – 50 ітерацій / с.

4. GPU-симуляція хвиль – 5 ітерацій на кадр; розрізнена сітка 64 × 64 на ПК та 32 × 32 на мобільних.

5. Обробка зіткнень човна з пірсами, буями, береговою лінією, нарахування пошкоджень.

6. Фіксація успішного входу у док-зону, вивід статистики, запис рекорду.

7. Вибір «Повторити» або «Наступний рівень».

Функціональні вимоги:

– Система хвиль: нелінійна висотна карта, локальні збурення, шлейф за кормою.

– Динаміка човна: інерція, реверсивний хід, максимальна швидкість 12 вузлів.

– Пошкодження: зниження HP при ударах > 1 м/с; критичне руйнування ≤ 0 HP.

– HUD: таймер ± 0,1 с, очки, шкала корпусу, підказки першого запуску.

– Мультиплатформний ввід: підтримати XInput-ґеймпад і джойстик MobileJoystick.

– Збереження прогресу: JSON + PlayerPrefs, окремий файл progress.json.

– Меню налаштувань: якість графіки, гучність, інверсія керма, скидання статистики.

– Автодетект продуктивності: на мобільних примусово пресет «Mobile Optimized».

– Двомовність інтерфейсу (uk/en) через Unity Localization.

– Експорт логів маневрування у CSV для подальшого аналізу.

Нефункціональні вимоги:

– Час завантаження сцени ≤ 7 с на ПК і ≤ 12 с на mid-range Android.

– Кадрова частота: 60 fps на ПК, 40 fps на цільових телефонах.

– Споживання пам’яті: < 1,2 ГБ RAM на ПК, < 700 МБ на мобільних.

– Енергоспоживання мобільної збірки: не перевищувати 30 % CPU-ядра й 40 % GPU за середнього пресету.

– Відмова-безпечність: будь-який виняток у скриптах не повинен приводити до аварійного завершення застосунку; перед виходом усе аудіо зупиняється, дані зберігаються.

Архітектура та модулі:

- **WaterSimulation** – GPU-шейдери (CustomRenderTexture), менеджер зон оновлення.
- **BoatController / DamageSystem / ImpactSound** – фізика й колізії судна.
- **GameManager / DockTrigger** – таймер, очки, логіка кінця сесії.
- **InputAdapter / MobileJoystick** – уніфікований ввід.
- **SoundManager** – обгортка навколо двох AudioSource, стрічка FMOD опціонально.
- **SaveLoadSystem** – серіалізація рекордів; шифрування AES-128 для iOS Keychain.
- **SettingsMenu** – Canvas-екран, прив’язаний до ScriptableObject-налаштувань.
- **PlatformQualityManager** – одноразова оптимізація графічних параметрів.

Інструментальні засоби: Unity 2025 LTS, URP 2025.1, Rider 2024.1 або Visual Studio 2022, Blender 4.1 для low-poly-моделей, GitHub Actions для CI-зб

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Сформулюємо призначення програми – симулятор швартування маломірного судна, реалізований у середовищі Unity 2025 LTS, який демонструє інтерактивну взаємодію користувача з тривимірним середовищем на різних апаратних платформах (ПК / Android / iOS).

Загальний життєвий цикл розробленого програмного забезпечення:

– *Етап ініціалізації*. Після запуску рушій завантажує стартову сцену, створює головне вікно, ініціалізує синглтони (SaveLoadSystem, SoundManager, InputAdapter) та підтягує останній набір рекордів. Паралельно PlatformQualityManager перевіряє тип пристрою й корегує параметри графіки.

– *Головний цикл гри*. Кожен кадр Unity послідовно викликає Update() скриптів керування (InputAdapter, GameManager, BoatController) і FixedUpdate() фізичних модулів. Отримані сили та моменти передаються в Rigidbody човна, після чого LateUpdate() камер відображає оновлену сцену. GPU-шейдер WaterSimulation генерує хвилі асинхронно, без додаткового навантаження на процесор.

– *Обробка подій*. Коли колайдер човна перетинає зону DockTrigger, GameManager завершує рівень, розраховує бали й час, порівнює їх із ProgressData та, за потреби, викликає SaveLoadSystem.Save. Зіткнення корпуса з об’єктами середовища фіксуються ImpactSound; сигнал передається SoundManager-у й DamageSystem, який знижує значення HP.

– *Взаємодія з користувачем*. HUD щоразу отримує від GameManager поточний час, кількість очок і відсоток ушкоджень; SettingsMenu дає змогу змінити гучність, графічний пресет і скинути прогрес.

– *Завершення сесії*. Користувач повертається в головне меню або закриває

застосунок. Перед виходом PlayerPrefs зберігаються, а SoundManager коректно відпускає ресурси аудіо-потоків.

Нижче наведено основні розроблені в ході реалізації кваліфікаційної роботи скрипти для симуляції морського човна.

WaterSimulation.cs

Цей скрипт відповідає за візуальне й фізичне відтворення хвильової поверхні у грі. За допомогою компонента CustomRenderTexture на графічному процесорі виконується чисельна ітераційна симуляція хвиль. На кожному кадрі рушій очищає зони оновлення, задає базову зону для безперервної еволюції й, за потреби, локальну точку збурення під курсором. Завдяки цьому розробник отримує інтерактивну водну поверхню з мінімальними витратами CPU-часу. У методі Start() додатково відбувається ініціалізація текстури та завантаження структури ProgressData, що дає сценарію доступ до рекордів гравця безпосередньо з моменту запуску сцени.

BoatController.cs

BoatController інкапсулює рух човна. Компонент отримує linейний та кутовий ввід із InputAdapter і переводить його у фізичні дії над Rigidbody: лінійний поштовх уперед/назад та крутний момент для повороту навколо осі Y. Прискорення й обертання налаштовуються публічними полями thrustForce і turnTorque. Скрипт від'єднує механіку керування від конкретної платформи, зберігаючи однаковий алгоритм руху для ПК, консолей і мобільних пристроїв.

DockTrigger.cs

DockTrigger – це колайдер-тригер, розміщений у зоні швартування. Коли об’єкт із тегом Player (човен) входить у тригер, скрипт викликає метод OnBoatDocked у GameManager, тим самим сигналізуючи про завершення рівня. Після цього виконується запис результатів через SaveLoadSystem.UpdateProgress: до файлу прогресу потрапляють ідентифікатор рівня, час проходження та рівень пошкоджень корпусу.

GameManager.cs

GameManager є центральним диспетчером сесії. Він веде поточний таймер, підраховує бали, оновлює текстові поля Canvas і зберігає рекорди. Після старту сцени завантажує ProgressData, ініціалізує змінні та у циклі Update нарощує час. Метод FinishRun викликається DockTrigger-ом; він порівнює досягнення гравця з існуючими рекордами, за потреби оновлює їх і зберігає через SaveLoadSystem. Додатковий метод AddScore дозволяє іншим системам (наприклад, чекпоінтам) інкрементувати очки й негайно відбивати це в інтерфейсі.

CameraFollow.cs

CameraFollow реалізує плавне стеження камери за човном без використання Cinemachine. На кожному кадрі у LateUpdate він обчислює бажану позицію камери як суму позиції цілі й вектора offset, після чого інтерполює координати методом Lerp із заданою швидкістю smoothSpeed. Додатковий виклик LookAt гарантує, що об’єктив завжди спрямований на човен, забезпечуючи гравцеві стабільний огляд.

SoundManager.cs

SoundManager – глобальний аудіо-сервіс, який тримає дві основні аудіодоріжки: звук двигуна та звук зіткнення. Через патерн Singleton він доступний із будь-якого скрипту. Метод SetEnginePitch змінює тон і гучність мотору залежно від дроселя, а PlayImpact відтворює звук удару з корекцією гучності та висоти відповідно до сили зіткнення.

ImpactSound.cs

Цей компонент навішується на колайдери човна. Під час події OnCollisionEnter він вимірює відносну швидкість зіткнення, нормує її та передає отриману величину у SoundManager і DamageSystem. Завдяки цьому гра синхронізує аудіо-фідбек із механічними ушкодженнями корпусу.

DamageSystem.cs

DamageSystem відстежує цілісність човна. Він зберігає поточні очки

здоров’я IHP, дозволяє застосувати пошкодження методом ApplyDamage і сповіщає GameManager про зміну відсотка HP, щоб оновити індикатори на екрані. Реалізований як Singleton для спрощеного доступу з інших скриптів.

LevelManager.cs

LevelManager зберігає статичні параметри конкретної сцени. У поточній реалізації головним є вектор напрямку хвиль напрямХвиль, який може зчитувати WaterSimulation або інші системи, щоб синхронізувати поведінку середовища з дизайном рівня. Об’єкт позначає себе як Current у Awake, забезпечуючи глобальний доступ.

InputAdapter.cs

InputAdapter абстрагує різницю між Legacy Input та Unity Input System, а також між десктопами й мобільними платформами. У режимі мобільного пристрою скрипт читає координати з MobileJoystick; на ПК він повертає значення осей Horizontal і Vertical. Завдяки цьому BoatController отримує єдине API для керування, незалежно від типу пристрою.

SettingsMenu.cs

SettingsMenu керує застосуванням налаштувань якості графіки та гучності звуку. Через методи SetQuality і SetVolume він напряму викликає QualitySettings і AudioListener, а в Start планується завантаження збережених параметрів із PlayerPrefs, що забезпечує збереження призначеної користувачем конфігурації між сесіями.

SaveLoadSystem.cs

SaveLoadSystem надає повний цикл роботи з прогресом гравця. Структура ProgressData серіалізується у JSON-рядок і зберігається у PlayerPrefs під ключем HarborPilot_Progress_v1. Методи Load і Save відповідають за базові операції, UpdateProgress – за оновлення рекордів після проходження рівня, а ResetProgress – за повне скидання даних. Таким чином, система гарантує надійне збереження статистики на всіх платформах.

MobileJoystick.cs

MobileJoystick реалізує екранний контролер для сенсорних пристроїв. Скрипт обробляє події перетягування, дотику й відпускання тач-панелі, переміщує ручку джойстика в межах кола радіусом maxRange, обчислює нормалізований вектор та зберігає його у властивості Input. InputAdapter читає ці значення, щоб перетворити жести користувача на вектор керування човном.

3.2 Розробка структурної схеми

На рис. 3.1 подано структурну схему системи симулятора управління морським човном.

Рисунок 3.1 – Структурна схема системи

У верхній частині зображено апаратно-програмний ланцюг виведення графіки: відеоадаптер, його драйвер і модуль процесорної взаємодії. Відеоадаптер забезпечує безпосередній рендеринг кадрів на дисплей, драйвер

відеоадаптера реалізує API низького рівня (DirectX, Vulkan), а модуль процесорної взаємодії слугує посередником між ігровою логікою та графічним підсистемами, передаючи підготовлені масиви вершин, текстур і команд рендерингу.

Нижче, у пунктирній рамці, розташовано власне програмний комплекс симулятора. Він складається з кількох функціональних блоків і двох баз даних ресурсів. База асетів містить 3D моделі, аудіофайли та конфігураційні скрипт-об’єкти; база текстур і шейдерів зберігає картографічні дані для матеріалів і коди GPU-шардерів. Дані ресурси завантажуються під час ініціалізації рівня та стають спільним підґрунтям для всіх ігрових підсистем.

Блок «Скрипти симуляції води» реалізує чисельний розв’язок рівнянь хвиль і формує динамічну висотну карту поверхні. Ці дані передаються як у графічний пайплайн (для візуалізації хвиль), так і до фізичних модулів, що відповідають за плавучість і взаємодію об’єктів із рідиною.

Блок «Скрипти симуляції та управління для човна» об’єднує механіку руху, обробку користувацького вводу й систему пошкоджень. Він отримує керівні впливи від периферійних пристроїв через Input-підсистему, трансформує їх у сили й моменти, прикладені до фізичної моделі човна, а також фіксує зіткнення та їх наслідки.

Блок «Скрипти симуляції перешкод та обробки зіткнень» відповідає за статичні й динамічні об’єкти оточення (пірси, буї, інші судна). Він взаємодіє із водой та корпусом човна, генеруючи події колізій і сповіщаючи про них модуль пошкоджень.

Нарешті, блок «Скрипти інтерфейсу користувача» забезпечує відображення часу, очок, стану здоров’я та системного меню. Він одержує з GameManager актуальні значення метрик і видає їх на Canvas, а також пересилає команди з меню до інших підсистем (зміна налаштувань, перезапуск рівня тощо).

Таким чином, схема демонструє чітку ієрархію: апаратний рівень графічного виводу, узагальнений модуль взаємодії процесора з GPU та

внутрішню структуру симулятора, де ресурси й окремі скриптові підсистеми координуються для досягнення цілісної, інтерактивної поведінки.

3.3 Розробка функціональної схеми

На рис. 3.2 наведено функціональну схему розробленого програмного забезпечення.

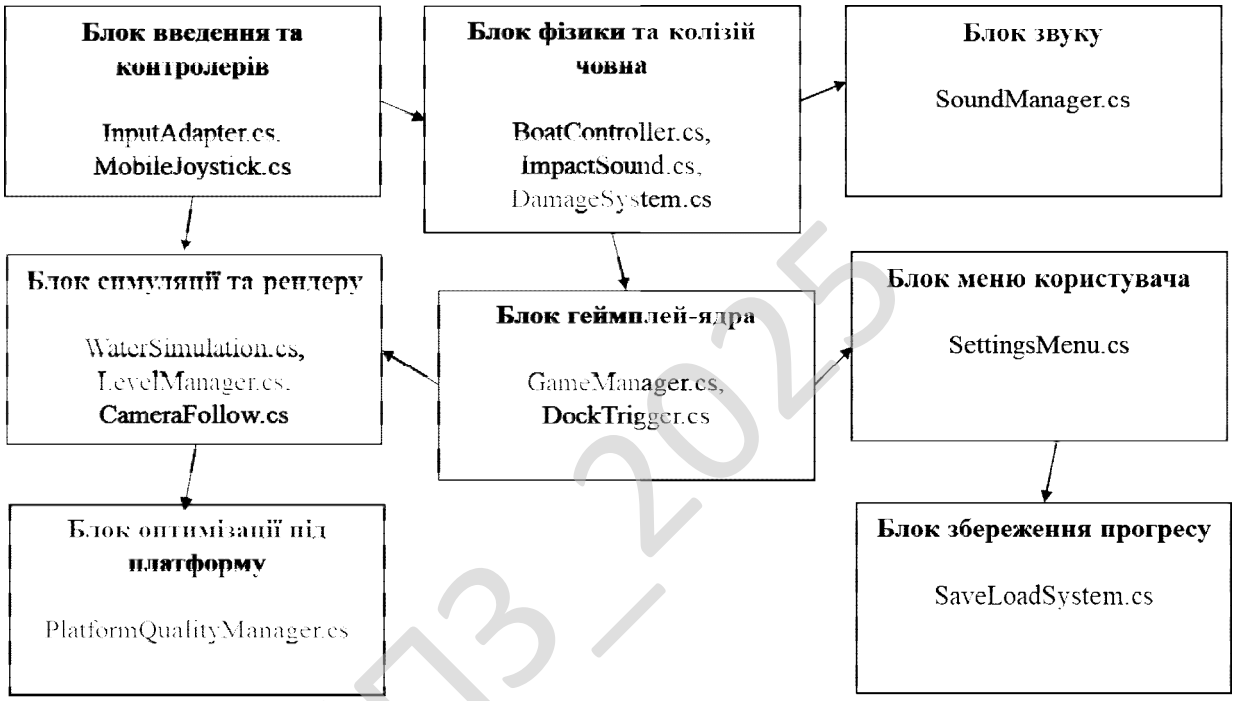

Рисунок 3.2 – Функціональна схема системи

Усі скрипти згруповано у взаємодіючі логічні блоки, що відповідають за окремі аспекти роботи застосунку.

Блок введення та контролерів забезпечує уніфіковане захоплення керувальних сигналів з різних пристроїв. Скрипт InputAdapter інкапсулює відмінності між клавіатурою, ґеймпадом і сенсорним екраном, тоді як MobileJoystick формує цифровий аналог джойстика для мобільних платформ. Отримані осі надсилаються до двох незалежних підсистем – симуляції/рендеру та фізики човна.

Блок симуляції та рендеру обробляє всі обчислення, пов’язані з середовищем. WaterSimulation формує висотну карту хвиль, LevelManager надає параметри конкретного рівня (розмір, напрям течії), а CameraFollow реалізує кінематичне стеження за судном. Результатом роботи блоку є готові дані для GPU-рендерингу та фізичної взаємодії.

Блок фізики та колізій човна безпосередньо керує переміщенням об’єкта Rigidbody. BoatController перетворює осі вводу на сили й моменти, ImpactSound фіксує удари корпусу й генерує аудіосигнали, а DamageSystem накопичує пошкодження, що впливають на стан геймплею. Для звукового оформлення перетин із блоком SoundManager здійснюється шляхом викликів відповідних методів синглтона.

Центральним елементом є блок геймплей-ядра. GameManager веде таймер, підраховує очки, контролює стан сесії та ініціює збереження рекордів. DockTrigger на рівні сцени сигналізує про успішне швартування, що замикає цикл гри. З боку інтерфейсу користувача GameManager взаємодіє з SettingsMenu, завдяки чому меню може змінювати параметри геймплею або запускати скидання прогресу.

Блок збереження прогресу складається зі скрипта SaveLoadSystem, який серіалізує структуру ProgressData у форматі JSON і зберігає її у PlayerPrefs. Він отримує дані від геймплей-ядра та повертає їх при старті нової сесії, забезпечуючи неперервність гри.

Блок оптимізації під платформу (PlatformQualityManager) активується при старті й адаптує налаштування графіки та щільність сітки хвиль до можливостей девайса. Цей модуль не бере участі у гостьовій логіці, але впливає на продуктивність блоку симуляції/рендеру.

Таким чином, схема демонструє розподіл обов’язків між висхідним потоком даних (ввід → фізика → геймплей → UI та збереження) та низхідними сервісами (оптимізація, рендер, звук). Чіткі інтерфейси між блоками мінімізують залежності та полегшують супровід проєкту.

3.4 Розробка діаграми процесів

Діаграма процесів (рис. 3.3) відображає послідовність подій, які відбуваються під час одного сеансу роботи симулятора човна.

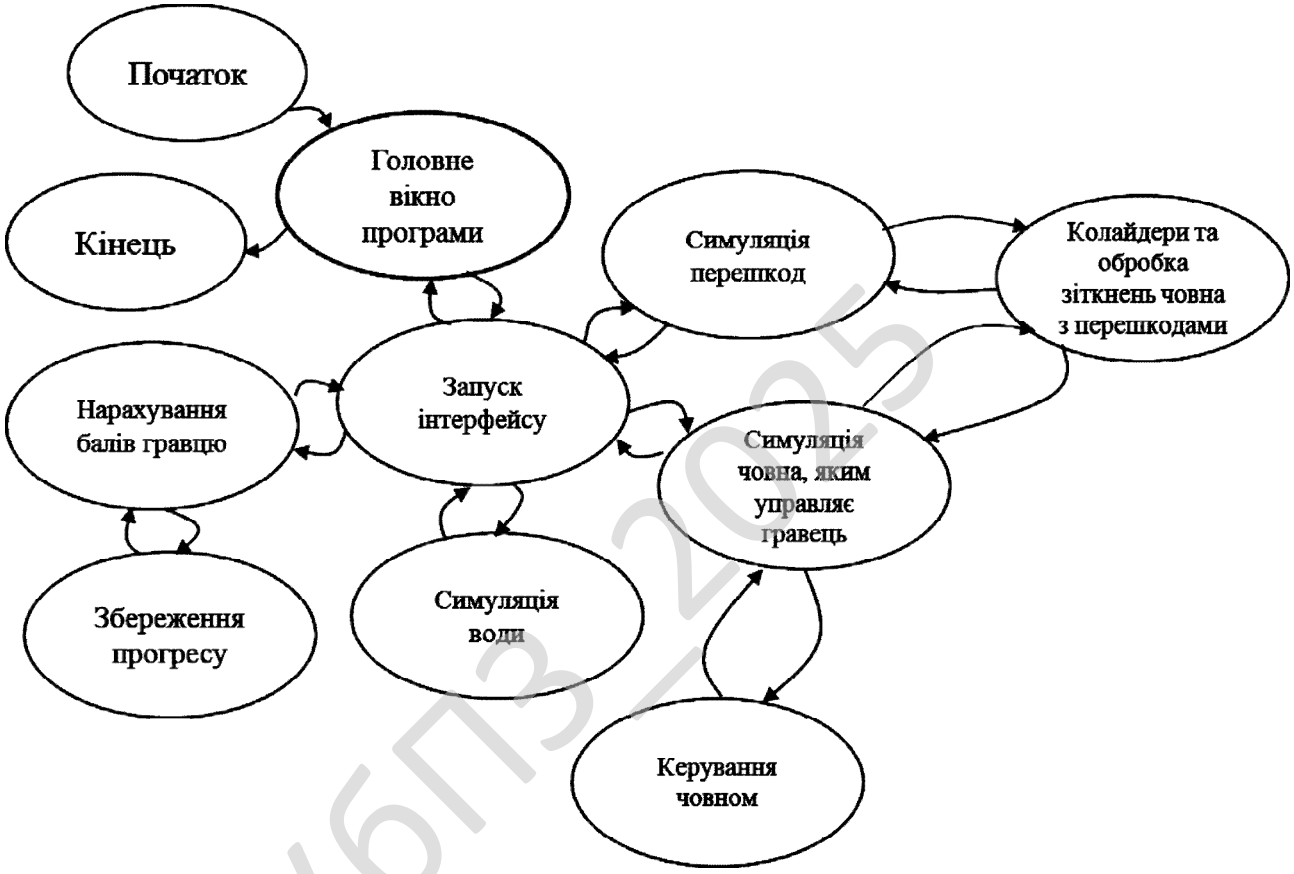

Рисунок 3.3 – Діаграма процесів системи

Процес починається з вузла «Початок», що відповідає запуску застосунку та ініціалізації рушія Unity. Далі керування переходить до «Головного вікна програми», де виконуються завантаження необхідних ресурсів сцени, перевірка параметрів обладнання та відображення стартового меню.

З головного вікна користувач активує «Запуск інтерфейсу». На цьому етапі створюються об’єкти Canvas, ініціалізуються лічильники часу й очок, а також відображаються елементи HUD. Паралельно інтерфейсний модуль ініціює

три фізично-графічні підпроцеси: «Симуляція води», «Симуляція човна, яким управляє гравець» та «Симуляція перешкод».

«Симуляція води» безперервно розраховує висотну карту хвиль і передає дані в GPU, забезпечуючи візуальну достовірність та коректну взаємодію тіл з рідиною. «Симуляція човна» приймає керувальні дії з підпроцесу «Керування човном», де InputAdapter трансформує сигнали з клавіатури, ґеймпада чи мобільного джойстика у вектори тяги та повороту. Внаслідок цього BoatController змінює стан Rigidbody, а оновлені координати повертаються до графічного рендеру й інтерфейсу користувача.

«Симуляція перешкод» генерує рухомі або статичні об’єкти навколишнього середовища. Коли трапляється геометричне перетинання колайдера човна з колайдером перешкоди, спрацьовує підпроцес «Колайдери та обробка зіткнень човна з перешкодами». Він розраховує силу удару, формує аудіоефект і передає величину пошкоджень до DamageSystem, що змінює показник стану корпусу.

Після кожного кадру GameManager аналізує події, отримані з фізичних модулів, і виконує «Нарахування балів гравцю». За безпечне проходження секторів або збирання бонусів підвищується рахунок; за зіткнення чи перевитрату часу можливі штрафи. Оновлені значення передаються у блок інтерфейсу та в «Збереження прогресу», де SaveLoadSystem порівнює їх із рекордними й, за потреби, зберігає у PlayerPrefs.

Зворотний шлях від «Нарахування балів» до «Запуску інтерфейсу» підтримує актуалізацію HUD у реальному часі. Коли користувач успішно швартує судно, DockTrigger надсилає сигнал GameManager-у про завершення рівня; останній завершує підпроцеси, викликає заключне збереження й передає керування до «Кінець», що закриває сцену або повертає програму у головне меню.

Таким чином, кожен еліпс діаграми репрезентує окремий функціональний підпроцес, а спрямовані дуги показують, яка послідовність викликів або обміну

даними відбувається між компонентами під час життєвого циклу гри. Схема наочно ілюструє, як вводи користувача, графічно-фізичні обчислення та зберігання статистики утворюють замкнений, інтерактивний цикл гри.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рис. 4.1 наведена блок-схема основної програми. На рисунку представлено блок-схему основного алгоритму роботи комп'ютерної гри в жанрі симулятора човна, створеної на основі рушія Unity.

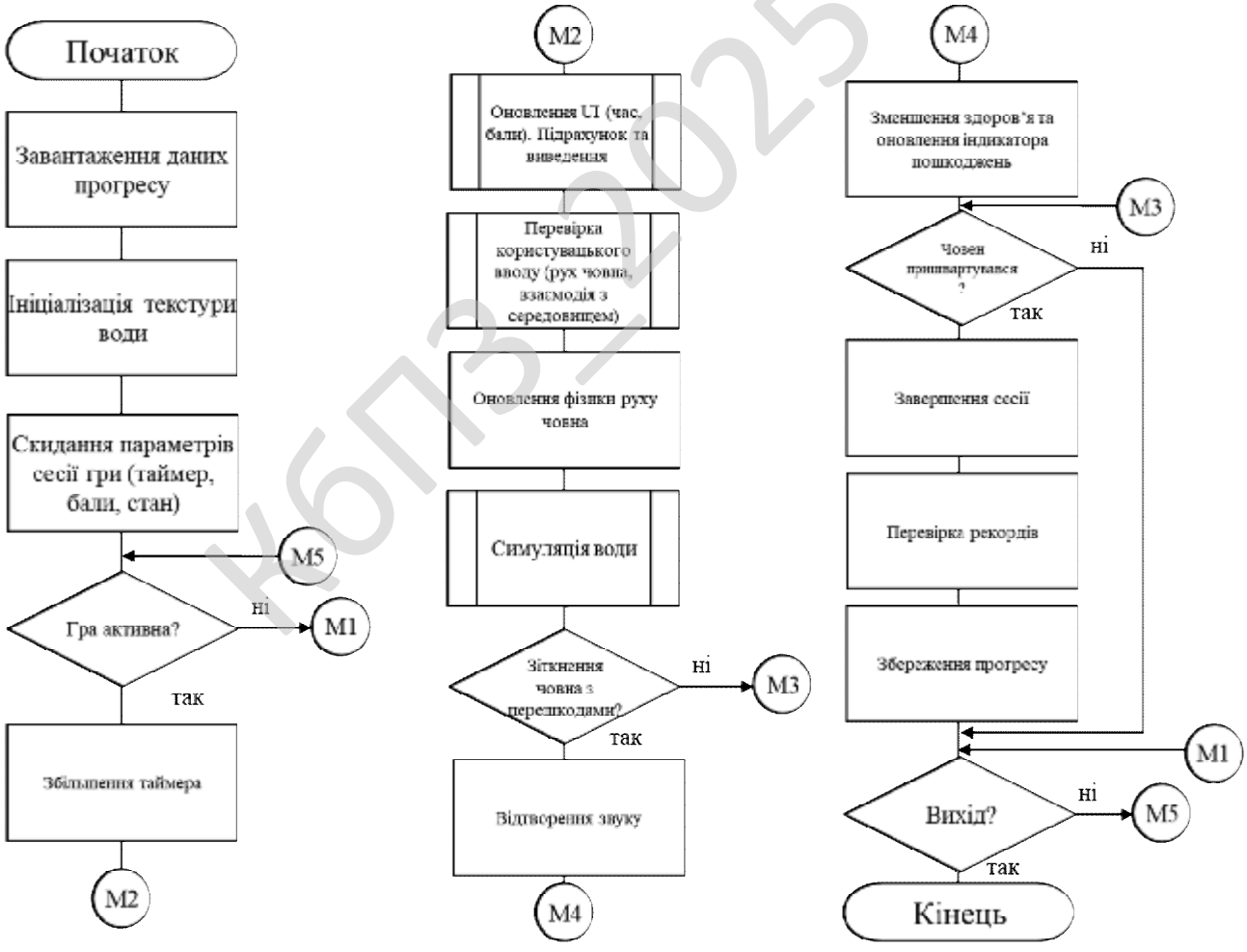

Рисунок 4.1 – Блок-схема роботи основної програми

Схема демонструє логічну послідовність процесів, які виконуються під час запуску, гри та завершення ігрової сесії. Умовно вона поділена на три вертикальні гілки, які взаємодіють між собою через умовні переходи (ромби) та мітки переходів (M1–M5).

Процес розпочинається з початкової ініціалізації, яка включає завантаження даних прогресу гравця та ініціалізацію хвильової текстури води. Далі виконується скидання параметрів сесії – таймера, балів і стану гри. Після цього здійснюється перевірка, чи активна гра. Якщо так – запускається цикл оновлення.

Цикл гри включає збільшення таймера, оновлення UI (вивід часу та балів), а також обробку вводу користувача — це може бути керування човном або взаємодія з водною поверхнею. Далі відбувається оновлення фізики човна згідно з введеними параметрами руху і симуляція води.

Після кожного кадру перевіряється, чи відбулося зіткнення човна з перешкодами. Якщо так – запускається відтворення звуку зіткнення та зменшується запас здоров’я човна, оновлюється відповідний індикатор пошкоджень.

Окремим блоком є обробка події швартування човна. Якщо гравець успішно пришвартувався, то сесія завершується, система перевіряє, чи було побито рекорди, після чого зберігає новий прогрес.

У фінальному етапі перевіряється, чи вирішив гравець вийти з гри. Якщо ні – процес повертається до початку циклу оновлення. Якщо так – завершується робота всієї програми.

Таким чином, схема охоплює основні аспекти функціонування гри: від завантаження даних і управління фізикою до збереження результатів та завершення сесії, що дозволяє ефективно візуалізувати логіку програмного забезпечення симулятора.

Блок-схема на рисунку 4.2 деталізує роботу підпрограми, що відповідає за водну частину симуляції.

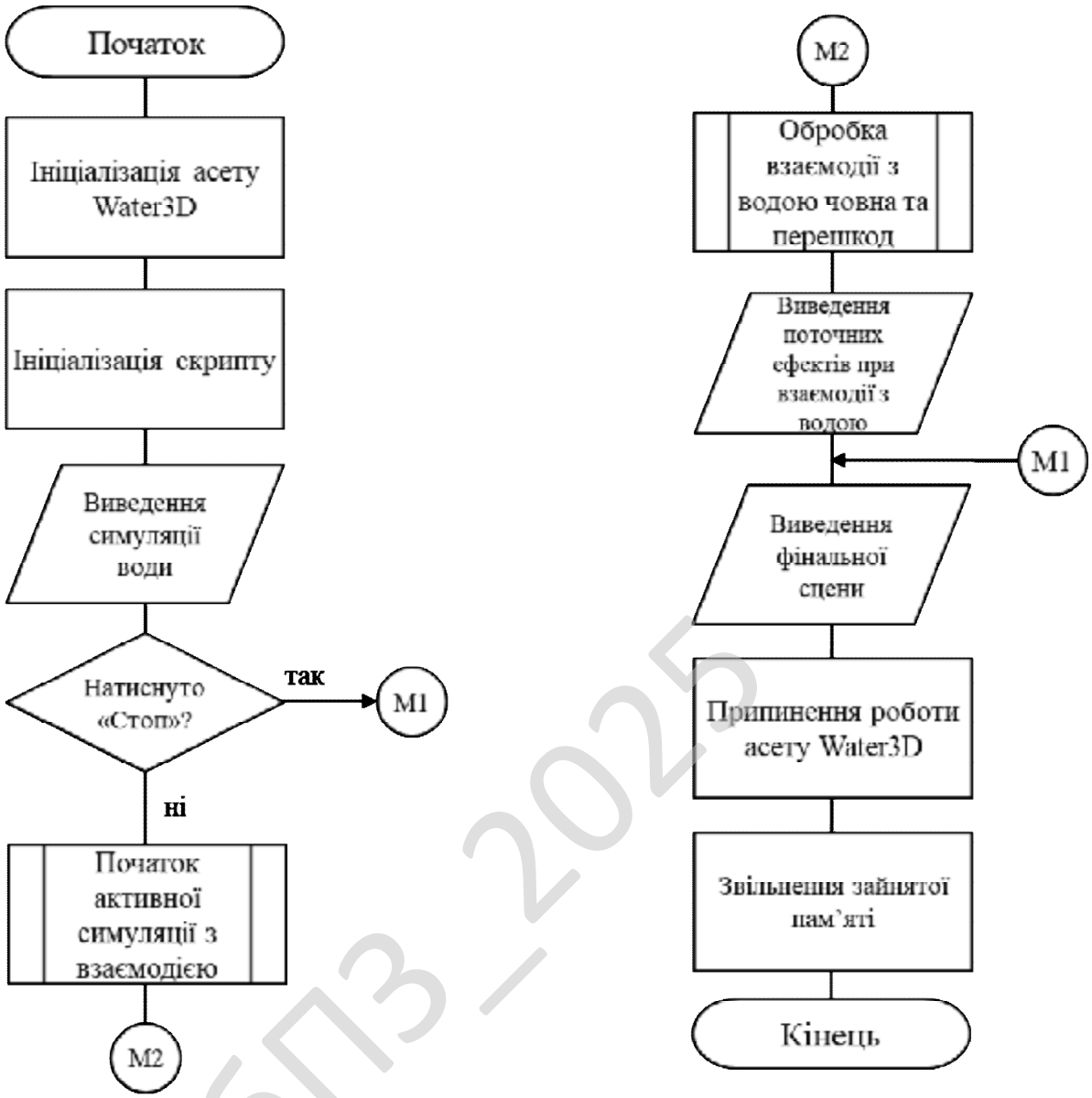

Рисунок 4.2 – Блок-схема підпрограми симуляції води

Після входу у вузол «Початок» рушій виконує ініціалізацію графічного асета Water3D, який містить шейдери, текстури висот та параметри хвиль. Далі створюється і налаштовується керівний скрипт – він реєструє зони оновлення текстури і підписується на події фізичного рушія. Коли обидва об’єкти підготовлено, програма вперше відмальовує спокійну поверхню – користувач бачить статичну сцену без взаємодії.

У цей момент можливе натискання службової кнопки «Стоп». Якщо оператор зупиняє процес, керування переходить через мітку M1 відразу до секції

завершення; якщо ні, активується цикл інтерактивної симуляції. На етапі «Початок активної симуляції з взаємодією» скрипт перемикає Water3D у режим безперервного розрахунку і переходить до мітки M2.

У межах програми за міткою M2 виконується обробка колізій: систему хвиль інформують тригери човна та перешкод, у результаті чого генеруються сплески, хвилі від гідродинамічного гвинта й оббурт води навколо опор. Одночасно рендерер виводить у режимі реального часу дрібні візуальні ефекти – наприклад, бризки й пінний шлейф за кормою. Після того як користувач завершує навігацію або натискає «Стоп», керування знову проходить мітку M1 і відбувається відтворення фінальної статичної сцени.

Заключна секція полягає у коректному завершенні роботи асета Water3D: обчислення зупиняються, GPU-ресурси звільняються, буфери висот очищуються. Після деалокації зайнятої оперативної пам’яті програма переходить до вузла «Кінець», що завершує життєвий цикл водної підсистеми. Таким чином, схема показує два режими функціонування – попередній перегляд і повна інтерактивна симуляція – а також демонструє, як управляються ресурси під час запуску й зупинки.

Розглянемо основні скрипти розробленого програмного забезпечення.

Скрипт симуляції води:

```
using UnityEngine;                  // Підключаємо базовий простір назв Unity

/// <summary>
/// Симуляція хвильової текстури за допомогою CustomRenderTexture.
/// Клік ЛКМ – позитивний сплеск, ПКМ – негативний.
/// </summary>
public class WaterSimulation : MonoBehaviour
{
    [SerializeField]                  // Посилання на текстуру, що рендериться
шейдером

 private CustomRenderTexture waterTexture;

    [SerializeField, Range(1, 20)]    // Скільки ітерацій симулятора виконувати
за кадр
    private int iterationsPerFrame = 5;

    private ProgressData _progress;          // рекорди гравця (завантажимо у
Start)

    private void Start()
```

```
    {

     _progress = SaveLoadSystem.LoadOrCreateDefault();  // завантаження
прогресу
     waterTexture.Initialize();   // ініціалізуємо текстуру перед першим
оновленням
    }

    private void Update()
    {
        waterTexture.ClearUpdateZones(); // очищаємо попередні зони оновлення
        UpdateInteractionZones();        // додаємо зони за користувацьким
ввідо
        waterTexture.Update(iterationsPerFrame); // запускаємо обчислення
шейдера
    }

    /// <summary>
    /// Створює дві зони оновлення:
    /// 1) уся текстура (для звичайної еволюції);
    /// 2) крихітна точка під курсором – для сплеску/ямки.
    /// </summary>
    private void UpdateInteractionZones()
    {
        bool leftClick  = Input.GetMouseButton(0);
        bool rightClick = Input.GetMouseButton(1);
        if (!leftClick && !rightClick) return;

        if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition),
out RaycastHit hit))
        {
            // Зона 0: уся поверхня хвиль
            var fullZone = new CustomRenderTextureUpdateZone
            {
                needSwap         = true,
                passIndex        = 0,   // основний пас шейдера
                rotation         = 0f,
                updateZoneCenter  = new Vector2(0.5f, 0.5f),
                updateZoneSize    = Vector2.one
            };

            // Зона 1: точка під курсором (пас 1 – підняти, пас 2 – опустити)
            var clickZone = new CustomRenderTextureUpdateZone
            {
                needSwap          = true,
                passIndex         = leftClick ? 1 : 2,
                rotation          = 0f,
                updateZoneCenter  = new Vector2(hit.textureCoord.x, 1f -
hit.textureCoord.y),
                updateZoneSize    = new Vector2(0.01f, 0.01f)
            };


    waterTexture.SetUpdateZones(new[] { fullZone, clickZone });
        }
    }
}
```

Скрипт керування човном:

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class BoatController : MonoBehaviour
{
    [Header("Параметри руху")]
    public float thrustForce = 450f;      // Сила лінійного прискорення
    public float turnTorque  = 90f;       // Крутний момент для обертання

    private Rigidbody rb;
    private Vector2   inputVector;

    private void Awake() => rb = GetComponent<Rigidbody>();

    private void Update()
    {
        // Варіант управління для ПК
        //inputVector.x = Input.GetAxis("Horizontal"); // A/D або ←/→
       // inputVector.y = Input.GetAxis("Vertical");   // W/S або ↑/↓

          // Варіант управління для мобільного телефону
         float steer    = InputAdapter.Instance.GetSteer();
         float throttle = InputAdapter.Instance.GetThrottle();
    }

    private void FixedUpdate()
    {
        // Рух уперед/назад
        Vector3 force = transform.forward * inputVector.y * thrustForce;
        rb.AddForce(force, ForceMode.Force);

        // Поворот навколо осі Y
        float torque = inputVector.x * turnTorque;
        rb.AddTorque(Vector3.up * torque, ForceMode.Force);
    }
}
```

Скрипт логіки гри та підрахунку балів:

```
using UnityEngine;
using TMPro;

public class GameManager : MonoBehaviour
{
    // ---------- UI ------------------------------------------------------
--
    [Header("UI")]
    [SerializeField] private TextMeshProUGUI timeLabel;   // час гри
    [SerializeField] private TextMeshProUGUI scoreLabel;  // бали

    // ---------- Геймплей ------------------------------------------------
---
    private float _runTime;           // поточний таймер заїзду
    private int   _score;             // поточні бали
    private bool  _isRunning;

    // ---------- Прогрес -------------------------------------------------
---
```

```
    private ProgressData _progress;   // bestTime, bestScore та ін.

    private void Start() {
        // Завантажуємо рекорди (файл створиться, якщо його ще нема)
        _progress = SaveLoadSystem.LoadOrCreateDefault();

        // Скидаємо сесію
        NewSession();
    }

    private void NewSession() {
        _runTime = 0f;
        _score   = 0;
        _isRunning = true;
        UpdateUI();
    }

    private void Update() {
        if (!_isRunning) return;

        _runTime += Time.deltaTime;
        UpdateUI();
    }

    private void UpdateUI() {
        // іконка «clock»
        timeLabel.text  = $"<sprite name=\"clock\"> {_runTime:00}:{_runTime %
60:00}";
        // іконка «star»
        scoreLabel.text = $"<sprite name=\"star\"> {_score}";
    }

    // Викликати цей метод, коли човен коректно пришвартувався
    public void FinishRun() {
        _isRunning = false;

        // Оновлюємо рекорди, якщо покращили
        bool needSave = false;
        if (_progress.bestTime <= 0f || _runTime < _progress.bestTime) {
            _progress.bestTime = _runTime;
            needSave = true;
        }
        if (_score > _progress.bestScore) {
            _progress.bestScore = _score;
            needSave = true;
        }
        if (needSave) SaveLoadSystem.Save(_progress);
    }

    // додаємо бали під час проходження чекпоінта
    public void AddScore(int amount) {
        _score += amount;
        UpdateUI();
    }
}
```

Скрипт управління човном з мобільного пристрою:

```
using UnityEngine;
using UnityEngine.EventSystems;
```

```
public class MobileJoystick : MonoBehaviour, IDragHandler, IPointerUpHandler,
IPointerDownHandler {
    [SerializeField] private RectTransform knob;
    [SerializeField] private float maxRange = 100f;

    private Vector2 _input;
    public  Vector2 Input => _input;

    public void OnDrag(PointerEventData ev) {
        Vector2 pos = ev.position - (Vector2)transform.position;
        pos = Vector2.ClampMagnitude(pos, maxRange);
        knob.anchoredPosition = pos;
        _input = pos / maxRange;
    }
    public void OnPointerDown(PointerEventData ev) => OnDrag(ev);
    public void OnPointerUp(PointerEventData ev)  {
        knob.anchoredPosition = Vector2.zero;
        _input = Vector2.zero;
    }
}
```

4.2 Захист розробленого програмного забезпечення

У процесі розробки комп'ютерної гри на рушії Unity актуальним завданням стає захист програмного забезпечення від несанкціонованого копіювання та використання. Одним із найефективніших методів забезпечення такого захисту є впровадження системи ліцензування, яка базується на використанні *унікального ліцензійного ключа* з перевіркою через сервер. Цей підхід дозволяє здійснювати контроль за кількістю активацій, верифікувати користувача при запуску гри, а також реалізувати додаткові засоби захисту, зокрема прив'язку до конкретного пристрою.

У рамках цього підходу кожному користувачу гри після придбання надається унікальний ліцензійний ключ. Під час першого запуску гри на пристрої користувач вводить цей ключ у відповідне текстове поле. Далі клієнтська частина, реалізована в середовищі Unity, за допомогою інструменту UnityWebRequest відправляє HTTP-запит на сервер для перевірки ключа. Серверна частина, реалізована, наприклад, на платформі Node.js або Python (Flask), перевіряє, чи існує ключ у базі даних, чи ще не був використаний більше допустимої кількості разів (наприклад, не активований більш ніж тричі), та

повертає відповідь у форматі JSON. У разі позитивного результату перевірки, гра активується, а інформація про пристрій або IP-адресу користувача зберігається на сервері.

Фрагмент коду, що відповідає за надсилання запиту на перевірку ліцензії з боку клієнта, виглядає наступним чином:

```
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;

public class LicenseValidator : MonoBehaviour
{
    public string licenseKey;

    public void Validate()
    {
        StartCoroutine(CheckLicense(licenseKey));
    }

    private IEnumerator CheckLicense(string key)
    {
        string url = "https://example.com/api/validate?key=" + key;
        UnityWebRequest www = UnityWebRequest.Get(url);
        yield return www.SendWebRequest();

        if (www.result == UnityWebRequest.Result.Success)
        {
            if (www.downloadHandler.text == "VALID")
            {
                Debug.Log("Ліцензія підтверджена. Гру активовано.");
                // Дозволити доступ до гри
            }
            else
            {
                Debug.Log("Недійсний ліцензійний ключ.");
                // Блокувати доступ або обмежити функціонал
            }
        }
        else
```

```
            {
                Debug.Log("Помилка підключення до серверу: " + www.error);
            }
        }
    }
```

Відповідна серверна частина може бути реалізована з використанням бази даних, у якій зберігаються дозволені ключі та кількість їх активацій. Приклад реалізації ендпоінту на Python з використанням Flask:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# Імітація бази даних ключів
valid_keys = {
    "AAAA-BBBB-CCCC-DDDD": {"uses": 0, "max_uses": 3}
}

@app.route('/api/validate')
def validate():
    key = request.args.get('key')
    if key in valid_keys:
        data = valid_keys[key]
        if data["uses"] < data["max_uses"]:
            data["uses"] += 1
            return "VALID"
        else:
            return "USED"
    return "INVALID"

if __name__ == '__main__':
    app.run()
```

Для юридичного захисту програмного продукту, розробленого на Unity, доцільно застосувати модель ліцензування на основі типової **End-User License Agreement (EULA)** – тобто Ліцензійної угоди з кінцевим користувачем. Така

ліцензія описує правовий статус користувача гри, заборону на її несанкціоноване копіювання, декомпіляцію чи поширення, а також містить положення про умови використання придбаного ключа.

Таким чином, запропонована система захисту програмного забезпечення поєднує як технічні, так і юридичні механізми, зокрема верифікацію ключа через серверну частину та укладену з користувачем ліцензійну угоду. Це забезпечує не лише обмеження доступу до неліцензійного використання, але й підвищує контроль над поширенням продукту, сприяє підтримці авторських прав і дозволяє впроваджувати гнучкі механізми монетизації гри.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Щоб запустити «Sea Adventures» на персональному комп’ютері, достатньо завантажити архів з офіційної сторінки проєкту, розпакувати його у будь-яку теку й запустити виконуваний файл. На Windows це SeaAdventures.exe, на macOS – пакет .app, а на Linux – файл SeaAdventures.x86_64, який перед запуском необхідно позначити як виконуваний за допомогою команди chmod +x. Під час першого старту рушій Unity автоматично створить у теці «Документи» підкорінний каталог SeaAdventures, де зберігатимуться усі налаштування та файли прогресу. Додаткове встановлення драйверів чи бібліотек не потрібне; достатньо, щоб у системі був доступний рендерер DirectX 11, Vulkan або Metal.

На мобільному телефоні процедура відрізняється залежно від платформи. Для Android потрібно скопіювати .apk-файл на пристрій, дозволити інсталяцію з невідомих джерел у налаштуваннях безпеки та відкрити пакет через будь-який файловий менеджер, після чого система сама інсталює програму й додасть її іконку на робочий стіл. На iOS поширення здійснюється через TestFlight: користувач отримує запрошувальний код, вводить його в додатку TestFlight, натискає «Install» і вже за кілька секунд відкриває гру, як звичайну App Store-програму. Усі внутрішні дані зберігаються у захищеній sandbox-теці, тому при оновленні збірки рекорди не губляться.

Після запуску на екрані з’являється головне меню: великими кнопками пропонуються «Start», «Settings» і «High Scores» (рис. 5.1).

Натиснувши «Start», гравець одразу потрапляє на останній відкритий рівень. У лівому верхньому куті бачить іконку годинника з таймером, а під нею зірочку зі сумарними очками.

Керування на комп’ютері здійснюється клавішами W, A, S, D або лівим стиком ґеймпада: клавіша W чи рух стика вперед дає хід уперед, S – реверс, A і D відповідають за повороти ліворуч і праворуч. На телефоні роль цих клавіш

виконує екранний джойстик у лівій частині дисплея: переміщення великого пальця задає вектор тяги та кермового кута.

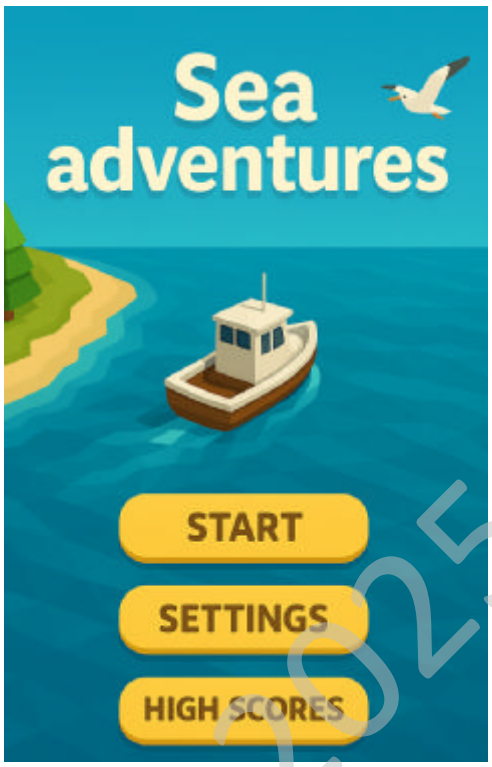

Рисунок 5.1 – Меню користувача

Головна мета гри – без ушкоджень довести човен до наміченої док-зони, позначеної контуром жовтого кольору. Удар об пірс чи іншу перешкоду супроводжується характерним звуком і зменшує запас міцності корпусу; ступінь пошкоджень відображається у правому верхньому куті у вигляді смуги. Після успішного швартування або критичного руйнування корпусу гра показує екран результатів з підсумковим часом, кількістю зароблених балів і кнопками «Retry» та «Next Level». Якщо досягнуто особистого рекорду, система негайно фіксує його у файлі прогресу, тож наступного разу статистика буде завантажена автоматично.

Головне вікно гри зображено на рис. 5.2

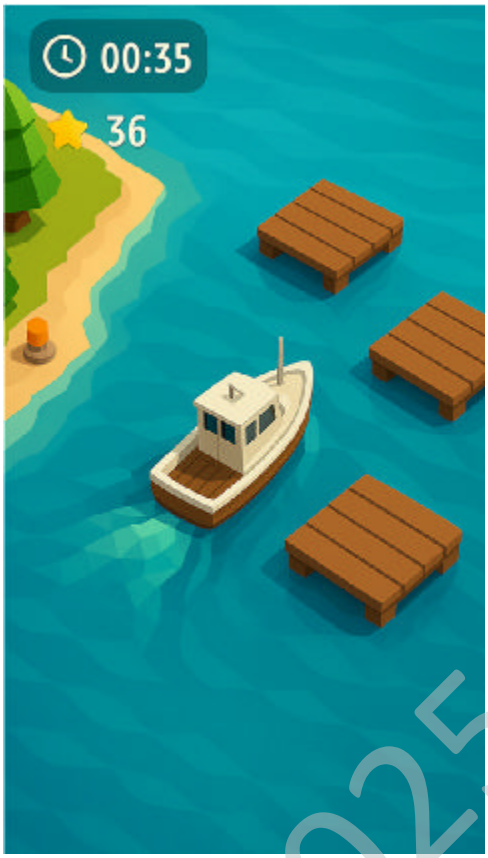

Рисунок 5.2 – Основне вікно розробленої гри

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для реалізації комп'ютерної гри у жанрі симулятора морського човна на основі ігрового рушія Unity.

Для вирішення поставленої мети було проведено:

– Огляд існуючих систем для реалізації комп'ютерних ігор у жанрі симуляторів на основі ігрового рушія Unity.

– Проектування системи для реалізації комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

– Програмна реалізація системи для реалізації комп'ютерної гри у жанрі симулятора морського човна на основі ігрового рушія Unity.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішувати завдання д симуляції морського човна на основі ігрового рушія Unity.

Розроблене програмне забезпечення представляє собою додаток, що можна використовувати практично на будь-якому сучасному комп'ютері або мобільному пристрої.

Програмне забезпечення розроблялося у об'єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки програмного забезпечення системи моделювання роботи комп'ютерних мереж використовувалися мова програмування C# та середовище розробки Unity. Дані мова програмування і середовище розробки дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

У п'ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення програмного забезпечення та інструкція для його використання.

Для захисту розробленого програмного забезпечення було обрано вільну

ліцензію типової End-User License Agreement (EULA) перевірку унікального ліцензійного ключа при встановленні гри.

Програмне забезпечення для реалізації комп’ютерної гри у жанрі симулятора морського човна на основі ігрового рушія Unity є одночасно цікавою грою та навчальною симуляцією та може використовуватися широким колом користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity 2022 : [книга]. – Shelter Island, NY : Manning Publications, 2022. – 464 с. – Режим доступу: https://www.manning.com/books/unity-in-action-third-edition

2. Borromeo N. Hands-On Unity Game Development. Fourth Edition. Unlock the Power of Unity 2023 : [книга]. – Birmingham : Packt Publishing, 2023. – 744 с. – Режим доступу: https://www.packtpub.com/product/hands-on-unity-game-development/9781835085714

3. Ferrone H. Learning C# by Developing Games with Unity 2022 : [посібник]. – 6-те вид. – Birmingham : Packt Publishing, 2022. – 798 с. – Режим доступу: https://www.packtpub.com/product/learning-c-by-developing-games-with-unity-2022/9781803242241

4. Thorn A. Unity 2023 By Example : [книга]. – Birmingham : Packt Publishing, 2023. – 824 с. – Режим доступу: https://www.packtpub.com/product/unity-2023-by-example/9781837637652

5. Carver N. Practical Shader Development for Unity : [монографія]. – Berkeley : Apress, 2020. – 465 с. – Режим доступу: https://link.springer.com/book/10.1007/978-1-4842-5195-4

6. Palmer G. Physics for Game Programmers. 3-rd ed. : [книга]. – Berkeley : Apress, 2023. – 514 с. – Режим доступу: https://link.springer.com/book/10.1007/978-1-4842-9092-2

7. Yannakakis G. N., Togelius J. Artificial Intelligence and Games : [монографія]. – Cham : Springer, 2018. – 337 с. – Режим доступу: https://link.springer.com/book/10.1007/978-3-319-63519-4

8. Albahari J., Albahari B. C# 12 in a Nutshell. The Definitive Reference : [книга]. – Sebastopol : O'Reilly Media, 2023. – 1100 с. – Режим доступу: https://dl.ebooksworld.ir/books/CSharp.12.in.a.Nutshell.The.Definitive.Reference.9781098147440.pdf

9. Troelsen A., Japikse P. Pro C# 10 with .NET 6. Foundational Principles and Practices in Programming : [книга]. – Berkeley : Apress, 2022. – 1512 с. – Режим доступу: https://dl.ebooksworld.ir/books/Pro.CSharp.10.with.NET.6.Andrew.Troelsen.Phil.Japikse.Apress.9781484278680.EBooksWorld.ir.pdf

10. Nystrom R. Game Programming Patterns : [книга]. – Genever Benning, 2014. – 354 с. – Режим доступу: https://gameprogrammingpatterns.com

11. Knuth D. E. The Art of Computer Programming. Vol. 1 : Fundamental Algorithms. 4-th ed. – Boston : Addison-Wesley, 2023. – 780 с.

12. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4-th ed. – Cambridge : MIT Press, 2022. – 1312 с.

13. Креневич А. М. Алгоритми і структури даних : підручник. – Київ : ВПЦ «Київський університет», 2021. – 200 с. – Режим доступу: https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf

14. Онищенко В. В., Коник Р. С. Алгоритми та структури даних C++ : навч. посібник. – Львів : ЛНУ ім. І. Франка, 2019. – 280 с. – Режим доступу: https://duikt.edu.ua/ua/lib/1/category/2153

15. Осадчий М. І. Основи алгоритмізації та програмування : навчальний посібник. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 312 с. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/48282/1/Osnovy_alhorytmizatsii.pdf

16. Tessendorf J. Simulating Ocean Water. – SIGGRAPH Course Notes, 2004. – 20 с. – Режим доступу: https://tessendorf.org/research/publications.php

17. Ranković M. Dynamic Water Physics for Games. – Game Developer Conference Europe, 2019. – URL: https://www.gdcvault.com

18. Unity Technologies. Unity Manual : Scriptable Render Pipeline. – 2023. – URL: https://docs.unity3d.com/Manual/ScriptableRenderPipeline.html

19. Unity Technologies. Unity Scripting API : CustomRenderTexture. – 2025. – URL: https://docs.unity3d.com/ScriptReference/CustomRenderTexture.html

20. Lost Polygon. Dynamic Water System : Asset Store Documentation. – 2016. – URL: https://discussions.unity.com/t/dynamic-water-system-released/512489

21. Gregory J. Game Engine Architecture. 4-th ed. – Boca Raton : CRC Press, 2021. – 1332 с.

22. Millington I., Funge J. Artificial Intelligence for Games. 3-rd ed. – Boca Raton : CRC Press, 2022. – 1037 с.

23. Goldberg D. E. Genetic Algorithms in Search, Optimization and Machine Learning. – Boston : Addison-Wesley, 1989. – 432 с.

24. Batchelor B. Game Audio Programming 2. Principles and Practices. – Boca Raton : CRC Press, 2020. – 408 с.

25. Souza D. B. Unity 2D Game Development : [книга]. – Birmingham : Packt Publishing, 2021. – 352 с.

26. Goldstone W. Unity 2021 Game Development Essentials. – 6-те вид. – Birmingham : Packt Publishing, 2021. – 660 с.

27. Dickmann S. M., Grosser M. Game Programming Patterns in Unity 2023 : від теорії до практики. – Берлін : Springer Vieweg, 2023. – 486 с. – Режим доступу: https://link.springer.com/book/10.1007/978-3-658-39490-7

28. Valente M. Advanced Shaders for Unity 2022. – Birmingham : Packt Publishing, 2022. – 330 с. – Режим доступу: https://www.packtpub.com/product/advanced-shaders-for-unity-2022/9781801077760

29. Hecker C. Finite-Difference Simulations of Shallow-Water Waves. – ACM SIGGRAPH Course Notes, 2019. – 15 с. – URL: https://chrishecker.com

30. Geigel J. Water Surface Modeling for Real-Time Graphics. – Journal of Graphics Tools. – 2021. – Т. 25, № 3. – С. 1–18. – URL: https://www.jgt-online.org

31. Матвіїшин Є. П. Технології комп’ютерної графіки : навчальний посібник. – Львів : Видавництво ЛНУ ім. І. Франка, 2020. – 276 с. – Режим доступу: https://repository.lnu.edu.ua/handle/123456789/38060

32. Мокрий С. В. Unity та C# у розробці інтерактивних симуляторів. – Вісник НТУУ «КПІ». Серія «Інформатика». – 2022. – № 74. – С. 54–62. – URL:

http://journals.kpi.ua

33. Кириченко В. Г. Реалістична анімація рідин у відеоіграх. – Сучасні інформаційні системи. – 2023. – Т. 7, № 1. – С. 12–20. – URL: https://ais.khpi.edu.ua

34. Shewell P. Unity UI Cookbook. – 3-тє вид. – Birmingham : Packt Publishing, 2023. – 488 с. – Режим доступу: https://www.packtpub.com/product/unity-ui-cookbook-third-edition/9781803239074

35. Qu X., Stutterheim K. Unity Mobile Game Development. – Shelter Island : Manning Publications, 2021. – 504 с. – URL: https://www.manning.com/books/unity-mobile-game-development

36. Soukup P. Mastering C# Source Generators. – Нью-Йорк : Apress, 2024. – 318 с. – URL: https://link.springer.com/book/10.1007/978-1-4842-9485-2

37. Burns M. Effective C#: 50 Specific Ways to Improve Your Programs. 3-rd ed. – Boston : Addison-Wesley, 2022. – 368 с.

38. Sedgewick R., Wayne K. Algorithms, 4-th ed. – Boston : Addison-Wesley, 2020. – 955 с.

39. Langr J. Modern C++ and C# Design Patterns. – Берлін : Springer, 2021. – 412 с. – URL: https://link.springer.com/book/10.1007/978-3-030-80563-4

40. Eberly D. Game Physics: Second Edition. – Boca Raton : CRC Press, 2021. – 960 с.

41. Shiffman D. Nature of Code. – 2-ге вид. – Нью-Йорк : D. Shiffman, 2021. – 450 с. – URL: https://natureofcode.com/book

42. Unity Technologies. Unity Manual 2025 LTS : Physics. – 2025. – URL: https://docs.unity3d.com/Manual/PhysicsSection.html

43. Unity Technologies. Input System Package 1.8 Documentation. – 2025. – URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8

44. Khronos Group. Vulkan 1.3 Specification. – 2023. – URL: https://registry.khronos.org/vulkan

45. McGuire M., Luebke D. Computer Graphics Archive: Water Rendering

Survey. – 2022. – URL: https://graphics-archive.com/papers/water-survey.pdf

46. Пономаренко Р. О. Програмування комп’ютерних ігор у рушії Unity : курс лекцій. – Кропивницький : ЦНТУ, 2024. – 192 с.

47. Костик А. І. Паралельні обчислення у графічних застосунках. – Харків : ХНУРЕ, 2021. – 260 с.

48. Пасіхов В. В. Основи фізики реалістичних візуалізацій. – Київ : КНЕУ, 2022. – 234 с.

49. Humble J., Farley D. Continuous Delivery: Reliable Software Releases. – Бостон : Addison-Wesley, 2022. – 512 с. – URL: https://continuousdelivery.com

50. Caballero H. Thinking Unity-wise: Architectural Patterns for Games. – GDC Vault, 2023. – URL: https://gdcvault.com

51. Valente M. Practical Game Design with Unity and Blender. – 2-ге вид. – Birmingham : Packt Publishing, 2021. – 462 с.

52. Gillian S. Game Audio Implementation with FMOD and Unity. – Нью-Йорк : Routledge, 2022. – 276 с.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №48-02 від 17.01.2025 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує комп'ютерну гру у жанрі симулятора на основі ігрового рушія Unity.

5.2 Показники призначення

Система повинна забезпечувати:

– систему для моделювання роботи комп’ютерних мереж;
– цілісність даних у процесі роботи та при зберіганні;
– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel Core i7 (12-го покоління) / 16 ГБ RAM / SSD 512 ГБ + HDD 1 ТБ / NVIDIA GeForce RTX 3060 8 GB або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мовах програмування C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 24.05.2025 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист __.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Є.В. Мелешко

Програмне забезпечення комп'ютерної гри у жанрі симулятора на основі ігрового рушія Unity

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 17

Літера: РП

Кропивницький – 2025 року

Файл WaterSimulation.cs основної програми

```
using UnityEngine;                  // Підключаємо базовий простір назв Unity

/// <summary>
/// Симуляція хвильової текстури за допомогою CustomRenderTexture.
/// Клік ЛКМ — позитивний сплеск, ПКМ — негативний.
/// </summary>
public class WaterSimulation : MonoBehaviour
{
    [SerializeField]                // Посилання на текстуру, що рендериться
шейдером
    private CustomRenderTexture waterTexture;

    [SerializeField, Range(1, 20)]   // Скільки ітерацій симулятора виконувати
за кадр
    private int iterationsPerFrame = 5;

    private ProgressData _progress;          // рекорди гравця (завантажимо у
Start)

    private void Start()
    {

     _progress = SaveLoadSystem.LoadOrCreateDefault();  // завантаження прогресу
     waterTexture.Initialize();   // ініціалізуємо текстуру перед першим
оновленням
    }

    private void Update()
    {
        waterTexture.ClearUpdateZones(); // очищаємо попередні зони оновлення
        UpdateInteractionZones();        // додаємо зони за користувацьким ввідо
        waterTexture.Update(iterationsPerFrame); // запускаємо обчислення
шейдера
    }

    /// <summary>
    /// Створює дві зони оновлення:
    /// 1) уся текстура (для звичайної еволюції);
    /// 2) крихітна точка під курсором – для сплеску/ямки.
    /// </summary>
    private void UpdateInteractionZones()
    {
        bool leftClick  = Input.GetMouseButton(0);
        bool rightClick = Input.GetMouseButton(1);
        if (!leftClick && !rightClick) return;

        if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition),
out RaycastHit hit))
        {
            // Зона 0: уся поверхня хвиль
            var fullZone = new CustomRenderTextureUpdateZone
            {
                needSwap          = true,
                passIndex         = 0,   // основний пас шейдера
                rotation          = 0f,
                updateZoneCenter  = new Vector2(0.5f, 0.5f),
                updateZoneSize    = Vector2.one
            };

            // Зона 1: точка під курсором (пас 1 – підняти, пас 2 – опустити)
            var clickZone = new CustomRenderTextureUpdateZone
            {
```

```
                needSwap          = true,
                passIndex         = leftClick ? 1 : 2,
                rotation          = 0f,
                updateZoneCenter  = new Vector2(hit.textureCoord.x, 1f -
hit.textureCoord.y),
                updateZoneSize    = new Vector2(0.01f, 0.01f)
            };

            waterTexture.SetUpdateZones(new[] { fullZone, clickZone });
        }
    }
}
```

Файл BoatController.cs – керування човном (WASD або ґеймпад)

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class BoatController : MonoBehaviour
{
    [Header("Параметри руху")]
    public float thrustForce = 450f;      // Сила лінійного прискорення
    public float turnTorque  = 90f;       // Крутний момент для обертання

    private Rigidbody rb;
    private Vector2   inputVector;

    private void Awake() => rb = GetComponent<Rigidbody>();

    private void Update()
    {
        // Варіант управління для ПК
        //inputVector.x = Input.GetAxis("Horizontal"); // A/D або ←/→
       // inputVector.y = Input.GetAxis("Vertical");   // W/S або ↑/↓

          // Варіант управління для мобільного телефону
        float steer    = InputAdapter.Instance.GetSteer();
        float throttle = InputAdapter.Instance.GetThrottle();
    }

    private void FixedUpdate()
    {
        // Рух уперед/назад
        Vector3 force = transform.forward * inputVector.y * thrustForce;
        rb.AddForce(force, ForceMode.Force);

        // Поворот навколо осі Y
        float torque = inputVector.x * turnTorque;
        rb.AddTorque(Vector3.up * torque, ForceMode.Force);
    }
}
```

Файл DockTrigger.cs – тригер "фінішної" зони швартування

```
using UnityEngine;

[RequireComponent(typeof(Collider))]
public class DockTrigger : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))    // човен повинен мати тег "Player"
        {
            FindObjectOfType<GameManager>()?.OnBoatDocked();
        }

    SaveLoadSystem.UpdateProgress(currentLevelId, currentTime, currentDamage);
    }
}
```

Файл GameManager.cs – логіка сесії (таймер, штрафи, UI)

```
using UnityEngine;
using TMPro;

public class GameManager : MonoBehaviour
{
    // ---------- UI ------------------------------------------------------------
    [Header("UI")]
    [SerializeField] private TextMeshProUGUI timeLabel;   // час гри
    [SerializeField] private TextMeshProUGUI scoreLabel;  // бали

    // ---------- Геймплей ------------------------------------------------------
-
    private float _runTime;           // поточний таймер заїзду
    private int   _score;             // поточні бали
    private bool  _isRunning;

    // ---------- Прогрес -------------------------------------------------------
-
    private ProgressData _progress;   // bestTime, bestScore та ін.

    private void Start() {
        // Завантажуємо рекорди (файл створиться, якщо його ще нема)
        _progress = SaveLoadSystem.LoadOrCreateDefault();

        // Скидаємо сесію
        NewSession();
    }

    private void NewSession() {
        _runTime = 0f;
        _score   = 0;
        _isRunning = true;
        UpdateUI();
    }

    private void Update() {
        if (!_isRunning) return;

        _runTime += Time.deltaTime;
        UpdateUI();
    }

    private void UpdateUI() {
        // іконка «clock»
        timeLabel.text  = $"<sprite name=\"clock\"> {_runTime:00}:{_runTime %
60:00}";
        // іконка «star»
        scoreLabel.text = $"<sprite name=\"star\"> {_score}";
    }

    // Викликати цей метод, коли човен коректно пришвартувався
    public void FinishRun() {
        _isRunning = false;

        // Оновлюємо рекорди, якщо покращили
        bool needSave = false;
        if (_progress.bestTime <= 0f || _runTime < _progress.bestTime) {
            _progress.bestTime = _runTime;
            needSave = true;
        }
        if (_score > _progress.bestScore) {
            _progress.bestScore = _score;
            needSave = true;
        }
        if (needSave) SaveLoadSystem.Save(_progress);
    }
```

```
    // додаємо бали під час проходження чекпоінта
    public void AddScore(int amount) {
        _score += amount;
        UpdateUI();
    }
}
```

Файл CameraFollow.cs – проста Cinemachine-free камера

```
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    [SerializeField] private Transform target;   // човен
    [SerializeField] private Vector3   offset = new Vector3(0, 10, -12);
    [SerializeField] private float     smoothSpeed = 5f;

    private void LateUpdate()
    {
        if (!target) return;
        Vector3 desiredPosition = target.position + offset;
        transform.position = Vector3.Lerp(transform.position, desiredPosition,
smoothSpeed * Time.deltaTime);
        transform.LookAt(target);
    }
}
```

Файл SoundManager.cs – синглтон програвача Audiosource-ів

```
using UnityEngine;

public class SoundManager : MonoBehaviour {
    public static SoundManager Instance { get; private set; }

    [Header("Джерела аудіо")]
    [SerializeField] private AudioSource двигун;
    [SerializeField] private AudioSource зіткнення;

    private void Awake() {
        if (Instance != null) { Destroy(gameObject); return; }
        Instance = this; DontDestroyOnLoad(gameObject);
    }

    public void PlayImpact(float сила) {
        зіткнення.pitch = Mathf.Lerp(0.8f, 1.2f, сила);
        зіткнення.volume = Mathf.Clamp01(сила);
        зіткнення.Play();
    }

    public void SetEnginePitch(float throttle) {
        if (двигун == null) return;
        двигун.pitch = Mathf.Lerp(0.8f, 1.4f, throttle);
        if (!двигун.isPlaying) двигун.Play();
    }
}
```

Файл ImpactSound.cs – звуки при зіткненнях + DamageSystem

```
using UnityEngine;

[RequireComponent(typeof(Collider))]
public class ImpactSound : MonoBehaviour {
    [SerializeField] private float мінімальнаШвидкість = 1f;

    private void OnCollisionEnter(Collision other) {
        float сила = other.relativeVelocity.magnitude;
        if (сила < мінімальнаШвидкість) return;
        float норм = Mathf.InverseLerp(мінімальнаШвидкість, 10f, сила);
        SoundManager.Instance?.PlayImpact(норм);
        DamageSystem.Instance?.ApplyDamage(norm * 5f);
    }
}
```

Файл DamageSystem.cs – підрахунок пошкоджень корпусу

```
using UnityEngine;

public class DamageSystem : MonoBehaviour {
    public static DamageSystem Instance { get; private set; }
    [SerializeField] private float maxHP = 100f;
    public float IHP { get; private set; }

    private void Awake() {
        if (Instance != null) { Destroy(gameObject); return; }
        Instance = this;
        IHP = maxHP;
    }

    public void ApplyDamage(float dmg) {
        IHP = Mathf.Max(0, IHP - dmg);
        GameManager.Instance?.UpdateHealth(IHP / maxHP);
    }
}
```

Файл LevelManager.cs – дані сцени та статичний доступ

```
using UnityEngine;

public class LevelManager : MonoBehaviour {
    public static LevelManager Current { get; private set; }
    [Tooltip("Вектор напрямку хвиль у рівні (нормалізований)")]
    public Vector2 напрямХвиль = Vector2.right;
    private void Awake() { Current = this; }
}
```

Файл InputAdapter.cs – шар між Legacy Input та Input System 1.8

```
using UnityEngine;

/// <summary>
/// Один-єдиний адаптер вводу для всіх платформ.
/// • На ПК / консолі читає стандартні осі «Horizontal» / «Vertical».
/// • На Android / iOS бере дані з MobileJoystick (тач-керування).
/// </summary>
public class InputAdapter : MonoBehaviour
{
    // ---------- Singleton ----------------------------------------------------
    public static InputAdapter Instance { get; private set; }

    private void Awake() {
        if (Instance != null && Instance != this) {
            Destroy(gameObject);
            return;
        }
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }

    // ---------- Налаштування -------------------------------------------------
-
#if UNITY_ANDROID || UNITY_IOS
    [SerializeField] private MobileJoystick mobileJoystick;     // посилання з
Інспектора
    private static readonly bool IsMobile = true;
#else
    private static readonly bool IsMobile = false;
#endif

    // ---------- Публічні фасади ----------------------------------------------
    /// <summary>Кермо: -1 … 1 (ліворуч → праворуч).</summary>
    public float GetSteer() {
        return IsMobile ? (mobileJoystick ? mobileJoystick.Input.x : 0f)
                        : Input.GetAxisRaw("Horizontal");       // A/D або
стрілки
    }

    /// <summary>Газ/реверс: -1 … 1 (назад → вперед).</summary>
    public float GetThrottle() {
        return IsMobile ? (mobileJoystick ? mobileJoystick.Input.y : 0f)
                        : Input.GetAxisRaw("Vertical");         // W/S або
стрілки
    }
}
```

Файл SettingsMenu.cs – зчитування/запис PlayerPrefs та Canvas UI

```
using UnityEngine;

public class SettingsMenu : MonoBehaviour {
    public void SetQuality(int q) { QualitySettings.SetQualityLevel(q); }
    public void SetVolume(float v) { AudioListener.volume = v; }
    private void Start() {
        // Завантажити налаштування
    }
}
```

Файл SaveLoadSystem.cs – система серіалізації й десеріалізації даних прогресу гравця

```
// Зберігає найкращий час, мін. ушкодження та перелік відкритих рівнів.
// Використовує JSON + PlayerPrefs

using System;
using System.Collections.Generic;
using UnityEngine;

[Serializable]
public class ProgressData {
    public float bestTime = float.MaxValue;          // найкращий час
проходження рівня (секунди)
    public float minDamage = float.MaxValue;         // найменші пошкодження
корпусу (%)
    public List<string> unlockedLevels = new();      // ідентифікатори відкритих
рівнів
}

public static class SaveLoadSystem {
    private const string ProgressKey = "HarborPilot_Progress_v1"; // ключ у
PlayerPrefs

    /// <summary>
    /// Повертає поточні дані прогресу або створює дефолтні.
    /// </summary>
    public static ProgressData Load() {
        if (!PlayerPrefs.HasKey(ProgressKey))
            return new ProgressData();

        string json = PlayerPrefs.GetString(ProgressKey);
        try {
            return JsonUtility.FromJson<ProgressData>(json) ?? new
ProgressData();
        } catch (Exception e) {
            Debug.LogWarning($"Пошкоджені дані прогресу: {e.Message}; створюю
нові.");
            return new ProgressData();
        }
    }

    /// <summary>
    /// Зберігає структуру ProgressData у PlayerPrefs.
    /// </summary>
    public static void Save(ProgressData data) {
        if (data == null) return;
        string json = JsonUtility.ToJson(data);
        PlayerPrefs.SetString(ProgressKey, json);
        PlayerPrefs.Save();
    }

    /// <summary>
    /// Оновлює рекорди після завершення рівня.
    /// </summary>
    /// <param name="levelId">Рядковий ідентифікатор сцени/рівня.</param>
    /// <param name="time">Час проходження в секундах.</param>
    /// <param name="damage">Відсоток пошкоджень (0-100).</param>
    public static void UpdateProgress(string levelId, float time, float damage)
{
        ProgressData data = Load();

        // Додаємо рівень у список, якщо ще не відкритий
        if (!data.unlockedLevels.Contains(levelId))
            data.unlockedLevels.Add(levelId);

        // Кращий час – менше значення
        if (time < data.bestTime)
```

```
            data.bestTime = time;

        // Менші пошкодження – краще значення
        if (damage < data.minDamage)
            data.minDamage = damage;

        Save(data);
    }

    /// <summary>
    /// Скидає усі дані прогресу (кнопка «Скинути» у налаштуваннях).
    /// </summary>
    public static void ResetProgress() {
        PlayerPrefs.DeleteKey(ProgressKey);
        PlayerPrefs.Save();
    }
}
```

Файл MobileJoystick.cs – система управління човном для мобільного пристрою

```
using UnityEngine;
using UnityEngine.EventSystems;

public class MobileJoystick : MonoBehaviour, IDragHandler, IPointerUpHandler,
IPointerDownHandler {
    [SerializeField] private RectTransform knob;
    [SerializeField] private float maxRange = 100f;

    private Vector2 _input;
    public  Vector2 Input => _input;

    public void OnDrag(PointerEventData ev) {
        Vector2 pos = ev.position - (Vector2)transform.position;
        pos = Vector2.ClampMagnitude(pos, maxRange);
        knob.anchoredPosition = pos;
        _input = pos / maxRange;
    }
    public void OnPointerDown(PointerEventData ev) => OnDrag(ev);
    public void OnPointerUp(PointerEventData ev)  {
        knob.anchoredPosition = Vector2.zero;
        _input = Vector2.zero;
    }
}
```