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
“____” _____ 2021 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY”

Виконав здобувач вищої освіти
ІІ курсу, групи _____
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____Сніховський А.О.
«____» _____ 2021 р.

Керівник проекту
доктор технічних наук, доцент
_____ Єлизавета МЕЛЕШКО
«____» _____ 2021 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *122 "Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерні науки"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« __» _____ 2021 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Сніховському Андрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY*

2. Керівник роботи *Мелешко Єлизавета Владиславівна, доктор техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №39-13 від 02.08.2021 року

3. Строк подання роботи до захисту *10.12.2021 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне забезпечення системи симуляції рідкого середовища на 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. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2021	12.11.2021
Охорона праці	Оришака О.В., к.т.н., доцент	04.11.2021	20.11.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2021 р.	
3.	Розробка моделі компонента	20.10.2021 р.	
4.	Розробка структур даних	25.10.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2021 р.	
6.	Програмування алгоритмів	10.11.2021 р.	
7.	Розрахунок економічної ефективності	13.11.2021 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2021 р.	
9.	Оформлення ПЗ	17.11.2021 р.	
10.	Попередній захист роботи	28.11.2021 р.	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Сніховський А.О. Дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній випускній кваліфікаційній роботі розроблено програмне забезпечення, яке призначено для системи симуляції рідкого середовища на UNITY.

Метою розробки є дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY.

Об'єктом дослідження є процес симуляції рідкого середовища у комп'ютерних іграх.

Предметом дослідження є методи та алгоритми симуляції рідкого середовища на UNITY.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах комп'ютерної графіки, а також теорії ймовірності.

Результат роботи – програмна реалізація системи симуляції рідкого середовища на UNITY.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

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

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 7/8/10.

Програму розроблено на мові програмування C# в середовищах розробки Microsoft Visual Studio та Unity.

Ключові слова: комп'ютерна інженерія, симуляція, рідке середовище

ABSTRACT

Snikhovskyi A.O. Research and software realization of the system of the liquid environment simulation system at UNITY. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi 2021

In this master's thesis, software designed for the system of the liquid environment simulation system at UNITY.

The purpose of the development is the research and program implementation of the system of the liquid environment simulation system at UNITY.

The object of the study is the process of simulating a liquid medium in computer games.

The subject of the study is methods and algorithms for simulating a liquid medium at UNITY.

The research methods are based on the theory of object-oriented programming, the theory of algorithms, computer graphics methods, and probability theory. The result of the work is the software implementation of the system of the liquid environment simulation system at UNITY.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on an IBM PC running Windows 7/8/10 PC.

The program was developed in the C# programming language in Microsoft Visual Studio and Unity development environments.

Keywords: computer engineering, simulation, liquid medium

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 3
ВСТУП 4
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 6
1.1 Призначення системи 6
1.2 Область застосування 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 8
2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи 8
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування 13
2.3 Розгорнута постановка завдання 16
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 18
3.1 Опис функціонування системи 18
3.2 Розробка структурної схеми 25
3.3 Розробка функціональної схеми 26
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ 30
4.1 Блок-схеми та опис алгоритмів функціонування системи 30
4.2 Захист розробленого програмного забезпечення 35
5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ 39
6 НАУКОВА НОВИЗНА 43
7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ 44
7.1 Техніко-економічне обґрунтування теми магістерської роботи 44
7.2 Розрахунок трудомісткості розробки програмної продукції 46

7.3 Визначення чисельності виконавців і планового фонду зарплати 48
7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника 52
7.5 Визначення собівартості розробки та ціни програмної продукції 56
7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції 60
7.7 Визначення експлуатаційних витрат 60
7.8 Визначення економічної ефективності програмної продукції 62
7.9 Висновки 64
8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ 65
8.1 Вступ 65
8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером 66
8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста 67
8.4 Розробка заходів з умов поліпшення охорони праці 70
8.5 Розрахункова частина 71
8.6 Висновки до розділу 73
9 ОСНОВНІ ВИСНОВКИ 74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 76

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АЗ	–	апаратний засіб;
MGA	–	архітектура множинних поколінь записів;
ОС	–	операційна система;
Unity	–	ігровий движок;
ПЗ	–	постановка задач;
ТЗ	–	технічна задача.

ВСТУП

Актуальність теми. Популярність відносно простих ігрових движків зростає кожен рік. На них розробляють не тільки ігри, а й додатки для візуалізації інтер'єрів, природних локацій, візуалізації фізичних явищ тощо. Існує багато движків для дослідження та вивчення, але лише один досить добре працює з симуляцією фізичних явищ якнайкраще – це ігровий движок Unity, який працює з мовою програмування C#. Метою даної роботи було спроектувати та реалізувати просте і невибагливе до характеристик комп'ютера програмне забезпечення для симуляцій рідин, з простим та зрозумілим інтерфейсам.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY.

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

– Огляд існуючих систем симуляції рідкого середовища на UNITY.

– Дослідження системи симуляції рідкого середовища на UNITY.

– Програмна реалізація системи симуляції рідкого середовища на UNITY.

Об'єктом дослідження є процес симуляції рідкого середовища у комп'ютерних іграх.

Предметом дослідження є методи та алгоритми симуляції рідкого середовища на UNITY.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах комп'ютерної графіки, а також теорії ймовірності.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Удосконалено метод симуляції рідкого середовища для комп'ютерних ігор у середовищі ігрового рушія UNITY.

2. Розроблено вітчизняний продукт симуляції рідкого середовища для комп’ютерних ігор у середовищі ігрового рушія UNITY, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в т ому, що розроблені алгоритми дозволяють успішно вирішувати задачі симуляції рідкого середовища на UNITY.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY, є актуальною задачею, яка потребує вирішення у даній кваліфікаціній роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розроблена система призначена для використання на персональних комп'ютерах різної потужності для моделювання та візуалізації такого фізичного явища як рідкі середовища, а саме різних водойм.

Також існує великий потенціал використання симуляцій рідинних середовищ в комп'ютерних іграх. В іграх це може використовуватись для придання більшої фотореалістичності зображенням, або надання рідинам найбільш реалістичної поведінки при інтерактивній взаємодії гравця з ними.

Велика ймовірність використовування розробленого програмного забезпечення у дизайні, для задання реалістичного зображення рідині та проектування високоякістних рендерів з використанням явищ рідини.

В навчанні дане програмне забезпечення може використовуватись як віртуальний посібник по алгоритмам комп'ютерної графіки та моделюванні фізичних явищ під час вивчення властивостей та поведінки рідин та взаємодії з ними. Тож, дане програмне забезпечення підходить для дистанційного навчання як посібник та практична база для вивчення взаємодії рідин з середовищем та об'єктами.

1.2 Область застосування

Розроблене в ході роботи програмне забезпечення можна використовувати при налаштуванні програмного середовища будь яких ПК під управлінням ОС Windows. Така система може бути корисною в б агатьох сферах використання сучасних комп'ютерів.

Особливо доцільним, буде використання запропонованого програмного забезпечення та положень описаних в пояснювальній записці для навчальних закладів середнього та вищого рівнів. Воно надасть можливість користувачам і студентам комп'ютерних спеціальностей наочно зрозуміти основні принципи симуляції водного середовища для своїх досліджень.

Приведені дослідження та аналіз літературних джерел прискорить засвоєння та закріплення навчального матеріалу з вк азаної тематики. Крім того надасть можливість розроблювати більш оптимальні алгоритми при розробці власних програмних додатків для симуляції рідких середовищ.

Використані в ході роботи над проектом алгоритми та прийоми програмування, а також супроводжений докладними коментарями програмний код і пояснювальна записка будуть корисні починаючим програмістам і можуть бути використані при підготовці студентів зі спеціальності «Комп'ютерні науки».

Слід згадати, що програмне забезпечення розроблене в процесі виконання кваліфікаційної роботи не має великих вимог до ПК. Програма може працювати на усіх версіях ОС Windows починаючи з Windows 7.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Повних аналогів розроблюваного програмного забезпечення на сьогоднішній день не існує, найбільш схожа на нього програма RealFlow розроблена Next Limit (рис 2.1).

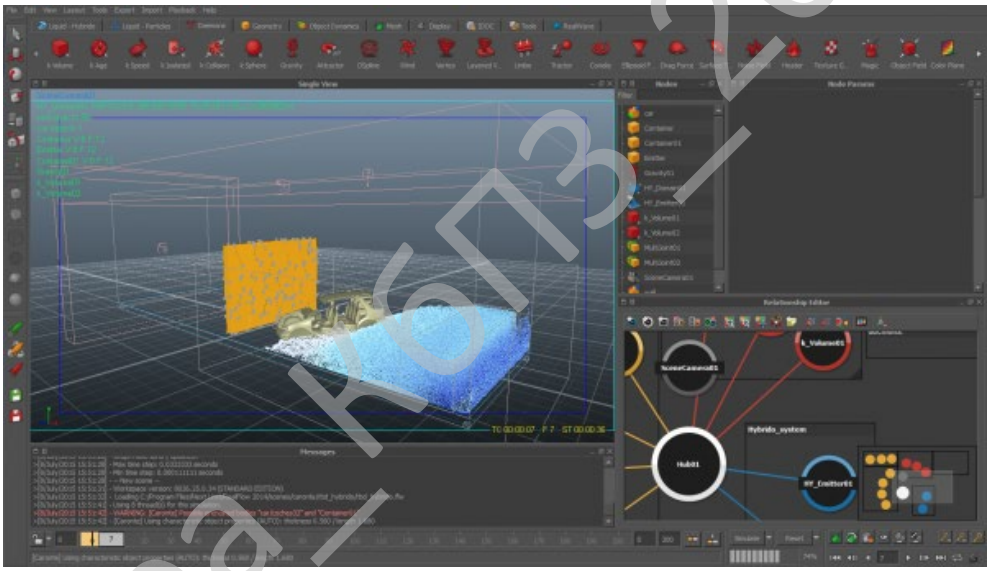

Рисунок 2.1 – Головне вікно RealFlow

Інтерфейс RealFlow розробники зробили зрозумілим і зручним на думку численних користувачів. Симулятор рідин – програма неординарна і унікальна, розробники цілком могли зробити щось з ун ікальним інтерфейсом. Але аніматор, який бажає розібратися з н овим інструментом, не повинен зазнавати труднощів при освоєнні програми. Тому вигляд RealFlow – це збірний інтерфейс додатків для

роботи з 3D . Найбільше він схожий на Maya з її наборами інструментів на вкладках.

У центрі вікна RealFlow – вікна проекцій, причому навігація в них т очно така ж, як і в більшості пакетів для створення тривимірної графіки (наприклад, в тому ж Autodesk Maya): утримування клавіші ALT з одночасним натисканням лівої кнопки миші призводить до повороту сцени у вікні проекції, ALT і натиснута права кнопка миші дозволяють наблизити або віддалити сцену і так далі.

Під вікнами проекцій розташована часова шкала для настройки анімації і кнопки для запуску процесу симуляції. У правій частині знаходиться редактор зв'язків. Він являє собою діаграму так званих вузлів сцени - компонентів і подій. Коли ви створюєте нову сцену або вносите зміни в уж е існуючий проект, ця діаграма автоматично змінюється. Крім того, користувач може за допомогою цієї діаграми самостійно налаштовувати зв'язку і керувати подієвої моделлю.

Над редактором подій знаходяться два вікна: вікно зі списком компонентів сцени і вікно з параметрами компонента, обраного в списку. Всі елементи інтерфейсу програми можна розташовувати на свій розсуд - вони легко переміщаються, акуратно вписуючись у вікно RealFlow і займаючи нове місце. Панелі інструментів також можна редагувати, змінюючи набори інструментів на них. Можна в тому числі і створювати власні панелі інструментів.

Для максимальної зручності в програмі є редактор гарячих клавіш - будь-який інструмент або команда можуть бути викликані натисненням однієї або декількох кнопок. Все це дуже легко і просто налаштовується - так само, як в серйозних 3D-редакторах. Всі призначені для користувача настройки записуються в профілі: окремо можна зберігати розташування елементів вікна, окремо - відредаговані поєднання гарячих клавіш.

У будь-якому потужному тривимірному редакторі присутня інструмент для роботи з ч астинками. Потрібно сказати, що цей інструмент дуже сильно змінився за короткий проміжок часу. Спочатку пакети для 3D -моделювання та анімації містили лише базові функції частинок: в сцену можна було додавати джерела

частинок з різною геометрією емітера, а в налаштуваннях цих джерел можна було встановити параметри випускаються об'єктів - форму, швидкість, тривалість життя і т.д.

Однак фантазія фахівців з в ізуальних ефектів не знала меж, і виникла необхідність появи нового класу частинок - «розумних», для яких можна було б налаштувати подієву модель, склавши її з л анцюжка умов для п отоку. Так з'явилися додатки Thinking Particles, Particle Flow та інші цікаві засоби для управління частками. Але всі вони не замінять RealFlow, яку також можна віднести до категорії «розумних» частинок. Програма від Next Limit орієнтована на рідинні ефекти, і в цьому її сильна сторона. Додаток включає в себе масу шаблонів і готових умов, характерних для сцен, де присутні текучі поверхні або потоки.

Найпростіші джерела частинок знаходяться на вкладці Liquid Particles. Однак, якщо ви додасте який-небудь джерело в сцену, самих частинок ви н е побачите. У в ікні проекції буде схематично відображена геометрія емітера, але частинки випускати не будуть, навіть якщо ви натиснете на Play для відтворення анімації. У звичайних 3D-редакторах частки летять за замовчуванням, варто тільки додати їх в пр оект. Однак RealFlow не просто візуалізує частки, а вирішує фізичну задачу (нехай навіть таку просту, як випускання частинок з одного джерела). Тому, щоб побачити рідинний потік частинок, потрібно натиснути на кнопку Simulate, запустивши тим самим процес прорахунку поведінки частинок.

Після завершення симуляції шкала анімації змінить свій колір на жовтий - цим кольором програма показує, який часовий проміжок сцени оброблений. Тепер анімацію можна відтворити кнопкою Play. Після симуляції RealFlow запише кеш в папки, які вказувалися при створенні нового проекту.

Зверніть увагу, що джерело часток не пов'язаний динамічно з потоком, що відображається у вікні проекції. У звичайному 3D-редакторі ви змінюєте положення джерела або його параметри, і це відразу позначається на анімації у вікні проекції. У RealFlow зміни видно тільки після того, як ви пр оведете чергову

симуляцію. Так що, якщо просто змінити положення джерела, у в ікні RealFlow прорахований потік частинок залишиться на місці.

Для обнулення кеша потрібно натиснути кнопку Reset - і прораховані частки зникнуть. Якщо кеш НЕ обнулити і спробувати заново зробити прорахунок з якогось кадру, RealFlow попередить про те, що дані в кеші будуть перезаписані.

Для того щоб аніматору було простіше працювати, в програму були включені найрізноманітніші джерела частинок. За кількістю і різноманітністю емітерів RealFlow може дати фору професійному 3D-редактору. В арсеналі симулятора рідин є такі типи джерел:

1. Linear - джерело з емітером-сплайном у вигляді прямої;
2. Spline - джерело в вигляді кривої довільної форми;
3. Cylinder - циліндричний джерело;
4. Triangle - джерело у вигляді трикутника;
5. Circle - плоский круглий емітер;
6. Square - квадратний джерело;
7. Sphere - сферичний джерело частинок.

Object - джерело частинок, створюваний на основі геометрії імпортованої в сцену моделі;

Fibers - дуже незвичайне джерело частинок, що дозволяє розмістити частки на «нитках», які виходять з в ершини об'єкта і розташовуються по нормалі до поверхні;

Bitmap - джерело частинок, характер випускання якого залежить від маски (чорно-біле зображення або секвенция таких зображень);

Fill Object - використання такого типу емітера дозволяє оточити поверхню об'єкта частинками.

Строго кажучи, RealFlow є не тільки симулятором рідинних ефектів, але ще і засобом для імітації будь-яких текучих об'єктів. Це означає, що область застосування даної програми виходить за рамки простого моделювання води або

рідинних ефектів. Тому в н алаштуваннях кожного джерела частинок ви м ожете знайти такий параметр, як тип частинок.

У розпорядженні користувача кілька основних профілів «плинності» частинок:

- Gas - тип частинок для імітації газових потоків;
- Liquid - варіант частинок для імітації рідин;
- Elastics - варіант джерела частинок, при якому потік не буде мати яскраво вираженим ефектом «злипання», тобто маса буде подібна гумі;
- Dumb - спрощений тип джерела частинок з м інімальним набором налаштувань;
- Script - тип частинок, який можна описати вручну за допомогою редактора скриптів і математичних формул.

Кожен джерело частинок має свій набір параметрів, причому цей набір змінюється в з алежності від обраного типу частинок. Так, наприклад, якщо ви використовуєте в джерелі частки типу Gas, то можете управляти настроюванням температури, а якщо вибрали для джерела варіант Elastics, то на температуру впливати не можете (для даного типу частинок це просто ні до чого), зате можете вказувати ступінь загасання потоку за допомогою параметра Damping.

Вкладка Liquid-Hybrido також містить набір джерел частинок, і вони в корені відрізняються від інших частинок RealFlow тим, щ о використовують алгоритм взаємодії, відмінний від SPH. Називається ця технологія Hybrido. Як вже було сказано вище, з її д опомогою можна виконувати прорахунок великих і об'ємних сцен, таких як, наприклад, водні поверхні. Дана технологія включає засоби для автоматичного створення вторинних сплесків і піни, а також туману.

WebGL Water

Ще один аналог, це простий симулятор води на веб-базі (рис 2.8).

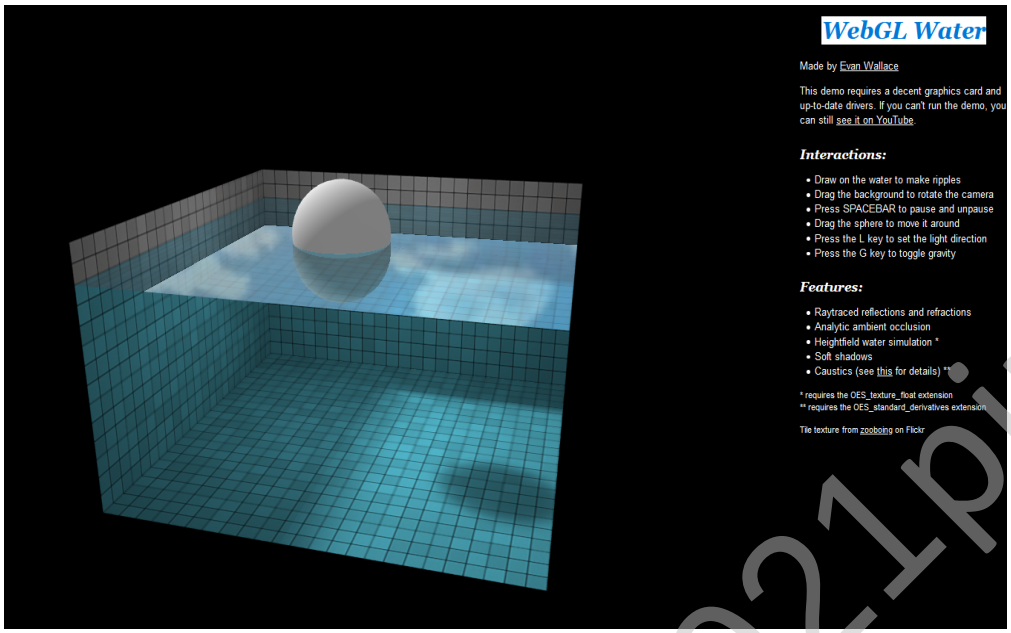

Рисунок 2.8 – Вікно WebGL Water

Особливості:

- Променеві відбиття та заломлення.
- Аналітична оклюзія навколишнього середовища.
- Моделювання води на висоті.
- М’які тіні.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки програми симуляції водного середовища була вибрана мова программування C# та середовище розробки Unity.

C# – сучасна об'єктно-орієнтована мова програмування високого рівня. Відноситься до широко відомого сімейства мов програмування C.

C# є об'єктно-орієнтованою мовою, але підтримує також і компонентно-орієнтоване програмування. Розробка сучасних додатків все більше тяжіє до створення програмних компонентів у формі автономних і самоописових пакетів, що реалізують окремі функціональні можливості. Головна особливість таких

компонентів в тому, що вони являють собою модель програмування з властивостями, методами і подіями. У них є атрибути, що надають декларативні відомості про компоненті. Вони включають в себе власну документацію. C# надає мовні конструкції, безпосередньо підтримують таку концепцію роботи. Завдяки цьому C# підходить для створення і застосування програмних компонентів.

Ось лише кілька функцій мови C#, що забезпечують надійність і стійкість додатків. Прибирання сміття автоматично звільняє пам'ять, зайняту недосяжними невикористовуваними об'єктами. Обробка винятків надає структурований і розширюваний підхід до виявлення помилок і їх відновлення. Тіпобезпечна структура мови унеможливлює читання з н еініціалізованих змінних, індексацію масивів за межами їх кордонів або виконання неперевірених привидів типів.

У C# існує єдина система типів. Всі типи C#, включаючи типи-примітиви, такі як int і double, успадковують від одного кореневого типу object. Таким чином, всі типи використовують загальний набір операцій, і значення будь-якого типу можна зберігати, передавати і обробляти схожим чином. Крім того, C# підтримує призначені для користувача посилальні типи і типи значень, дозволяючи як динамічно виділяти пам'ять для об'єктів, так і зберігати спрощені структури в стеці.

Щоб забезпечити сумісність програм і бібліотек C# при подальшому розвитку, при розробці C# багато уваги було приділено управлінню версіями. Багато мови програмування обходять увагою це питання. В результаті програми на цих мовах ламаються частіше, ніж хотілося б, при виході нових версій залежних бібліотек. Питання управління версіями істотно вплинули на такі аспекти розробки C#, як окремі модифікатори virtual і override, правила вирішення перевантаження методів і підтримка явного оголошення членів інтерфейсу.

У недавніх версіях C# були використані інші парадигми програмування. C# включає функції, що підтримують прийоми функціонального програмування, такі як лямбда-вирази. Інші нові можливості підтримують поділ даних і

алгоритмів, наприклад зіставлення шаблонів.

Є ще дуже багато важливої інформації про мову C#. Ці короткі огляди містять базові відомості про всі елементи мови і дозволять вам краще розібратися в тому, як влаштована ця мова:

- Структура програми: організаційна структура C# грунтується на таких поняттях, як програми, простору імен, типи, члени і збірки.

- Вирази створюються з операндів і операторів. Вирази повертають значення.

- Класи і об'єкти: класи є найважливішим типом в мові C#. Об'єкти представляють собою екземпляри класів. Класи створюються описом їх членів, які також описані в цій статті.

- Масиви: масив - це структура даних, що містить кілька змінних, доступ до яких здійснюється за який обчислюється індексам.

- Інтерфейси: інтерфейс визначає контракт, який може бути реалізований класами і структурами. Інтерфейс може містити методи, властивості, події і індексатори. Інтерфейс не надає реалізацію членів, які в ньому визначені. Він лише перераховує члени, які повинні бути визначені в класах або структурах, що реалізують цей інтерфейс.

- Делегати: тип delegate представляє посилання на методи з конкретним списком параметрів і типом значення, що повертається. Делегати дозволяють використовувати методи як сутності, зберігаючи їх у змінні і передаючи в якості параметрів. Принцип роботи делегатів близький до покажчиків функцій з деяких мов, але на відміну від покажчиків функцій делегати є об'єктно-орієнтованими і строго типізований.

- Атрибути: атрибути дозволяють програмам вказувати додаткові описові дані про типи, членах та інших сутності.

2.3 Розгорнута постановка завдання

Всі програми, представлені в огляді, виконують одну і ту саму функцію, і навряд чи можна сказати однозначно, яка краща, а яка - гірша. Багаті можливості RealFlow оцінять користувачі, що збираються робити високоякісні рендери зображень та відео, але не підійдуть початківцям, які просто цікавляться тим, як веде себе вода.

Досліджені ПЗ мають широкий функціонал, але також високі вимоги до параметрів комп'ютерної системи, закритий програмний код та високу ціну.

Аналіз існуючих програмних продуктів дозволив скласти основні вимоги до програмного забезпечення, яке буде розроблено в кваліфікаційній роботі:

1. система має працювати на персональних комп'ютерах невеликої потужності;
2. Повинна підтримуватись ОС Windows останніх версій;
3. Программа не повинна перевантажувати систему користувача;
4. Система повинна мати зручний та легкозрозумілий інтерфейс.

Розроблена в роботі система дозволяє без зайвих зусиль та вивчення складнрого інтерфейсу просто зрозуміти як буде проводитися симуляція води.

Для реалізації поставленої мети в роботі над проектом необхідно виконати наступні завдання:

1. Необхідно провести аналіз літературних та Інтернет-джерел з д аної тематики;
2. Дослідити існуючі програмні аналоги, визначити їх переваги та недоліки;
3. Проаналізувати засоби розробки програм і обрати найбільш відповіний даному проекту;
4. Розробити структурну схему системи;
5. Розробити функціональну схему системи;
6. Розробити діаграму процесів;

7. Розробити алгоритми роботи програмного забезпечення;

8. Розробити програмне забезпечення, яке б реалізувало наступні функціїї і мало такі властивості:

а) Зручна взаємодія з середовищем;

б) Дозволяє без зусиль вивчати фізичні явища водного середовища;

в) ПЗ повинно мати інтуїтивно зрозумілий інтерфейс і супроводжуватись технічним описом (довідкою);

г) ПЗ повинно бути розроблено на сучасній мові програмування високого рівня використовувати всі можливості сучасних середовищ розробки;

д) ПЗ повинно працювати під ОС сімейства Windows не нижче версії 7 x64.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Рідина є явищем, який не можна уявити статичним геометричним об'єктом. Поверхневі хвилі можуть мати досить складну форму, тому побудувати її реалістичне зображення нетривіальне завдання. Особливо гостро дана проблема проявляється при побудові великих відкритих просторів, таких як море або океан. Моделювання водної поверхні складається з двох частин: моделювання форми (або геометрії) водної поверхні і моделювання оптичних явищ, що відбуваються на поверхні води і в її т овщі. Очевидно, що для отримання візуально схожого зображення водної поверхні необхідно розглядати обидві частини явища. На даний час існує досить велика кількість моделей і методів синтезу поверхневого хвилювання. Всі відомі методи за точністю поділяються на дві категорії:

1. Перші використовують математичні моделі гідродинаміки. В основі таких методів лежать фундаментальні моделі обчислювальної гідродинаміки, такі як рівняння Нав'є-Стокса або різні хвильові моделі.

2. Другі використовують статистичні та емпіричні моделі. В основі таких методів лежать моделі, отримані емпіричним шляхом з певною метою, наприклад, візуальної подібності. Такі методи, як правило, виконують лише імітацію процесів реального світу і широко використовуються в р ізних комп'ютерних симуляторах, іграх, а також при створенні кінофільмів. Але наукова цінність даних методів невисока. За способом опису для анімації поверхневого хвилювання використовують сіткові методи, зокрема, методи, засновані на підході Ейлера. Сіткові методи використовують стаціонарну, найчастіше регулярну сітку, крізь яку рухаються частинки (малі обсяги) суцільного середовища, а всі фізичні характеристики визначаються в вузлах даної сітки, тобто вони не пов'язані з конкретними матеріальними частками, а в кожен момент часу є характеристиками

різних частинок, що знаходяться в даний момент в даній точці простору. Різних варіацій сіткових методів розроблено велику кількість. До найбільш використовуваних відносяться:

1. Такі, що використовують карти висот для представлення середовища. За допомогою цих методів моделюються хвильові процеси в двовимірному просторі, вирішуючи рівняння Нав'є-Стокса або інші рівняння лише для двох вимірів або використовуючи уявлення середовища у вигляді набору стовпів рідини. Процес візуалізації описується полігональною графікою без проміжних перетворень. Дана група підходів має безліч обмежень і їх неможливо використовувати при описі процесу обвалення хвиль.

2. Такі, що використовують тривимірні сітки. Опис динаміки рідкого середовища відбувається з використанням тривимірної сітки. Як правило, застосовуються методи засновані на вирішенні повних рівнянь Нав'є-Стокса і рішення мають високу точність. Такі методи застосовуються в наукової візуалізації, але істотним мінусом є величезна обчислювальна складність, що не дозволяє застосовувати такі методи в додатках реального часу за прийнятною розмірністю сітки. Методи цього класу дозволяють розраховувати завдання з великими деформаціями і найчастіше застосовуються для задач гідро- і газодинаміки.

Головна мета підходів, які працюють в просторової області, представити геометрію поверхні середовища, використовуючи суму періодичних функцій, що розвиваються у часі використовуючи зрушення фаз. Велика частина з відомих методів заснована на аналітичній моделі суперпозиції хвиль, і рішення тут або задається відразу у вигляді лінійної комбінації тригонометричних функцій зі спеціально підібраними коефіцієнтами, або виходить в результаті застосування зворотного перетворення Фур'є із спеціально заданим спектром. Найбільш успішну модель запропонував Джеррі Тессендорф. Реалізація хвилювання водної поверхні має на увазі створення поля висот, що складається з набору синусоїд, з різними амплітудами і фазами.

У методі Тессендорфа, пропонується генерувати ці амплітуди і фази, ґрунтуючись на емпіричних даних океанографії, а потім за допомогою швидкого перетворення Фур'є, отримати суму відповідних синусоїд. На математичних методах ґрунтуються інструменти тривимірної анімаційної візуалізації. Так, вільний пакет для створення тривимірної графіки Blender містить реалізацію стабільного методу сіткових рівнянь Больцмана для моделювання рідини, пакет Maya дозволяє використовувати шейдера, засновані на карті висот, FumeFX – плагін до 3ds Max, заснований на внутрішній технології VoxelFlow, яка використовує нестискувані рівняння Ейлера для маси і збереження імпульсу, Houdini Ocean Toolkit – плагін до пакету Houdini, заснований на методі Тессендорфа і т.д. Інструментарій HOT складається з ш ейдера для поверхні і деформуючого елемента.

Шейдер виводить векторну карту зміщення, яка використовується під час рендерінгу, деформатор модіфікує фігуру у вікні перегляду. Вплив деформатора відпрацьовується на зображенні з низьким розрішенням поверхні океану, яке потім при рендері можна об'єднати з п оверхнею океану з б ільш високим розрішенням Прикладом механізму симуляції необмеженої водної поверхні може бути інструмент Maya Ocean System, засновані на флюїдах За допомогою Ocean System можна створювати і анімувати морську поверхню в будь-яку погоду, так само як явища, що відбуваються на поверхні невеликого ставка, басейну або океану. Поверхня рідини генерується і відображається як полігональна сітка, а геометрія океану задається картою висот та зміщень. Карти висот і векторні карти зміщення, використовує Шейдер океану для отримання графічного зображення

Unity3D – кросплатформенних ігровий движок від компанії Unity Technologies. Історія створення движка досить цікава і повчальна. Цікава, бо двоє хлопців захотіли зробити гру, але для цього їм не підходили існуючі інструменти. І вони вирішили зробити свій движок, а потім вже робити на ньому гру. І після того, як вони зробили движок, вони зрозуміли, що їм не так-то й цікаво робити гри, а більше подобається займатися безпосередньо движком. Так і почалася історія

одного з найвідоміших і потужних двигунів. А повчальна ця історія тому, що ніколи не знаєш, чим обернеться ту чи іншу починання.

Unity - безкоштовний движок. Обмеження - при запуску гри п оказується логотип Unity. Купивши розширену версію, ви п озбудетеся і від логотипу. Безкоштовність движка - це те, що привернуло багатьох до розробки ігор на нього.

Unity використовує компонентно-орієнтований підхід. Все в грі - це об'єкт, куди додані різні компоненти. Наприклад, якщо ми робимо платформер, ми додаємо GameObject, і до цього GameObject додаємо графічний компонент (для відтворення гравця) і компонент управління (щоб можна було управляти гравцем клавіатурою або мишкою). Таких різних компонентів можна додати будь-яку кількість до будь-якого GameObject. Тобто, створення гри в Unity - це додавання GameObject-ів, і додавання їм корисних компонентів.

Скрипти для Unity на мові C#. Також є J avaScript і Boo (урізана версія Python), але реально використовується лише C#. Написання своїх компонентів - це досить складне заняття. Фактично, це звичайне програмування.

Unity - хороший вибір для створення середніх по складності проектів як для ПК, так і для мобільних пристроїв. Велика кількість готових Ассет, включаючи скрипти, дуже цьому допомагає. Ну і велика спільнота - це теж добре, вам допоможуть вирішити будь-якої затикаючи, якщо він виникне.

Один з козирів Unity - це список підтримуваних платформ, де може запускатися програми. Unity працює майже всюди - на ПК (всі операційні системи), на Андроїд, на iOS, на SmartTV, в браузері, на різних екзотичних системах - наприклад, Tizen OS. Але якщо працювати з чимось специфічним, наприклад, низькорівневим доступом до заліза в тому ж Андроїді - доведеться писати частина коду на Java, потім компонувати все це з Unity. Аналогічно з iOS. Також, зібрати додаток під iOS можна лише з-під MacOS X. Тобто, не маючи макбуков або чогось схожого, гру на iOS ви так просто не випустіть. Це не недолік Unity, це обмеження Apple.

Що стосується процесу створення ігор, то тут або Windows, або Mac OS X. Є експериментальні збірки редактора під Linux, але поки що вони досить глючний. Всі створення гри відбувається в редакторі Unity, редагування коду скриптів можна робити або в MonoDevelop (йде за замовчуванням), або використовувати сторонній редактор. Багато хто використовує Visual Studio. Деякі налаштовують для цих цілей Sublime Text.

Потужний плюс Unity - це Ассет. Все в грі, включаючи код, картинки, представляється Ассет (Asset). Asset можна експортувати, імпортувати. Таким чином, сторонні розробники можуть робити цілі заготовки для ігор. Все, що залишиться - це замінити картинки, підправити скрипти - і можна релізіти гру. Але різні Asset можуть бути несумісні між собою як в прямому сенсі, так і не підходити по стилю, але це вже деталі.

Є спеціальний онлайн-магазин - Unity Asset Store. Там продаються готові Ассет від сторонніх розробників. Будь-який бажаючий може зробити свій Ассет, і викласти його в продаж в ць ому магазині. Деякі люди зробили на цьому цілий бізнес, завдяки великому ринку Unity-користувачів. Також важливий момент, що магазин доступний прямо з редактора Unity. Тобто, додавання нових Ассет максімлально спрощується.

Наступною крутий фішкою Unity є ком'юніті. Воно величезне. Якщо виникає якесь питання, скоріше за все, воно вже багато разів ставилося, і стільки ж раз вже було вирішене. І можна знайти відповідб на нього на профільних форумах, на StackOverflow. А якщо ж знайшовся питання, на яке не вдалося знайти відповіді – можна поставити його на офіційному форумі Unity, і з великою ймовірністю відповідь дадуть у той же день. Це величезний плюс рушія в порівнянні з іншими.

Звичайно, є і мінуси. Перший недолік – це повільна робота. Якщо порівнювати з іншими двигунами, той же LibGDX або Cocos2D-X, Unity повільний. У цих движків різні цілі, і Unity орієнтований на проекти побільші, але факт залишається фактом. Для м аленької 2D гри, будь -якого платформера, Unity буде працювати повільніше, ніж альтернативи.

Наступне - це великий розмір програми. Великий - це значить, що якщо зібрати порожній проект з н астройками за замовчуванням для Андроїд, ви отримаєте інсталяційний файл близько 20 мегабайт. Для ПК ця цифра буде близько 100 мегабайт. Для в еликих проектів розміром в г ігабайти це не страшно, але ось для маленьких Андроїд-ігор, де вся графіка і звуки займають п'ять мегабайт, тягнути додаткових 20 мегабайт може бути неприємно.

Ще один мінус - це як би не мінус Unity, але пов'язаний з ним. Орієнтація на об'єкти і скрипти, що прикріплені до цих обьектам, підштовхує розробника створювати погану архітектуру. Додати нову можливість здається такою простою справою. Всього-то написати скрипт, і причепити його до об'єкту. Але з ростом проекту зв'язку між скриптами і об'єктами ускладнюються, і додавати нові фічі стає все важче і важче, гра стає більш повільною і глючной. Це чимось нагадує ситуацію з D elphi і чіпляням обробників подій на кнопки. Зрозуміло, що і на Unity можна писати інакше, контролювати кількість скриптів і зв'язку між ними. Більш того - великі проекти так і пишуть. Але недосвідчені розробники додають скрипти на об'єкти, а рушій не рахує це помилкою.

Якщо необхідно написати свое ПЗ, але немає бажання писати багато скриптів - Unity підійде. Накидати об'єктів, і пов'язати їх між собою вийде і без знання мови програмування на перших етапах розробки.

Microsoft Visual Studio - лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в р ідному, так і в керованому кодах для в сіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone, Android, IOS, .NET Compact Framework і Silverlight. Підтримує наступні мови: Visual Basic, C ++, C#, F #.

IntelliSense. Технологія автодоповнення Microsoft. Дописує назву функції при введенні початкових літер. Крім прямого призначення, IntelliSense використовується для доступу до документації та для усунення неоднозначності в іменах змінних, функцій і методів, використовуючи рефлексію.

Code Anilizer. Функціонал, який допомагає знайти помилки в коді. Суміщений з IntelliSense, тим, що всі помилки, повідомлення, потенційні помилки підсвічуються. Також вся ця інформація відображається у вікні "Error List".

Perfomance Analizer. Інструмент, що відображає витрати ресурсів при роботі додатка / сервісу у вигляді статистики і графіків.

Test Manager. Вбудований менеджер тестів. Після створення тесту можна за допомогою спеціального вікна запускати і настроювати тести.

Extension / Updates Manager. Менеджер плагінів, адаптерів, провайдерів. Дозволяє легко знайти, встановити, оновити будь-яке доповнення.

Nuget. Система управління пакетами для платформ розробки Microsoft, в першу чергу бібліотек .NET Framework. Управляється .NET Foundation. Зручна установка бібліотек в будь-який .Net проект.

Git Manager. Вбудований менеджер контролю версій. Спочатку працював тільки з Team Foundation Server. Зараз можна підключити Team Explorer (Назва менеджера) до будь-якого сховища. Присутні всі необхідні функції для роботи з git без запитів.

Archivator. Архіватор проектів. Після того, як проект готовий, потрібно зібрати виконуваний файл. Для кожної технології реалізований свій архіватор. Не потрібно встановлювати окремий софт, щоб зробити установочнік.

File Manager. Для додавання нового файлу в проект існує вбудований менеджер файлів. Зручне створення будь-яких файлів на основі шаблонів. Реалізована велика кількість стандартних шаблонів (Приклад: клас). Також можна додавати свої. При установці нової технології - додаються відповідні шаблони.

Views. Велика кількість різних вкладок для в ідображення різної корисної інформації, на зразок списку "GOTO", або відображення даних об'єкта в Debug режимі.

Customization. Можливість змінити зовнішній вигляд Visual Studio під себе. Зміни кольорів, теми, шрифтів, відступів і т.д. Розташування вікон в зручному вам вигляді.

Setting. Налаштування всього вище перерахованого функціоналу. Налаштування швидких клавіш, повідомлень, швидкий запуск, стартового вікна, вкладок, розмітки мов і багато іншого

Завдяки величезній кількості налаштувань, підтримуваних технологій, швидкодії і зручності Visual Studio вважається однією з кращих середовищ розробки. З мінусів можна виділити величезну вагу пакетів технологій.

3.2 Розробка структурної схеми

Структурна схема (рис 3.1) розбита на три блоки перерахуємо їх з низу вгору:

а) Нижній блок функціональної схеми це бл ок роботи бази даних в який укладені основні функціональні можливості програмного забезпечення.

б) Середній блок функціональної схеми описує взаємодію основного модуля програми з функціональними модулями і взаємодією з верхн ім і нижнім функціональним блоком схеми.

в) Верхній блок відповідає за вивід зображення, яке було створено в симуляції та виведено через драйвер на екран користувача

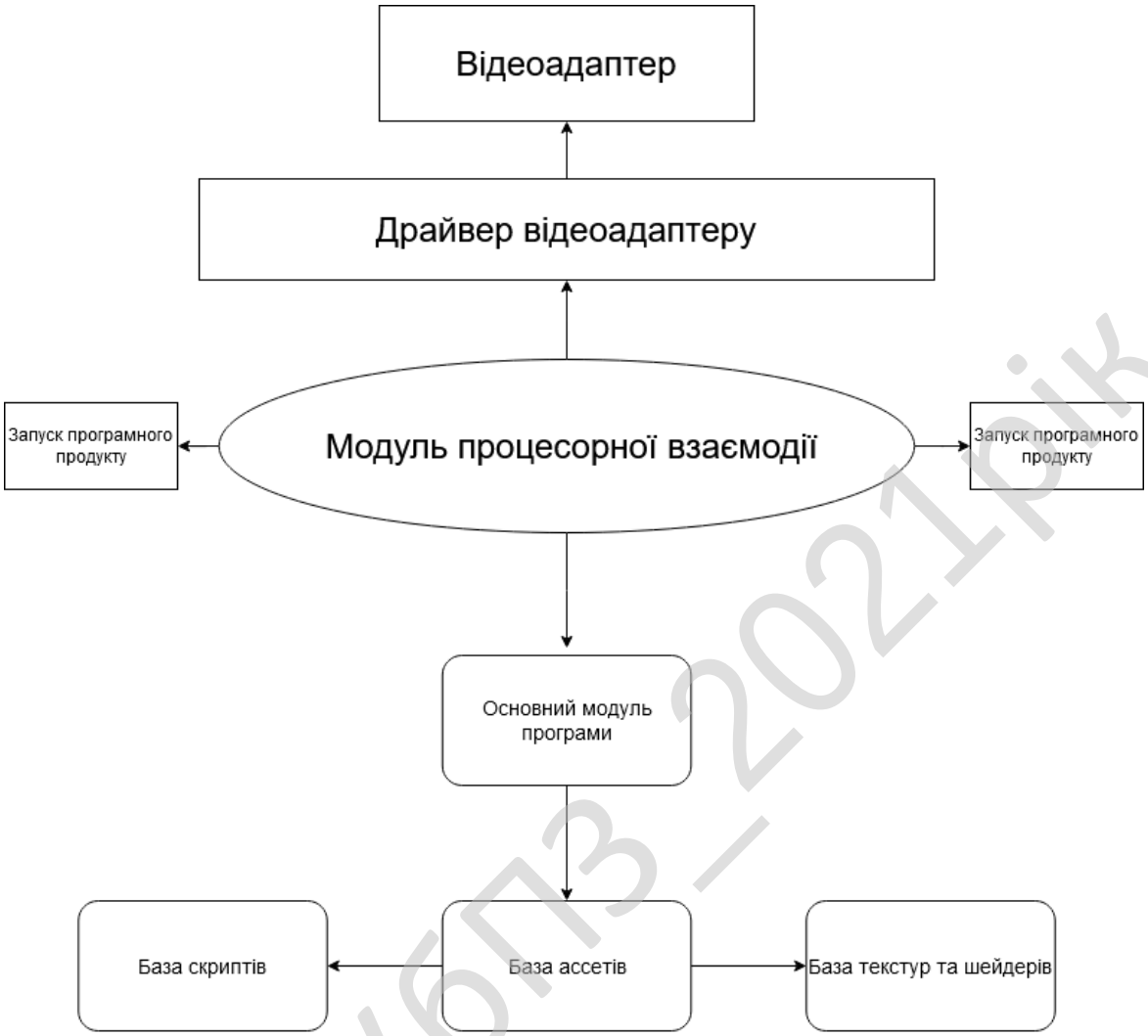

Рисунок 3.1 - Структурна схема системи

3.3 Розробка функціональної схеми

При розробці функціональної схеми (рис. 3.2) програмного забезпечення були розглянуті основні модулі програми, скриптові бази, взаємодія з ком'ютером користувача.

Структурна схема розділена на три блоки, а саме:

1. Файли ассетів.

2. Скрипти симуляцій.

3. Взаємодія з комп'ютером.

У першому блоці (з ліва на право) розглянута будова таблиць програми, 2D і 3D, а також основні скрипти симуляцій.

Файли ассетів. Ассет – основна одиниця взаємодії с ігровим движком Unity.

Liqid2d – ассет в як ому знаходяться скрипти,текстури,шейдери та положенна в 2d просторі симуляції,яка буде виконуватися після її виклику

Скрипт - мова програмування, розроблена для запису «сценаріїв», послідовностей операцій, які користувач може виконувати на комп'ютері. Прості скриптові мови раніше часто називали мовами пакетної. Сценарії зазвичай інтерпретуються, а не компілюються.

Скрипти – це “міні” программи які автоматизуть деяке завдання, яке без сценарію користувач робив би вручну, використовуючи інтерфейс програми. Мови таких скриптів спочатку орієнтувалися на використання як внутрішні керуючі мови у складних системах. Багато хто з них, проте, вийшли за межі сфери свого початкового застосування і використовуються нині в зовсім інших областях. Характерними особливостями даних мов є, п о-перше, їх інтерпретованість (компіляція або неможлива, або небажана), по-друге, простий синтаксис, а по-третє, легка розширюваність. Таким чином, вони ідеально підходять для використання в часто змінюваних програмах, дуже невеликих програмах або у випадках, коли для виконання операторів мови витрачається час, незрівнянний із часом їх розбору.

Текстура – це спосіб надання поверхні 3D або 2D деталі (полігону): кольору, блиску, матовості, фактури та інших фізичних властивостей (для імітації найчастіше якогось природного матеріалу, наприклад: дерева, паперу, металу, каменю тощо).

Поняття «текстура» є в ажливим елементом 3D-моделювання, оскільки дозволяє відтворити також малі об'єкти поверхні, створення яких полігонами виявилося б н адмірно ресурсомістким. Наприклад, шрами на шкірі, складки на одязі, дрібні камені, предмети на поверхні стін і ґрунту та багато іншого. Отже, текстура використовується для заповнення поверхонь об'єктів і як шар для додання певного ефекту або зміни геометрії всьому зображенню або його частини.

Шейдер – програма для одного із ступенів графічного конвеєра, що використовується в тривимірній графіці для визначення остаточних параметрів об'єкта чи зображення. Вона може включати в себе довільної складності опис поглинання та розсіювання світла, накладення текстури, віддзеркалення і заломлення, затінення, зміщення поверхні і ефекти пост-обробки.

Програмовані шейдери гнучкі та ефективні. Складні на вигляд поверхні можуть бути візуалізовані за допомогою простих геометричних форм. Наприклад, шейдери можуть бути використані для малювання поверхні із тривимірної керамічної плитки на абсолютно плоскій поверхні.

В програмних графічних рушіях весь ланцюжок рендерингу – від визначення видимих частин сцени до накладення текстури – створювався розробником ігор. У ц ей ланцюжок можна було включати власні нестандартні відеоефекти. З появою перших відеоакселераторів розробник виявився обмежений тим набором ефектів, який закладений в апаратне забезпечення.

Шейдери були розроблені для того, щоб мати змогу відтворювати складні відеоефекти з атомарних операцій. Попередниками шейдерів були процедурна генерація текстур і мультитекстурування. Але і ці механізми не забезпечують такої гнучкості, як шейдери.

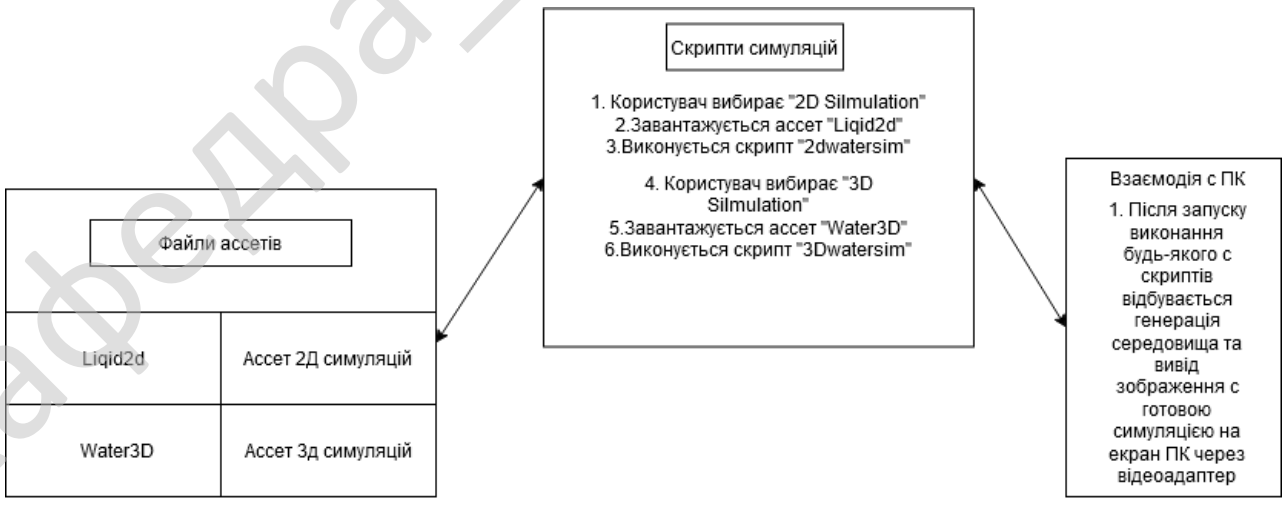

Рисунок 3.2 - Функціональна схема

3.4 Розробка діаграми процесів

На діаграмі процесів системи (рис. 3.3) відображено взаємодію процесів запропонованої системи симуляції водних середовищ та порядок дій, що відбуваються під час її роботи.

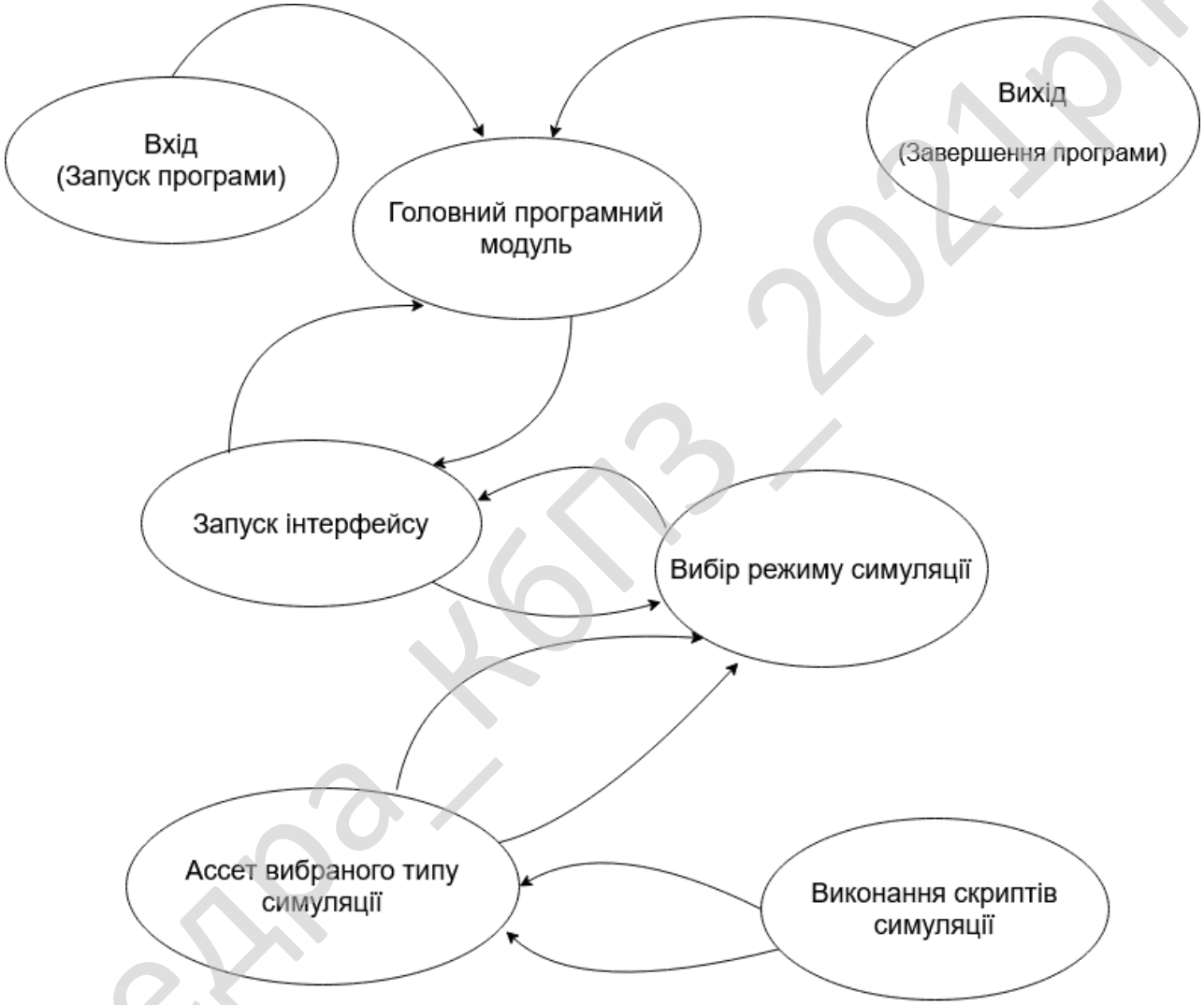

Рисунок 3.3 – Діаграма процесів системи

Головними процесами у системі є: запуск інтерфейсу, вибір режиму симуляції, ассет вибраного режиму симуляції та виконання скриптів симуляції.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

При розробці блок-схеми основної програми, були виділені такі основні стадії роботи алгоритму як:

1) Ініціалізація. Первинна стадія роботи програми, на даному етапі відбувається ініціалізація призначених для користувача типів даних і відповідно і ініціалізація змінних і масивів.

2) Блок інтерфейсу користувача. На стадії виводу інтерфейсу програма генерує інтерфейс с збережених ассетів та виводить його на монітор користувача

4) Блок роботи програмного забезпечення. Основний блок роботі програмного забезпечення – це робота з ассетами, а вже ассети взаємодіють з підключеними до них програмних засобів. Також доданий блок з інформацією про розробника,який показує всю інформацію про нього

5) Завершення роботи програми. Блок завершення програми винесений також в окремий модуль, в як ому перед завершення роботи програми відбувається звільнення занятої оперативної пам’яті та безпечний вихід з ПЗ.

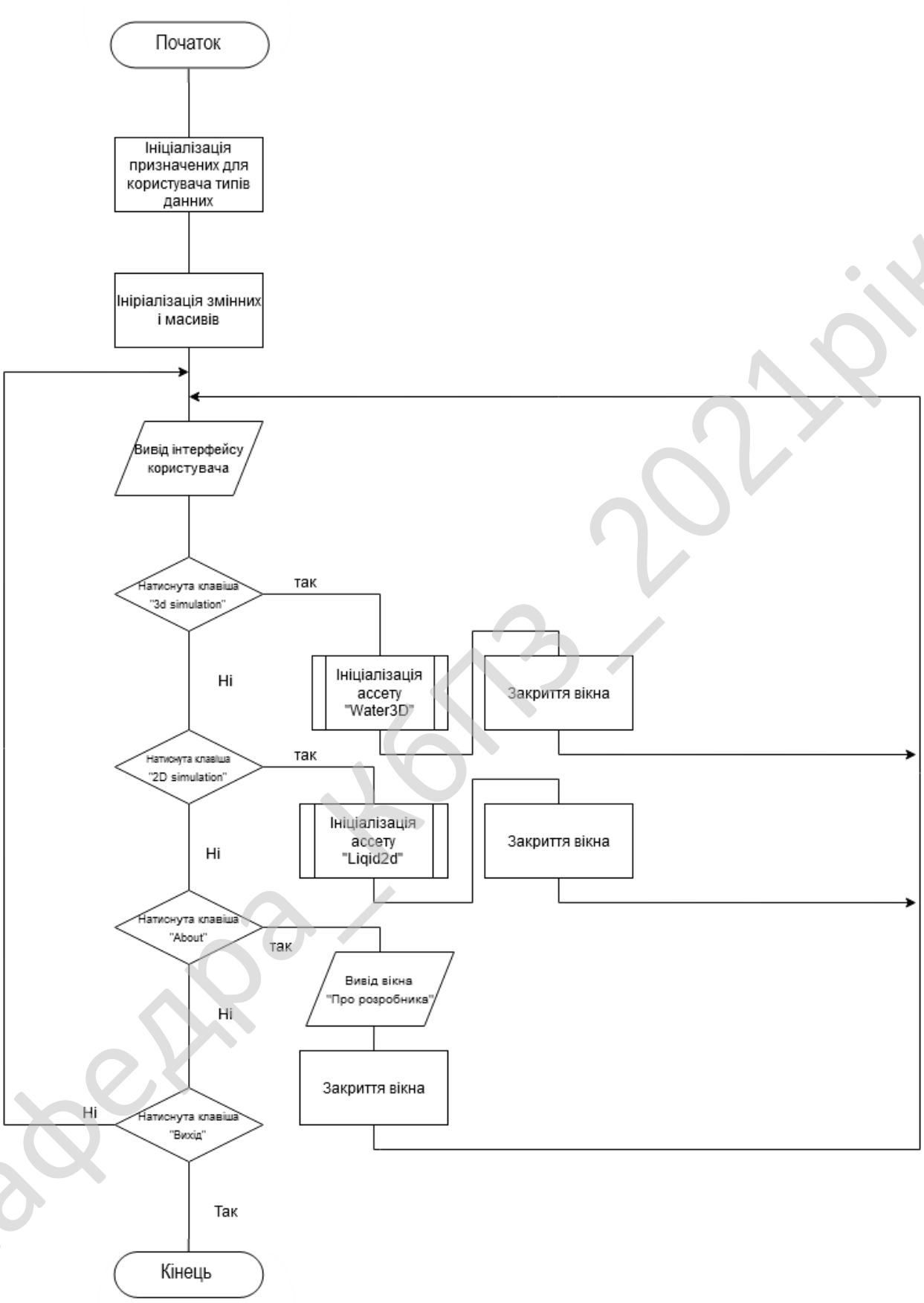

Рисунок 4.1 - Блок-схема головного модулю

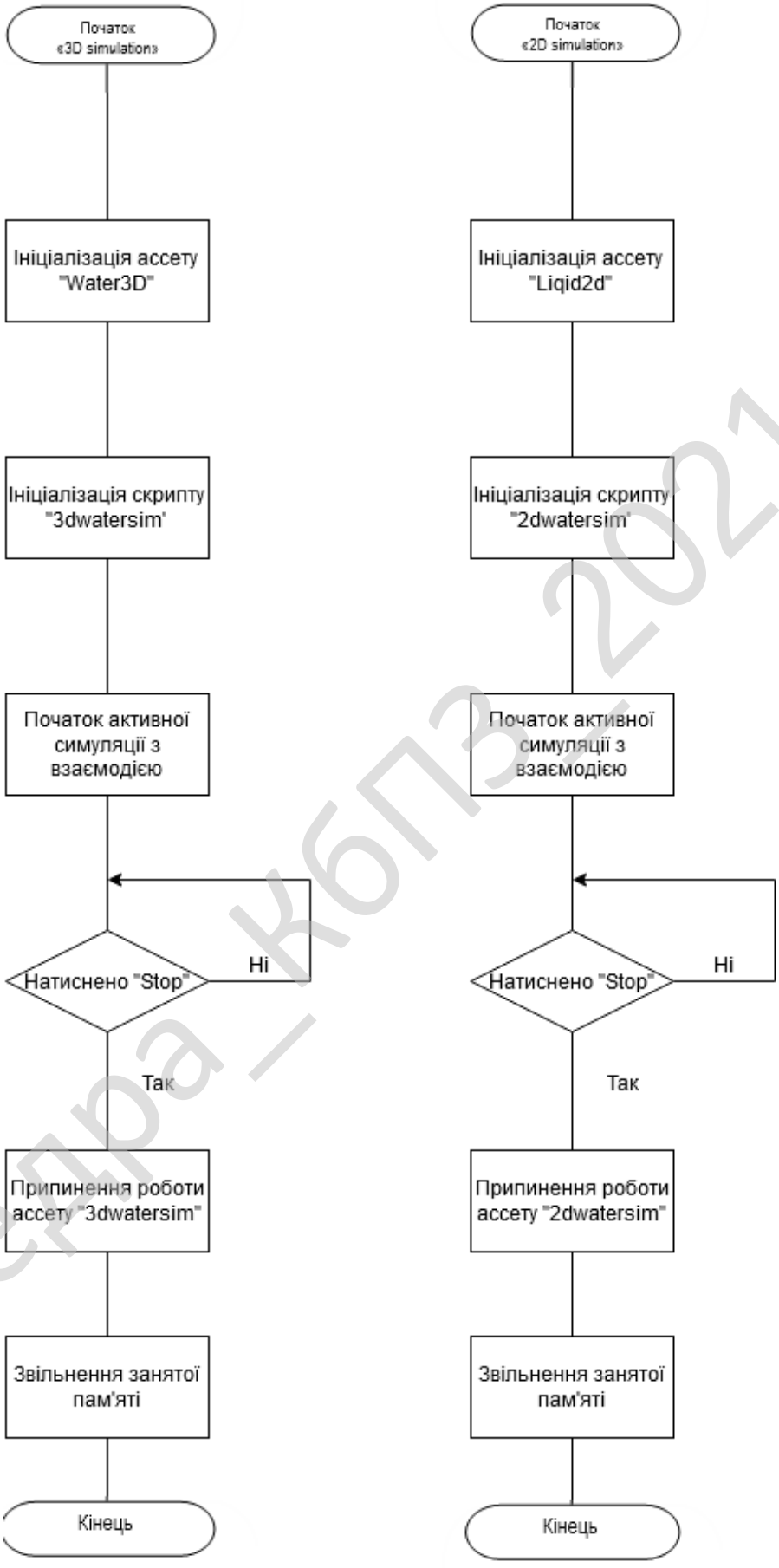

Рисунок 4.2 – Блок-схеми процедур системи

Розроблене програмне забезпечення служить універсальним інструментом в руках користувача, який хоче дізнатись більше про поведінку води в р ізних середовищах та у різних вимірах віртуального середовища

Програма призначена для створення віртуального середовища у я к ому рідина,у данному випадку – вода,може показувати себе як у реальному світі (рис. 4.1). Таким чином, здійснюється швидкий доступ до вбудованих в програму ассетів, які в свою чергу вже отримуюсь доступ до всього необхідного для них: скриптам, шейдерам, текстурам.

Рисунок 4.3 – Головне меню програми

Програма дозволяє автоматизувати процес симуляції рідин в віртуальному середовищі для 2D на 3D відображення

Програмне забезпечення працює автономно, на базі ігрового рушія “Unity” без високих вимог до конфігурації комп'ютера. Її автономність полягає у тому, що ПЗ не потребує інсталяції та буде працювати просто після його розархівування в будь-яке звурне місце для користувача. Програмне забезпечення

просте та не вимагає від користувача сильних познань в т емі симуляцій для роботи з моїм ПЗ

Користувачу залишиться тільки виконувати необхідні дії в з аданій послідовності.

Перерахуємо можливості розробленого програмного забезпечення:

1) Забезпечує швидкий доступ симуляцій;
2) Використовується для створення 2д та 3д;
3) Містить всі дані в одному місці;
4) Можливість автоматичного підключення всіх ассетів;
5) Автономна робота - інсталяція не є обов'язковою;
6) Не вимоглива до конфігурації комп'ютера;
7) Можливість завдання декількох програм в одному додатку бази;
8) Простота в роботі (обігу);
9) Створення бази для абсолютно будь-якого носія;
10) Робота без інтернет-з’єднання;
11) Інтеграція з інтернет-ресурсами;
12) Проста робота с частинками для симуляцій;
13) Просте на зрозуміле меню;
14) Гнучкі можливості настройки;
15) Можливість автоматичного складання бази шляхом вказівки основних параметрів;
16) Стабільна робота у всіх версіях MS Windows;
17) Можливість складання симуляцій будь-якої складності
18) Можливість створення і редагування власних файлів мов;
19) Робота програми з програмним забезпеченням, що заносяться, на відносних шляхах;.

4.2 Захист розробленого програмного забезпечення

Інформація стає все більш уразливою з різних причин:

а) зростаючі об'єми даних, що зберігаються та передаються;

б) розширення круга користувачів, що мають доступ до ресурсів ЕОМ, програм і даних;

в) ускладнення режимів експлуатації обчислювальних систем.

Тому все велику важливість вносить проблема захисту інформації від несанкціонованого доступу (НСД) при передачі і зберіганні. Суть цієї проблеми - постійна боротьба фахівців із захисту інформації з своїми "опонентами".

Характеристики складових алгоритмів шифрування це:

а) Назва алгоритму;

б) Розмір ключа біт;

в) Розмір блоку біт;

г) Розмір вектора ініціалізації біт;

д) Кількість циклів шифрування.

Захист інформації - сукупність заходів, методів і засобів, що забезпечують:

1) Виключення НСД до ресурсів ЕОМ, програм і даних;

2) Перевірку цілісності інформації;

3) Виключення несанкціонованого використовування програм (захист програм від копіювання).

Очевидна тенденція до переходу на цифрові методи передачі і зберігання інформації дозволяє застосовувати уніфіковані методи і алгоритми для з ахисту дискретної (текст, факс, телекс) і безперервної (мова) інформації.

Випробуваний метод захисту інформації від НСД - шифрування (криптографія). Шифруванням (encryption) називають процес перетворення відкритих даних (plaintext) в зашифровані (шифртекст, ciphertext) або

зашифрованих даних у відкриті за певними правилами із застосуванням ключів. У англомовній літературі шифрування та розшифрування - enciphering/deciphering.

За допомогою криптографічних методів можливо:

а) шифрування інформації;

б) реалізація електронного підпису;

в) розподіл ключів шифрування;

г) захист від випадкової або умисної зміни інформації.

До алгоритмів шифрування пред'являються певні вимоги:

а) Високий рівень захисту даних проти дешифрування і можливої модифікації;

б) Захищеність інформації повинна ґрунтуватися тільки на знанні ключа і не залежати від того, відомий алгоритм чи ні;

в) Мала зміна початкового тексту або ключа повинна приводити до значної зміни шифрованого тексту (ефект "обвалу");

г) Область значень ключа повинна виключати можливість дешифрування даних шляхом перебору значень ключа;

д) Економічність реалізації алгоритму при достатній швидкодії;

е) Вартість дешифрування даних без знання ключа повинна перевищувати вартість даних. Сьогодні в кріптології прийнято оперувати символами не у вигляді букв, а у вигляді чисел, ним відповідних. Так, в л атинському алфавіті можемо використовувати числа від 0 (відповідного A) до 25 (Z).

DES (Data Encryption Standard - Стандарт Шифрування Даних) - назва Федерального Стандарту Обробки інформації (FIPS). В термінах ANSI DEA визначений як стандарт X9.32.

DEA - розвиток алгоритму Lucifer, який був р озроблений на початку 1970-их років компанією IBM; на завершальних стадіях розробки активна участь приймала NSA і NBS. З м оменту публікації DEA (відоміший як DES), широко вивчався і відомий як один з кращих симетричних алгоритмів см рисунок 4.9.

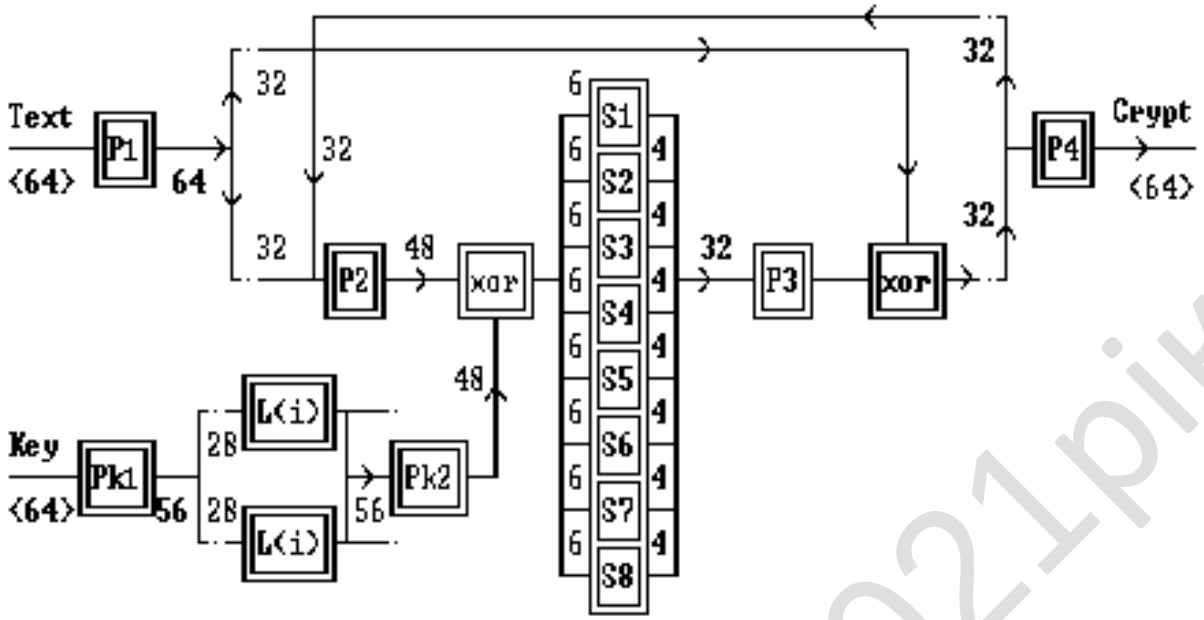

Рисунок 4.7 - Схема алгоритму

де:

Text початковий текст (блок 64 біти)

Crypt зашифрований блок

Key 64-х розрядний ключ

числа розрядність на даній вітці алгоритму

P, Pk перестановки

S підстановка 6 біт -> 4 бита

L(i) зрушення (i -- номер ітерації)

xor складання по модулю 2

конкатенація бітових рядків, причому верхня - спереду

розбиття рядка на дві, причому перша - вгорі

обмежена крапками ділянка повторюється 16 разів

Перестановки виконуються по звичній формулі D[i]=S[P[i]], де

S початковий рядок (масив символів, нумерація з одиниці)

D результат перестановки (масив символів, нумерація з одиниці)

P таблиця перестановок (масив індексів в рядку S)

S - підстановка 6->4. У в ідповідність шести бітам ставиться чотири. Підстановка виробляється за наступним правилом: хай початковий бітовий рядок - /abcdef/, тоді /af/ - номер рядка, а /bcde/ - номер стовпця. Рядок і стовпець визначають місцезнаходження результату в S-таблиці. Наприклад, при використовуванні таблицю S6, число 58 (111010) переводиться в 13 (1101).

Крім звичного його застосування, цей алгоритм можна використовувати для створення "односторонніх" функцій; Для цього ключ і початковий текст міняються місцями: у формулі Crypt=DES(Text,Key) початковий текст може бути несекретним і фіксованим. А Crypt розглядається як функція ключа - Key. DEA оперує блоками 64-бітового розміру і використовує 56-бітовий ключ (8 п арних бітів повного 64-бітового ключа не використовуються). DEA - симетрична криптосистема, визначена як 16-раундовий шифр Фейстеля (Feistel) була спочатку призначена для апаратної реалізації. Коли DEA використовується для передачі інформації, то щоб зашифрувати і розшифрувати повідомлення або щоб створити і перевірити код достовірності повідомлення (MAC) відправник і одержувач повинні знати секретний ключ. DEA може також використовуватися одним користувачем, наприклад для шифрування файлів на жорсткому диску. У розрахованому на багато користувачів середовищі організувати захищений розподіл ключа складно; ідеальне рішення цієї проблеми пропонує криптографія загального ключа.

FIPS 46-3 містить також визначення Triple DES (DES потрійної довжини) (TDEA, відповідно X9.52). Найближчим часом DES і Triple DES будуть з амінені алгоритмом AES (Advanced Encryption Standard - Розширений Стандарт Шифрування). Алгоритм DES широко застосовується для захисту фінансової інформації: так, модуль повністю підтримує операції TripleDES для емісії і обробки.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програмне забезпечення призначене для створення віртуально симуляції рідини, в даному випадку – води, розпочинається с запуску .exe файлу в папці, яку разпакував користувач у зручному йому місці. Після запуску ПЗ користувач бачить головке меню програми (рис 5.1).

Рисунок 5.1 – Головне вікно програми

Для початку роботи потрібно вибрати,що сами цікавить користувача: 2д,чи 3д симуляція водного вередовища.

Перерахуємо функціональні кнопки програмного забезпечення, головного вікна програми.

а) 3D simulation (рисунок 5.3).

Викликає симуляцію водного середовища у 3х вимірному полі (рис. 5.2).

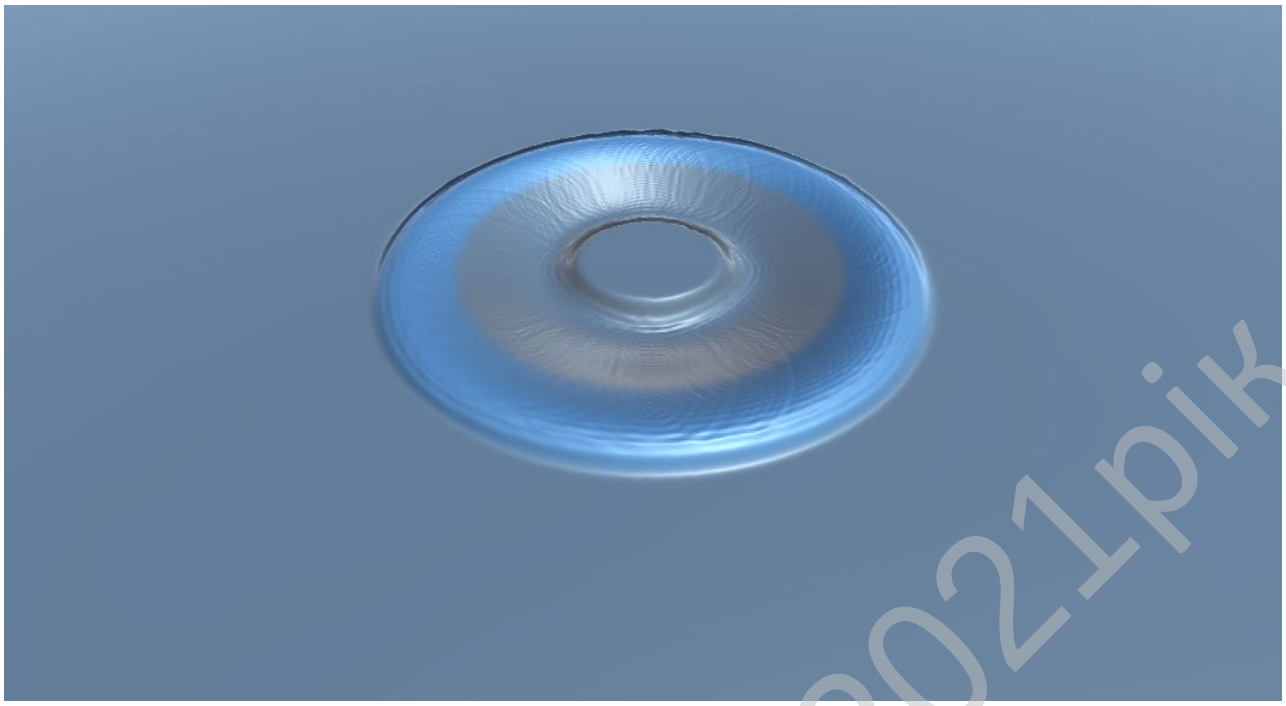

Рисунок 5.2 - Активна симуляція поверхні води у 3Д

Користувач може створювати хвилі,користуючись вказівником миші.

Коли користувач натискає ліву,або праву кнопку миші,утворюється невидимий об’єкт,який піднімається над поверхнею води,таким чином утворюючи різні хвилі (рис 5.3).

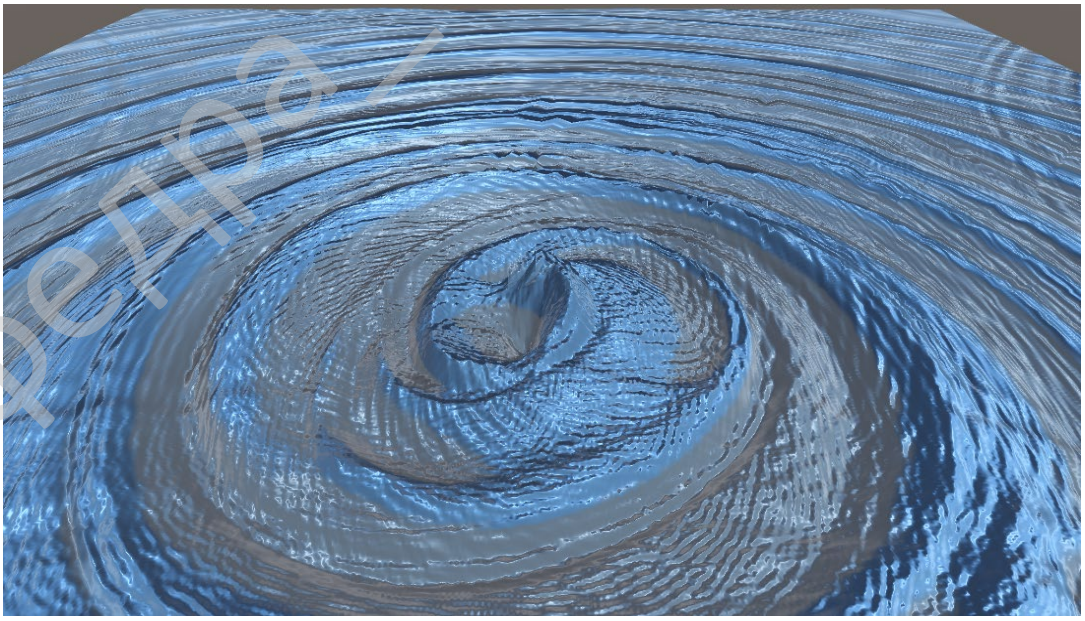

Рисунок 5.3 – Хвилі, зроблені користувачем

Користувач натискає кнопку “2D Simul ation” то йому показується вікно с 2д симуляцією середовища, в якому також є елемент взаємодії – кнопка “Rotate” яка перевертає уявне середовище, що вимушує воду рухатись під дією уявної гравітації (рис 5.4, рис 5.5).

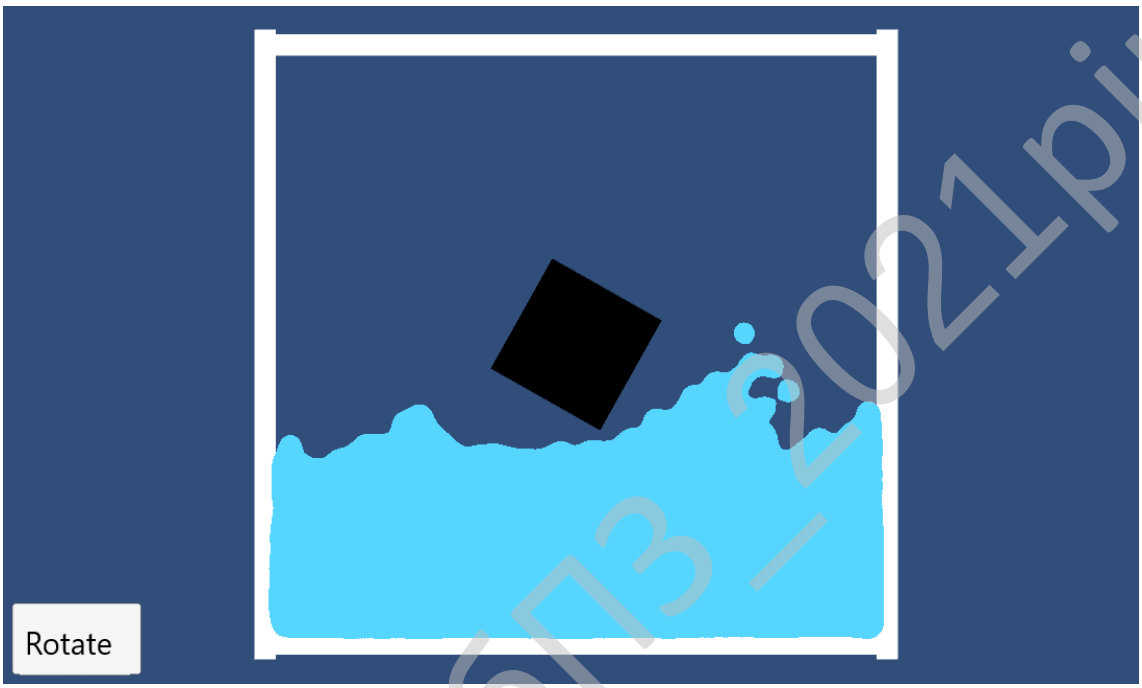

Рисунок 5.4 – Вікно 2D симуляції

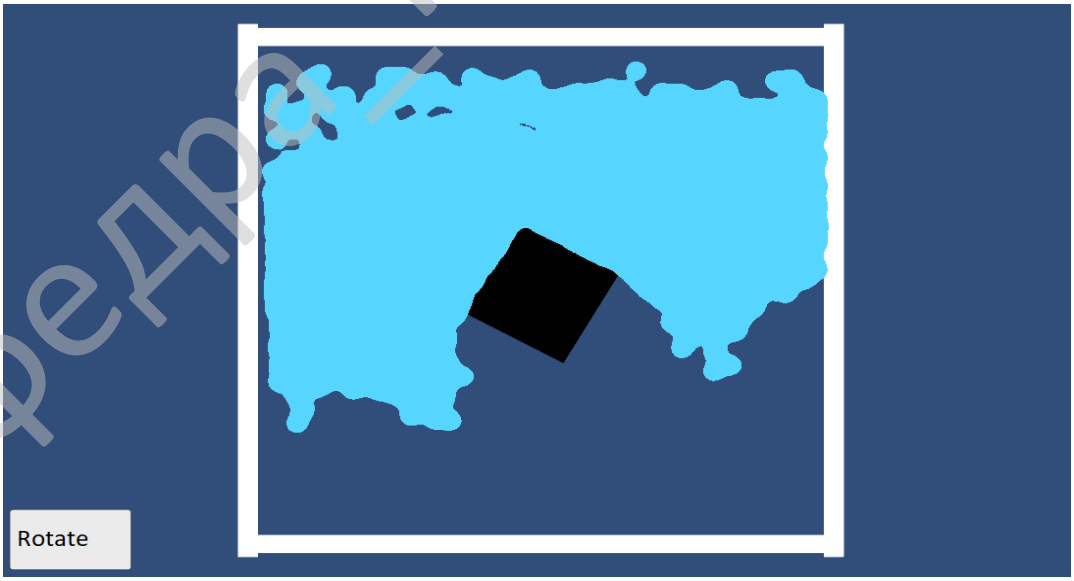

Рис 5.5 – Положення середовища після кількох натискань клавіші “Rotate”

Також є кл авіша “About”, в якій розміщенно інфо про розробника програмного забезпечення (рис 5.6).

Кнопка “Back” повертає вас до головного меню. Остання кнопка – кнопка “Quit” – вихід з ПЗ та закінчення його роботи.

Рисунок 5.6 – Інформація про розробника

6 НАУКОВА НОВИЗНА

Метою роботи є дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY.

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

– Огляд існуючих систем симуляції рідкого середовища на UNITY.

– Дослідження системи симуляції рідкого середовища на UNITY.

– Програмна реалізація системи симуляції рідкого середовища на UNITY.

Об'єктом дослідження є процес симуляції рідкого середовища у комп'ютерних іграх.

Предметом дослідження є методи та алгоритми симуляції рідкого середовища на UNITY.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах комп'ютерної графіки, а також теорії ймовірності.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Удосконалено метод симуляції рідкого середовища для комп'ютерних ігор у середовищі ігрового рушія UNITY.

2. Розроблено вітчизняний продукт симуляції рідкого середовища для комп'ютерних ігор у середовищі ігрового рушія UNITY, який має б ільш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі симуляції рідкого середовища на UNITY.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 24 днів (один місяць).

В магістерській роботі було проведено дослідження та виконана програмна реалізація системи симуляції рідкого середовища на UNITY.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
Кількість розроблених програм період, шт.	N	1
Кількість екземплярів програм, шт.	Ne	300 (2 ост. цифри № зал $*10^2$)
Запланований термін розробки, днів	Fpq	24 (1 місяць)
Група задачі підсистеми управління (1-6)	–	1
Ступінь новизни задачі (А, Б, В, Г)	–	Г
Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
Кількість макетів вхідної інформації	–	8
Кількість форм вихідної інформації.	–	6
Мова програмування (1-6)	–	1
Попередній досвід (1-6)	–	3
Гнучкість проекту ПП (1-6)	–	3
Детальність проекту ПП (1-6)	–	1
Рівень спрацьованості колективу (1-6)	–	2
Ступінь вимірності процесів (1-6)	–	3
Необхідна надійність програмного забезпечення (1-6)	–	3
Розмір бази даних (порівняно з розміром програми) (1-6)	–	4
Складність кінцевого програмного продукту (1-6)	–	5
Необхідний рівень забезпечення повторного використання (1-6)	–	2
Документованість відповідно до планованого життєвого циклу (1-6)	–	3
Вимоги до швидкодії ПП (1-6)	–	3
Обмеження на розміри основного сховища даних (1-6)	–	2
Різноманітність використовуваних обчислювальних платформ (1-6)	–	4
Професійний рівень аналітиків (1-6)	–	3
Професійний рівень програмістів (1-6)	–	4
Постійність складу команди розробників (1-6)	–	2
Досвід розробки додатків (1-6)	–	1
Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
Досвід роботи з програмними інструментами розробки (1-6)	–	3
Розробка ПЗ для декількох серверів одночасно (1-6)	–	3
Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
Вартість ПЗ у розробника (НМА), грн.	–	30000
Норматив додаткової зарплати, % :	$Н_д$	10
Норматив відрахувань у соціальні фонди, %	Нс	22
Норматив загальногосподарських витрат, %	Нг	15
Норматив витрат на освоєння нових мов програмування, %	Нп	15
Рівень рентабельності програмної продукції, %	Ре	40
Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \text{ Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \quad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що в ідображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+4{,}22+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}2^{1{,}027} = 5{,}5 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \quad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 5{,}5 \cdot (1 \cdot 1{,}09 \cdot 1{,}30 \cdot 0{,}91 \cdot 1 \cdot 1 \cdot 1 \cdot 1{,}15 \cdot 1 \cdot 0{,}87 \cdot 1{,}10 \cdot$$
$$\cdot 1{,}22 \cdot 1{,}12 \cdot 1{,}10 \cdot 1 \cdot 1 \cdot 1{,}10) = 12{,}9 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \quad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 3{,}23 \cdot 12{,}9^{0{,}33+0{,}2(1{,}027-1{,}01)} \cdot 83 = 131 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	15	Д7
Робочий проект	131	Ф 7.1-7.4
Впровадження	15	Д13
Всього	180	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{180 \cdot 1}{24 - 3} = 8{,}6 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	385	12	4620	77
Монітор	160	12	1920	32
Клавіатура	140	12	1680	28
Маніпулятор «мишка»	30	12	360	6
Принтер матричний	185	1	185	3
Принтер лазерний	355	2	710	12
Принтер струминний	300	1	300	5
Сканер	155	2	310	5
Концентратор–маршрутизатор	155	2	310	5
Кабельні господарства ЛВС на 1 м. п.	2,5	70	175	3
Кабельне господарство електромережі	48	50	2400	40
Копіювальний апарат	285	1	285	5
Усього за рік:			$З_ч$	221

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_{ч} \cdot n_{тіс}}{1{,}2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{221 \cdot 1}{1{,}2} = 184{,}1 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 184{,}1/(24 \cdot 8) = 1 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	1	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	1	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	1	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	1	
	Контроль взаєморозрахунків з постачальниками	1	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,5
	Створення графічних і стилістичних елементів сайту	1	
	Оформлення банерів і промо-сторінок	1	
	Розміщення графіки і контенту на Інтернет сторінках	1	
Всього		4	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,5
	Верстка друкованих видань	1	
	Додрукова підготовка макетів	1	
	Розміщення графіки і контенту на Інтернет сторінках	1	
Всього		4	

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних

обов'язків. Після визначення чисельності персоналу складається штатний розклад. Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	10500	10500
Продакт-менеджер	0,5	8500	4250
Інженер-програміст	8,6	7500	64500
Інженер-електронщик	1	6000	6000
Інженер-системотехнік	0,5	6000	3000
Адміністратор мережі	0,5	8000	4000
Дизайнер WEB	0,5	10000	5000
Всього за період розробки	$R_{сп}$ = 12,6	-	$Ф_{роб}$ = 97250

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{97250}{12,6 \cdot 24} = 322 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

S_y – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» (м. Кіровоград) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 800…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 25 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 13 \cdot 8 \cdot 20000 = 2080000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 208000 грн. Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$I_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 13 \cdot 3500 = 45500 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Brain за 24.10.21 – джерело http: //brain.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп'ютер		11457
Системний блок		7509
Процесор	Intel Core i7-4790 (4(8) ядра по 3.6 - 4.0 GHz); Cache Memory 8 MB	4200

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Системна плата	1st Player ATX NEW	1525
Відеокарта	PCIeX: ATI HD5670 SAPPHIRE 1024MB/128bit/DDR3/TV/DualDVI	430
Жорсткий диск	HDD: 500 Gb 7200 Serial ATA WD 16MB	490
Оперативна пам'ять	Kingston DDR3 2GB (KVR1333D3N9/2G) Intel/AMD – 2 шт	333
DVD-привод	-	-
Корпус	ATX Middle Tower GIGABYTE GZ-X4 Silver 500W (GZ-X4 Silver)	411
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	120
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37 Photo	2970
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійного живлення	UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS)	1348

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	13	11457	14894,1	163835,1
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2970	297	3267
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	185653,6

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
Будівлі	2080000	-	-
Передавальні пристрої	208000	-	-
Всього по групі	2288000	5	114400

Продовження таблиці 7.8

1	2	3	4
Група 4			
Обчислювальна техніка	185654	-	-
Всього по групі	185654	50	92827
Група 5, 6			
Вимірювальні пристрої	3999	25	-
Транспортні засоби	0	20	-
Господарський інвентар	45500	25	-
Всього по групі	49499	-	12374,75
Нематеріальні активи	30000	10	3000
Разом	$К_р$ = 2553153		$А_р$ = 222602

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_e}, \quad (7.11)$$

де: N_e – кількість екземплярів програм, шт.

$$З_о = 322 \cdot 180/300 = 193{,}2 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 193{,}2 \cdot 10 \cdot 0{,}01 = 19{,}3 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $H_c = 22\%$ від суми основної та додаткової зарплати:

$$C_{оц} = 0{,}01 \cdot H_c (З_о + З_д), \quad (7.13)$$

де: H_c – відрахування на соціальні потреби, %.

$$C_{оц} = 0{,}01 \cdot 22(193{,}2+19{,}3) = 46{,}8 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $H_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot H_г \cdot 0{,}01, \quad (7.14)$$

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 193{,}2 \cdot 15 \cdot 0{,}01 = 29 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм $n = 0{,}33$ приймаємо одну пачку паперу на три місяці розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п = 121$ грн., визначаємо вартість паперу за період розробки $N_м = 1$ міс:

$$З_{М1} = Ц_п \cdot N_м \cdot n. \quad (7.16)$$

$$З_{М1} = 121 \cdot 1 \cdot 0{,}33 = 40 \text{ грн.}$$

Згідно виданих викладачем норм до вартості запам'ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 3 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 3 грн./шт.

$$З_{М2} = 30 \cdot 3 + 3 = 93 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{МЗ} = \sum Ц_{з}., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570 = 1702 \text{ грн.}$$

$$З_{М} = (40 + 93 + 1702)/300 = 6,1 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0,01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 193,2 \cdot 15 \cdot 0,01 = 29 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 300 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 222602 \cdot 1/(300 \cdot 12) = 61,8 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$С_п = 193,2+19,3+46,8+29+6,1+29+61,8 = 385,2 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	193,2
2. Додаткова зарплата виконавців	$З_д$	19,3
3. Відрахування на соціальні потреби	$С_{оц}$	46,8
4. Загальногосподарські витрати	$Г_{осп}$	29
5. Витрати на матеріали	$З_М$	6,1
6. Освоєння нових операційних систем, мов програмування	$О_п$	29
7. Амортизація основних фондів	$А_м$	61,8
8. Повна собівартість програмного забезпечення	$С_п$	385,2
9. Плановий прибуток	$П_р$	154,1
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	539,3
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	107,7
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	647

Визначимо плановий прибуток за рівнем рентабельності ($Р_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 40%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \qquad (7.22)$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 40 \cdot 385{,}2 = 154{,}1 \text{ грн.}$$

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ц іну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	647
Всього капітальних витрат	–	647

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Витрати на профілактичні роботи:

$$З_р = T_р \cdot З_г \cdot (1 + 0{,}01 \cdot H_q) \cdot (1 + 0{,}01 \cdot H_c), \qquad (7.23)$$

де: $T_р$ – кількість годин обслуговування кожного комп’ютера за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	26840	16104
2. Витрати на електроенергію	$З_{ел}$	2268	2205
Витрати на амортизацію	$З_{ам}$	0	323,5
Всього витрат за рік	$І$	29108	18632,5

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 200 годин на рік до 120 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 200 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 26840\ грн,$$

до:

$$З_{р\ нов} = 120 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 16104\ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}15 \cdot 7200 \cdot 2{,}1 = 2268 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}15 \cdot 7000 \cdot 2{,}1 = 2205 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	647	–	323,5
Всього відрахувань	-	–	647	–	323,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \frac{n_{міс}}{12} \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (539,3 - 385,2) \cdot 300 - (0,05 \cdot 2288000 + 0,5 \cdot 185654 +$$
$$+0,25 \cdot 49499 + 0,1 \cdot 30000) \cdot 1/12 = 27679 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е} \frac{n_{міс}}{12}, \quad (7.26)$$

де: $K_р$ – балансова вартість основних фондів розробника.

$$T_в = \frac{2553153}{(539,3 - 385,2) \cdot 300 \cdot 12 / 1} = 4,5 \; року.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (29108\text{-}18632{,}5)\text{-}0{,}5 \cdot 647 = 10152 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{647}{29108 - 18632{,}5} = 0{,}1 \text{ року}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1	2	3
1. Кількість екземплярів програми	Прим.	300
2. Повна собівартість розробленої програми	Грн.	385,2
3. Ціна розробленої програми	Грн.	539,3
4. Плановий прибуток від реалізації розробленої програми	Грн.	154,1
Рентабельність програмної продукції	%	40
Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2553153
7. Загальний прибуток від реалізації програмної продукції	Грн.	46230

Продовження таблиці 7.13

1	2	3
Величина економічного ефекту при виготовлені програмної продукції	Грн.	27679
Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	4,5
).Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	647
.Величина економічного ефекту у користувача програмної продукції	Грн.	10152
.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,1

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

З давніх давен людство приділяє прискіпливу увагу безпеці життя і охорони праці як її складової частини. Умови праці розглядали Арістотель (384-322 до н.е.) і Гіппократ (460-377 до н.е.).

Законом України “Про охорону праці” регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами:

«Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста вулючають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначемо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Програміст працює з електронно-обчислювальною машиною (ЕОМ) та іншим обладнанням, яке є джерелом небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. Так як програміст постійно перебуває в приміщенні, тому для комфортних умов праці в цьому приміщенні необхідно створити належний мікроклімат.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- негативний вплив на органи зору людини;
- ризики ураження електричним струмом;
- недостатня, або надмірна освітленість робочого місця;

- монотонність праці;

- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

- несприятливі мікрокліматичні умови;

- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шуми;

- статичні навантаження на кістково-м’язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 - Розміри приміщення

Найменування	Значення, м
Ширина	3,4
Довжина	3,5
Висота	3

Таблиця 8.2 - Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	5,95
Обсяг, V	$м^3$	не менше 20.0	17,85

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють 2 особи. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [1], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [4] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату

складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 - Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Волого-гість,%	Швидкість повітря, м/с	Температура,°С	Волого-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	40-55	0,1
Тепла	23-25	50-70	0,1	24-25	55-65	0,1

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер HP 1100, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5 – 28 – 2006 р можна в іднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V -го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом

(під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 лк. Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Так як яскраве світло на д ілянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;
- мікроклімат відповідає нормативному значенню;
- акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою

наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення та залулення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

8.5 Розрахункова частина

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 3,4 м, довжина – 3,5 м, висота – 3 м.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = ESKZ / n,$$

де: F - світловий потік, що розраховується, Лм;

E - нормована мінімальна освітленість, Лк; *E* = 300 Лк;

S - площа освітлюваного приміщення (у нашому випадку S=3,4 х 3,5 = 11,9 м^2);

Z - відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = 1,1);

K - коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K* = 1,5);

n - коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і

обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50% [6].

Обчислимо індекс приміщення за формулою:

$$i = S / (h(A + B)),$$

де: S - площа приміщення, S = 11,9 м2;

h - розрахункова висота підвісу, h = 3 м;

A - ширина приміщення, *A* = 3,4 м;

B - довжина приміщення, *B* = 3,5 м.

Підставимо всі значення у формулу та визначемо індекса приміщення: i=0,57.

Знаючи індекс приміщення, за знаходимо n = 0.29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників відповідного типу [6]). Підставимо всі значення у формулу, визначемо світловий потік: F=20312 *Лм*.

Будемо використовувати лампи TL-F 12 36W LED 6000K IP65, світловий потік яких $F_л$ = 3000 Лм.

Число ламп визначається по формулі:

$$N = F / F_л$$

де: F - світловий потік,

$F_л$ - світловий потік одного світильника.

Підставимо всі значення у формулу та визначемо індекса приміщення:

$$N = 20312 / 3000 = 6,77 \text{ шт.}$$

Приймаємо необхідну кількість світильників 7 шт.

8.6 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Розроблене програмне забезпечення призначено для симуляції рідкого середовища у 2D та 3D вимірах.

Воно має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань з його сторони.

Система легко інтегрується в сучасні платформи і взаємодіє з усіма необхідними компонентами.

Програмне забезпечення вийшло з багатим функціоналом, що забезпечує йому широке розповсюдження у різних сферах діяльності. Від навчання і простого прагнення дізнатись як в еде себе рідина в умовах, які неможливо створити в д омашньому середовищі, до розробки комп'ютерних ігор і комп'ютерної графіки для відеоконтенту.

Також програмне забезпечення вийшло невимогливим до потужності ПК, або робочих станцій, на яких воно буде встановлено та буде використовуватись.

Рішення поставленого завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем симуляції рідкого середовища на UNITY.

– Досліджена система симуляції рідкого середовища на UNITY.

– На основі отриманих результатів досліджень створена програмна реалізація системи симуляції рідкого середовища на UNITY.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання симуляції рідкого середовища на UNITY.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функц іональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 7/8/10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм DES.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 10152 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,1 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C#. - М.: Питер, 2018. - 608 с.
2. Торн А. Основы анимации в Unity // издательство ДМК-Пресс, 2016. – 176 с.
3. Агуров, Павел C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.
4. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари, Бен Албахари. - М.: БХВ-Петербург, 2012. - 944 с.
5. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари, Бен Албахари. - М.: БХВ-Петербург, 2013. - 944 с.
6. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.
7. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.
8. Вагнер, Билл C# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 с.
9. Зиборов, В.В. Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2013. - 480 с.
10. Зиборов, Виктор Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2011. - 432 с.
11. Ишкова, Э. А. Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.
12. Касаткин, А. И. Профессиональное программирование на языке си. Управление ресурсами / А.И. Касаткин. - М.: Высшая школа, 2012. - 432 с.
13. Лотка, Рокфорд C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.

14. Мак-Дональд, Мэтью Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.

15. Марченко, А. Л. Основы программирования на C# 2.0 / А.Л. Марченко. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011. - 552 с.

16. Подбельский, В. В. Язык C#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

17. Прайс, Джейсон Visual C# 2.0. Полное руководство / Джейсон Прайс , Майк Гандэрлой. - М.: Век +, Корона-Век, Энтроп, 2010. - 736 с.

18. Рихтер, Джеффри CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# – М.: Питер, 2013. - 928 с.

19. Смоленцев, Н. К. MATLAB. Программирование на Visual C#, Borland JBuilder, VBA (+ CD-ROM) / Н.К. Смоленцев. - М.: ДМК Пресс, 2011. - 456 с.

20. Троелсен, Эндрю Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 с.

21. Троелсен, Эндрю Язык программирования C# 2008 и платформа .NET 3.5 / Эндрю Троелсен. - М.: Вильямс, 2010. - 370 с.

22. Фримен, Адам ASP.NET M VC 3 Framework с примерами на C# для профессионалов / Адам Фримен , Стивен Сандерсон. - М.: Вильямс, 2011. - 672 с.

23. Альфред, В. Ахо Структуры данных и алгоритмы / Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. - М.: Вильямс, 2016. - 400 с.

24. Бёрд, Ричард Жемчужины проектирования алгоритмов. Функциональный подход / Ричард Бёрд. - М.: ДМК Пресс, 2015. - 330 с.

25. Вигерс, Карл Разработка требований к программному обеспечению / Карл Вигерс , Джой Битти. - М.: Русская Редакция, БХВ-Петербург, 2016. - 736 с.

26. Головешкин, В. А. Теория рекурсии для программистов / В.А. Головешкин, М.В. Ульянов. - М.: ФИЗМАТЛИТ, 2006. - 296 с.

27. Зелковиц, М. Принципы разработки программного обеспечения / М. Зелковиц, А. Шоу, Дж. Гэннон. - М.: Мир, 1982. - 364 с.

28. Рассел Дж. Интерфейс программирования приложений / Джесси Рассел. - М.: VSD, 2012. - 811 с.

29. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев. - М.: Форум, 2015. - 352 с.

30. Макарова, Н.В. Основы программирования. учебник с практикумом - М.: КноРус, 2016. - 112 с.

31. Окулов С.М. Основы программирования / С.М. Окулов. - М.: Бином, 2015. - 336 с.

32. Кьоу Дж. Объектно-ориентированное программирование / Дж. Кьоу, М. Джеанини. - М.: Питер, 2015. - 240 с.

33. Хорев, П.Б. Объектно-ориентированное программирование: Учебное пособие / П.Б. Хорев. - М.: Академия, 2018. - 384 с.

Лесневский, А. С. Объектно-ориентированное программирование для начинающих / А.С. Лесневский. - М.: Бином. Лаборатория знаний, 2005. - 232 с

34. Вайсфельд М. Объектно-ориентированное мышление / М. Вайсфельд. - М.: Питер, 2014. - 998 с.

35. Могилев А. Методы программирования. Компьютерные вычисления / А. Могилев, Л. Листрова. - М.: БХВ-Петербург, 2008. - 320 с.

36. Клейнберг, Дж. Алгоритмы. Разработка и применение / Дж. Клейнберг, Е. Тардос. - М.: Питер, 2016. - 800 с.

37. Лафоре, Роберт Структуры данных и алгоритмы в Java / Роберт Лафоре. - М.: Питер, 2016. - 704 с.

38. Магда, Юрий Аппаратное обеспечение и эффективное программирование / Юрий Магда. - М.: Питер, 2007. - 352 с.

39. Макконнелл, Дж. Анализ алгоритмов. Активный обучающий подход / Дж. Макконнелл. - М.: Техносфера, 2009. - 416 с.

40. Робертсон, Л. А. Программирование - это просто. Пошаговый подход / Л.А. Робертсон. - М.: Бином. Лаборатория знаний, 2010. - 384 с.

41. Скиена, Стивен Алгоритмы. Руководство по разработке / Стивен Скиена. - М.: БХВ-Петербург, 2011. - 720 с.

42. Струченков, В. И. Методы оптимизации в прикладных задачах / В.И. Струченков. - М.: Солон-Пресс, 2012. - 167 с.

43. Хант, Э. Программист-прагматик. Путь от подмастерья к мастеру / Э. Хант, Д. Томас. - М.: ЛОРИ, 2013. - 288 с.

44. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 4-е изд. – СПб.: Питер, 2010. – 944 с.

45. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98 - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

46. Закон України «Про охорону праці» в ід 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

47. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

48. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

49. Постанова № 42 в ід 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

50. Центр п іслядипломної освіти та підвищення кваліфікації. - Режим доступу до ресурсу: https://cpo.stu.cn.ua

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування..... 2
2 Підстава для розробки..... 2
3 Мета та призначення розробки..... 2
4 Джерела розробки..... 2
5 Технічні вимоги..... 2
5.1 Вміст проекту..... 2
5.2 Показники призначення..... 3
5.3 Вимоги до функціональних характеристик..... 3
5.4 Вимоги до архітектури..... 3
5.5 Вимоги до надійності..... 3
5.6 Умови експлуатації..... 4
5.7 Вимоги до складу та параметрів технічних засобів..... 4
5.8 Вимоги до інформаційної і програмної сумісності..... 4
5.8.1 Обладнання..... 4
5.8.2 Мова програмування..... 4
5.8.3 Вхідні дані..... 5
5.8.4 Вихідні дані..... 5
6 Вимоги до програмної документації..... 5
7 Економічні вимоги..... 5
8 Вимоги щодо охорони праці..... 5
9 Перелік документів, що розробляються..... 6
10 Етапи розробки..... 6
11 Порядок контролю та приймання..... 6

					ВКРМ-122.21.0003.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Сніховський А.О			Дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Мелешко Є.В.				М	
Н. Контр.		Гермак В.С.				ЦНТУ КН-20М	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи симуляції рідкого середовища на UNITY.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі програмування та захисту інформації (нак. №39-13 від 02.08.2021 року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY.

4 Джерела розробки

Джерелом цієї магістерської 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;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи симуляції рідкого середовища на UNITY;

– 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 7/8/10 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 7/8/10.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Мова 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 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2018 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинна бути розглянуті умови праці програмістів під час розробки програмного забезпечення.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 79 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист 4.11.2021 р.

11.2 Подання магістерської роботи на захист .12.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Є.В. Мелешко

Дослідження та програмна реалізація системи симуляції рідкого середовища на UNITY

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 51

Літера: РП

Кропивницький – 2021 року

```
using UnityEngine;    //Головний із скриптів для симуляцій 3D середовищ

public class WaterSimulation : MonoBehaviour
{
    [SerializeField]
    CustomRenderTexture texture;

    [SerializeField]
    int iterationPerFrame = 5;

    void Start()
    {
        texture.Initialize();
    }

    void Update()
    {
        texture.ClearUpdateZones();
        UpdateZones();
        texture.Update(iterationPerFrame);
    }

    void UpdateZones()
    {
        bool leftClick = Input.GetMouseButton(0);
        bool rightClick = Input.GetMouseButton(1);
        if (!leftClick && !rightClick) return;

        RaycastHit hit;
        var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        if (Physics.Raycast(ray, out hit)) {
            var defaultZone = new CustomRenderTextureUpdateZone();
            defaultZone.needSwap = true;
            defaultZone.passIndex = 0;
            defaultZone.rotation = 0f;
            defaultZone.updateZoneCenter = new Vector2(0.5f, 0.5f);
            defaultZone.updateZoneSize = new Vector2(1f, 1f);

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

```
            texture.SetUpdateZones(new CustomRenderTextureUpdateZone[] {
defaultZone, clickZone });
        }
    }
```

```
// 2Д Симуляція
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LiquidManager : MonoBehaviour {
	public Vector3 direction;
	public Material _material;
	public Camera _liquidCamera;

	void OnRenderImage(RenderTexture src, RenderTexture dest) {
		RenderTexture _renderTexture =
RenderTexture.GetTemporary(Screen.width, Screen.height, 32);
		RenderTexture.active = _renderTexture;
		GL.Clear (false, true, Color.clear);
		RenderTexture.active = null;

		_liquidCamera.targetTexture = _renderTexture;
		_liquidCamera.Render ();
		_liquidCamera.targetTexture = null;

		_material.SetTexture ("_LiquidTex", _renderTexture);

		Graphics.Blit(src, dest, _material);

		RenderTexture.ReleaseTemporary (_renderTexture);
	}

	public void RandomPhysic() {
		direction = Random.insideUnitCircle;

		float len = Physics2D.gravity.magnitude;
		Physics2D.gravity = direction.normalized * len;
	}
}
```

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

public class LiquidGenarate : MonoBehaviour {
      public int _numLiquid = 10;
      public float _width = 1;
      public float _height = 1;
      public GameObject _prefab;

      void Start () {
            for (int i = 0; i < _numLiquid; i++) {
                  GameObject temp = (GameObject)Instantiate (_prefab);
                  float x = Random.Range (-1.0f, 1.0f) * _width / 2;
                  float y = Random.Range (-1.0f, 1.0f) * _height / 2;
                  temp.transform.localPosition = new Vector3 (x, y, 0);
                  temp.transform.parent = transform;
            }
      }

}
using UnityEngine;

public class WaterSimulation : MonoBehaviour
{
    [SerializeField]
    CustomRenderTexture texture;

    [SerializeField]
    int iterationPerFrame = 5;

    void Start()
    {
        texture.Initialize();
    }

    void Update()
    {
        texture.ClearUpdateZones();
        UpdateZones();
        texture.Update(iterationPerFrame);
    }

    void UpdateZones()
    {
        bool leftClick = Input.GetMouseButton(0);
```

```
        bool rightClick = Input.GetMouseButton(1);
        if (!leftClick && !rightClick) return;

        RaycastHit hit;
        var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        if (Physics.Raycast(ray, out hit)) {
            var defaultZone = new CustomRenderTextureUpdateZone();
            defaultZone.needSwap = true;
            defaultZone.passIndex = 0;
            defaultZone.rotation = 0f;
            defaultZone.updateZoneCenter = new Vector2(0.5f, 0.5f);
            defaultZone.updateZoneSize = new Vector2(1f, 1f);

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

            texture.SetUpdateZones(new CustomRenderTextureUpdateZone[] {
defaultZone, clickZone });
        }
    }
}
```

```
using System;
using UnityEngine;
using LostPolygon.DynamicWaterSystem;

namespace LostPolygon.DynamicWaterSystem {
    [AddComponentMenu("Lost Polygon/Dynamic Water System/Advanced Ambient Wave
Solver")]
    public class DynamicWaterSolverAdvancedAmbient :
DynamicWaterSolverSimulation {
        [Serializable]
        public class Wave {
            public float Amplitude = 2;
            public float Angle;
            public float Frequency = 20;
            public float Velocity = 5;
            public int Steepness = 1;
            public bool Circular;
            public Vector2 Position = new Vector2(0.5f, 0.5f);
            public bool Excluded = false;

            [NonSerialized]
            public Vector2 Direction;

            [NonSerialized]
            public Vector2 CircleShift;
        }

        public Wave[] Waves;

        public bool OnlyAmbient = true;

        private float[] _fieldSum;

        private float _time;

        public override void Initialize(Vector2Int gridSize) {
            base.Initialize(gridSize);

            _fieldSum = new float[_grid.x * _grid.y];

            _field = _fieldSum;

            _canInteract = !OnlyAmbient;
        }
```

```
        public override void CreateSplashNormalized(Vector2 center, float
radius, float force) {
            if (OnlyAmbient) {
                return;
            }

            CreateSplashNormalized(center, radius, force, ref FieldSim);
        }

        public override float GetFieldValue(float x, float z) {
            if (x <= 0 || z <= 0 || x >= _grid.x || z >= _grid.y) {
                return float.NegativeInfinity;
            }

            return _fieldSum[(int) z * _grid.x + (int) x];
        }

        public override void StepSimulation(float speed, float damping) {
            if (!OnlyAmbient) {
                base.StepSimulation(speed, damping);
            }

            _canInteract = !OnlyAmbient;
            bool isFieldObstructionNull = _fieldObstruction == null;

            _time += Time.deltaTime;

            foreach (Wave wave in Waves) {
                wave.Direction = new Vector2(FastFunctions.FastCos(wave.Angle *
FastFunctions.Deg2Rad), FastFunctions.FastSin(wave.Angle *
FastFunctions.Deg2Rad));
                wave.CircleShift = new Vector2(-Mathf.Clamp01(wave.Position.x) *
FastFunctions.DoublePi, -Mathf.Clamp01(wave.Position.y) *
FastFunctions.DoublePi);
            }
            Vector2 invGrid = new Vector2(1f / _grid.x, 1f / _grid.y);
            for (int i = 0; i < _grid.x; i++) {
                for (int j = 0; j < _grid.y; j++) {
                    int index = j * _grid.x + i;

                    if (!isFieldObstructionNull && _fieldObstruction[index] ==
byte.MinValue) {
                        continue;
                    }
```

```
                    float obstructionValue = isFieldObstructionNull ? 1f :
_fieldObstruction[index] * FastFunctions.InvertedByteMaxValue;

                    float normX = i * invGrid.x * FastFunctions.DoublePi;
                    float normY = j * invGrid.y * FastFunctions.DoublePi;
                    _fieldSum[index] = 0f;
                    for (int k = 0; k < Waves.Length; k++) {
                        Wave wave = Waves[k];
                        if (wave.Excluded) {
                            continue;
                        }

                        float val;
                        if (wave.Circular) {
                            normX += wave.CircleShift.x;
                            normY += wave.CircleShift.y;

                            val = normX * normX + normY * normY;

                            FastFunctions.FloatIntUnion u;
                            u.i = 0;
                            u.f = val;
                            float xhalf = 0.5f * val;
                            u.i = 0x5f375a86 - (u.i >> 1);
                            u.f = u.f * (1.5f - xhalf * u.f * u.f);

                            val = u.f * val * wave.Frequency + _time *
wave.Velocity;
                        } else {
                            val = (wave.Direction.x * normX + wave.Direction.y *
normY) * wave.Frequency + _time * wave.Velocity;
                        }

                        float tmpVal = (val + Mathf.PI) *
FastFunctions.InvDoublePi;
                        int floor = tmpVal >= 0 ? (int) (tmpVal) : (int)
((tmpVal) - 1);
                        val = val - FastFunctions.DoublePi * floor;

                        if (val < 0) {
                            val = 1.27323954f * val + 0.405284735f * val * val;
                        } else {
                            val = 1.27323954f * val - 0.405284735f * val * val;
                        }

                        tmpVal = (val + 1f) * 0.5f;
```

```
                        int steepness = wave.Steepness;

                        while (steepness > 0) {
                            tmpVal *= val;
                            steepness--;
                        }
                        val = tmpVal * wave.Amplitude * obstructionValue;
                        _fieldSum[index] += val;

                    }

                    if (!OnlyAmbient) {
                        _fieldSum[index] += FieldSimNew[index];
                    }
                }
            }
            _field = _fieldSum;

            _isDirty = true;
        }
    }
#if !UNITY_3_5
}
#endif
```

```
using UnityEngine;
using LostPolygon.DynamicWaterSystem;

#if !UNITY_3_5
namespace LostPolygon.DynamicWaterSystem {
#endif
    [AddComponentMenu("Lost Polygon/Dynamic Water System/Simple Ambient Wave
Solver")]
    public class DynamicWaterSolverAmbientSimple : DynamicWaterSolverSimulation
{
        public float AmbientWaveHeight = 0.3f;
        public float AmbientWaveFrequency = 1f;

        public float AmbientWaveSpeed = 1f;



        public bool OnlyAmbient = true;

        private float[] _fieldSum;
        private float _time;

        public override void Initialize(Vector2Int gridSize) {
            base.Initialize(gridSize);

            _fieldSum = new float[_grid.x * _grid.y];
            _canInteract = !OnlyAmbient;
        }

        public override void StepSimulation(float speed, float damping) {
            if (!OnlyAmbient) {
                base.StepSimulation(speed, damping);
            }

            _canInteract = !OnlyAmbient;

            bool isFieldObstructionNull = _fieldObstruction == null;

            _time += Time.deltaTime * AmbientWaveSpeed;

            Vector2 invGrid = new Vector2(1f / _grid.x, 1f / _grid.y);
            for (int i = 0; i < _grid.x; i++) {
                for (int j = 0; j < _grid.y; j++) {
                    int index = j * _grid.x + i;
                    if (!isFieldObstructionNull && _fieldObstruction[index] ==
byte.MinValue) {
                        continue;
```

```
                    }

                    float obstructionValue = isFieldObstructionNull ? 1f :
_fieldObstruction[index] * FastFunctions.InvertedByteMaxValue;
                    float normX = i * invGrid.x;
                    float normY = j * invGrid.y;

                    float val = (normX + normY) * FastFunctions.DoublePi;
                    if (OnlyAmbient) {
                        _fieldSum[index] = FastFunctions.FastSin(val *
AmbientWaveFrequency + _time) * AmbientWaveHeight * obstructionValue;
                    } else {
                        _fieldSum[index] = FastFunctions.FastSin(val *
AmbientWaveFrequency + _time) * AmbientWaveHeight * obstructionValue +
FieldSimNew[index];
                    }
                }
            }

            _field = _fieldSum;

            _isDirty = true;
        }

        public override void CreateSplashNormalized(Vector2 center, float
radius, float force) {
            if (!OnlyAmbient) {
                CreateSplashNormalized(center, radius, force, ref FieldSim);
            }
        }

        public override float GetFieldValue(float x, float z) {
            if (x <= 0 || z <= 0 || x >= _grid.x || z >= _grid.y) {
                return float.NegativeInfinity;
            }

            return _fieldSum[(int) z * _grid.x + (int) x];
        }
    }
#if !UNITY_3_5
}
#if !UNITY_FLASH
                if (_switchField) {
                    DynamicWaterNativeLibrary.SolveWaveEquationNative(_grid.x,
_grid.y, dt, damping, out max, FieldSim, FieldSimNew, FieldSimSpeed,
_fieldObstruction);
```

```
                    _field = FieldSimNew;
                } else {
                    DynamicWaterNativeLibrary.SolveWaveEquationNative(_grid.x,
_grid.y, dt, damping, out max, FieldSimNew, FieldSim, FieldSimSpeed,
_fieldObstruction);
                    _field = FieldSim;
                }
#endif
            } else {
                if (_switchField) {
                    LinearWaveEqueationSolver.Solve(FieldSim, FieldSimNew,
FieldSimSpeed, _fieldObstruction, _grid, dt, damping, out max);
                    _field = FieldSimNew;
                } else {
                    LinearWaveEqueationSolver.Solve(FieldSimNew, FieldSim,
FieldSimSpeed, _fieldObstruction, _grid, dt, damping, out max);
                    _field = FieldSim;
                }
            }

            _switchField = !_switchField;

            _isDirty = max > DirtyThreshold;
        }

        public override float GetFieldValue(float x, float z) {
            if (x <= 0 || z <= 0 || x >= _grid.x || z >= _grid.y) {
                return float.NegativeInfinity;
            }

            return FieldSim[(int) z * _grid.x + (int) x];
        }

        public override void CreateSplashNormalized(Vector2 center, float
radius, float force) {
            if (!_switchField) {
                CreateSplashNormalized(center, radius, force, ref FieldSimNew);
            } else {
                CreateSplashNormalized(center, radius, force, ref FieldSim);
            }
        }
    }
!UNITY_3_5
}
```

```
using UnityEngine;
using LostPolygon.DynamicWaterSystem;

#if !UNITY_3_5
namespace LostPolygon.DynamicWaterSystem {
#endif

    /// Простий ефект підводного бачення
    [RequireComponent(typeof(WaterDetector))]
    public class UnderwaterFog : MonoBehaviour {

        public FogMode FogMode = FogMode.Linear;
        public Color FogColor = new Color32(112, 183, 255, 255);
        public float FogDensity = 0.1f;
        public float FogStartDistance = 1f;
        public float FogEndDistance = 15f;
        public Material Skybox = null;
        private bool _defaultFog;
        private FogMode _defaultFogMode;
        private Color _defaultFogColor;
        private float _defaultFogDensity;
        private float _defaultFogStartDistance;
        private float _defaultFogEndDistance;
        private Material _defaultSkybox;

        private WaterDetector _waterDetector;
        private Camera _camera;

        private void Start() {

            _camera = gameObject.GetComponent<Camera>();
            if (_camera == null) {
                Debug.LogError(string.Format("The {0} script can only be
attached to the Camera!", GetType()), transform);
                Destroy(this);
                return;
            }


            Rigidbody tempRigidbody = gameObject.GetComponent<Rigidbody>();
            if (tempRigidbody == null) {
                tempRigidbody = gameObject.AddComponent<Rigidbody>();
                tempRigidbody.isKinematic = true;
                tempRigidbody.useGravity = false;
            } else {
```

```
                Debug.LogWarning("Rigidbody component is already attached to
this camera, unexpected behaviour may occur",
                                 transform);
            }

            Collider tempCollider = gameObject.GetComponent<Collider>();
            if (tempCollider == null) {
                tempCollider = gameObject.AddComponent<BoxCollider>();
                tempCollider.isTrigger = true;
                ((BoxCollider) tempCollider).size = Vector3.zero;
            } else {
                Debug.LogWarning("Collider already attached to this camera,
unexpected behaviour may occur", transform);
            }

            _waterDetector = GetComponent<WaterDetector>() ??
gameObject.AddComponent<WaterDetector>();
            _defaultFog = RenderSettings.fog;
            _defaultFogMode = RenderSettings.fogMode;
            _defaultFogColor = RenderSettings.fogColor;
            _defaultFogDensity = RenderSettings.fogDensity;
            _defaultFogStartDistance = RenderSettings.fogStartDistance;
            _defaultFogEndDistance = RenderSettings.fogEndDistance;
            _defaultSkybox = RenderSettings.skybox;
        }

        private void Update() {
            if (_waterDetector != null && _waterDetector.Water != null) {
                float waterLevel =
_waterDetector.GetWaterLevel(transform.position.x, transform.position.y,
transform.position.z);

                bool fogState =
_waterDetector.Water.Collider.bounds.Contains(_camera.transform.position) &&
                                _camera.transform.position.y < waterLevel;
                SetFog(fogState);
            } else {
                SetFog(false);
            }
        }

        private void SetFog(bool enableFog) {
            if (enableFog) {
                RenderSettings.fog = true;
                RenderSettings.fogMode = FogMode;
                RenderSettings.fogColor = FogColor;
```

```
                RenderSettings.fogDensity = FogDensity;
                RenderSettings.fogStartDistance = FogStartDistance;
                RenderSettings.fogEndDistance = FogEndDistance;
                if (Skybox != null) {
                    RenderSettings.skybox = Skybox;
                }
            } else {
                RenderSettings.fog = _defaultFog;
                RenderSettings.fogMode = _defaultFogMode;
                RenderSettings.fogColor = _defaultFogColor;
                RenderSettings.fogDensity = _defaultFogDensity;
                RenderSettings.fogStartDistance = _defaultFogStartDistance;
                RenderSettings.fogEndDistance = _defaultFogEndDistance;
                RenderSettings.skybox = _defaultSkybox;
            }
        }
    }
#if !UNITY_3_5
}
```

```
using LostPolygon.DynamicWaterSystem;
using UnityEngine;


namespace LostPolygon.DynamicWaterSystem {
    [RequireComponent(typeof(Collider))]
    public class WaterDetector : MonoBehaviour, IDynamicWaterFieldState {
        private IDynamicWaterFluidVolume _water;

        public IDynamicWaterFluidVolume Water {
            get {
                return _water;
            }
        }
        public float GetWaterLevel(Vector3 position) {
            return _water != null ? _water.GetWaterLevel(position) :
float.NegativeInfinity;
        }

        public float GetWaterLevel(float x, float y, float z) {
            return _water != null ? _water.GetWaterLevel(x, y, z) :
float.NegativeInfinity;
        }


        public float GetWaterLevel(float x, float z) {
            return _water != null ? _water.GetWaterLevel(x, z) :
float.NegativeInfinity;
        }

        private void OnTriggerEnter(Collider otherCollider) {
            if (otherCollider.CompareTag(FluidVolume.DynamicWaterTagName)) {
                _water = otherCollider.gameObject.GetComponent<DynamicWater>();
            }
        }

        private void OnTriggerExit(Collider otherCollider) {
            if (_water != null &&
otherCollider.CompareTag(FluidVolume.DynamicWaterTagName) &&
                otherCollider == _water.Collider) {
                _water = null;
            }
        }
    }
}
```

```
using UnityEngine;
using System.Collections.Generic;
using LostPolygon.DynamicWaterSystem;

#if !UNITY_3_5
namespace LostPolygon.DynamicWaterSystem {
#endif

    [AddComponentMenu("Lost Polygon/Dynamic Water System/Buoyancy Force")]
    [RequireComponent(typeof(Rigidbody))]
    public class BuoyancyForce : MonoBehaviour {
        public int Resolution {
            get {
                return _resolution;
            }
            set {
                int clamped = Mathf.Clamp(value, 1, 15);
                if (_resolution != clamped) {
                    _resolution = clamped;

                    if (Application.isPlaying) {
                        RecalculateVoxels();
                    }
                }
            }
        }


        public float Density {
            get {
                return _density;
            }
            set {
                value = Mathf.Clamp(value, 0.1f, float.PositiveInfinity);

                if (_density != value) {
                    _density = value;

                    if (_calculateMassFromDensity && _volume > 0.01f) {
                        _rigidbody.mass = _volume * _density;
                    }

                    RecalculateCache();
                }
            }
        }
```

```
        public float DragInFluid {
            get {
                return _dragInFluid;
            }
            set {
                _dragInFluid = Mathf.Clamp(value, 0f, float.PositiveInfinity);
            }
        }

        public float AngularDragInFluid {
            get {
                return _angularDragInFluid;
            }
            set {
                _angularDragInFluid = Mathf.Clamp(value, 0f,
float.PositiveInfinity);
            }
        }

        public float SplashForceFactor {
            get {
                return _splashForceFactor;
            }
            set {
                _splashForceFactor = Mathf.Clamp(value, 0f, 50f);
            }
        }

        public float MaxSplashForce {
            get {
                return _maxSplashForce;
            }
            set {
                _maxSplashForce = Mathf.Clamp(value, 0f, 50f);
            }
        }

        public bool ProcessChildren {
            get {
                return _processChildren;
            }
            set {
                _processChildren = value;

                if (_processChildren != value) {
```

```
            _processChildren = value;

            if (Application.isPlaying) {
                RecalculateVoxels();
            }

            RecalculateCache();
        }
    }
}

public bool CalculateMassFromDensity {
    get {
        return _calculateMassFromDensity;
    }
    set {
        if (_calculateMassFromDensity != value) {
            _calculateMassFromDensity = value;
            RecalculateCache();
        }
    }
}

public float SubmergedVolume {
    get {
        return _subMergedVolume;
    }
}

[SerializeField]
private int _resolution = 2;

[SerializeField]
private float _density = 750f;

[SerializeField]
private float _dragInFluid = 1f;

[SerializeField]
private float _angularDragInFluid = 1f;

[SerializeField]
private float _splashForceFactor = 2.5f;

[SerializeField]
private float _maxSplashForce = 2f;
```

```
[SerializeField]
private bool _calculateMassFromDensity = false;

[SerializeField]
private bool _processChildren;

private Transform _transform;
private Collider[] _colliders;
private Rigidbody _rigidbody;
private float _voxelSize;
private BuoyancyVoxel[] _buoyancyVoxels;
private Bounds _bounds;
private Vector3Int _voxelResolution;
private Vector3 _voxelArchimedesForce;
private IDynamicWaterFluidVolume _water;
private float _archimedesForceFactor;
private float _volume;
private float _dragNonFluid;
private float _angularDragNonFluid;
private float _subMergedVolume;
private float _subMergedVolumePrev;
private bool _isReady;

private struct BuoyancyVoxel {
    public Vector3 Position;
    public bool HadPassedWater;
    public bool IsOnColliderEdge;
}

private RecompiledMarker _recompileMarker;
private int _voxelsLength;
private float _maxSplashForceNormalized;
private float _splashForceFactorNormalized;
private float _forceNormalizeFactor;
private float _archimedesForceMagnitude;
private readonly Vector3 _upDirection = Vector3.up; // Force direction

private void Start() {
    _isReady = false;

    _recompileMarker = new RecompiledMarker();

    _transform = GetComponent<Transform>();
    _rigidbody = GetComponent<Rigidbody>();
```

```
            RecalculateVoxels();
            RecalculateCache();
        }

        private void RecalculateCache() {
            if (_water == null || !_isReady) {
                return;
            }

            _archimedesForceMagnitude = -gravityDirection.magnitude *
_archimedesForceFactor;
            _forceNormalizeFactor = _water.Density * _rigidbody.mass *
Time.deltaTime;

            _voxelArchimedesForce = -_upDirection * (_archimedesForceMagnitude *
_forceNormalizeFactor / _buoyancyVoxels.Length);


        private void RecalculateVoxels() {
            _isReady = false;


            if (_processChildren) {
                _colliders = GetComponentsInChildren<Collider>();
            } else {
                _colliders = GetComponent<Collider>() != null ? new[]
{GetComponent<Collider>()} : new Collider[] {};
            }

            if (_colliders.Length == 0) {
                Debug.LogError(
                    _processChildren ? "No colliders are attached to the object
and to the children, buoyancy disabled"
                        : "No colliders are attached to the object, buoyancy
disabled." +
                          " Enabled \"Process children\" if you have colliders
attached to the children",
                    this);

                Destroy(this);
                return;
            }
            _bounds = new Bounds();
            foreach (Collider col in _colliders)
                _bounds.Encapsulate(col.bounds);
```

```
            if (_bounds.extents.magnitude < 0.001f) {
                Debug.LogError("Collider bounds are zero-sized, buoyancy
disabled", this);

                Destroy(this);
                return;
            }


            _voxelSize = _bounds.size.magnitude / _resolution / 2f;

            _voxelResolution.x = Mathf.RoundToInt(_bounds.size.x / _voxelSize) +
1;
            _voxelResolution.y = Mathf.RoundToInt(_bounds.size.y / _voxelSize) +
1;
            _voxelResolution.z = Mathf.RoundToInt(_bounds.size.z / _voxelSize) +
1;

            _buoyancyVoxels = SliceIntoVoxels().ToArray();
            if (_buoyancyVoxels.Length == 0) {
                Debug.LogWarning("No buoyancy voxels were generated. Try to
increase the Resolution parameter", this);
            }


            _transform.rotation = originalRotation;
            _transform.position = originalPosition;

            _voxelSize = Mathf.Pow(
                _bounds.size.x * _bounds.size.y * _bounds.size.z /
(_voxelResolution.x * _voxelResolution.y * _voxelResolution.z),
                1f / 3f
                );


            _volume = _buoyancyVoxels.Length * _voxelSize * _voxelSize *
_voxelSize;

            if (_calculateMassFromDensity) {
                _rigidbody.mass = _volume * _density;
            }

            _isReady = true;
        }
```

```
        private List<BuoyancyVoxel> SliceIntoVoxels() {
            List<BuoyancyVoxel> voxelList = new
List<BuoyancyVoxel>(_voxelResolution.x * _voxelResolution.y *
_voxelResolution.z);

            for (int ix = 0; ix < _voxelResolution.x; ix++) {
                for (int iy = 0; iy < _voxelResolution.y; iy++) {
                    for (int iz = 0; iz < _voxelResolution.z; iz++) {
                        float x = _bounds.min.x + _bounds.size.x /
_voxelResolution.x * (0.5f + ix);
                        float y = _bounds.min.y + _bounds.size.y /
_voxelResolution.y * (0.5f + iy);
                        float z = _bounds.min.z + _bounds.size.z /
_voxelResolution.z * (0.5f + iz);

                        Vector3 point = new Vector3(x, y, z);
                        for (int k = 0; k < _colliders.Length; k++) {
                            if
(ColliderTools.IsPointInsideCollider(_colliders[k], point)) {
                                BuoyancyVoxel buoyancyVoxel;
                                buoyancyVoxel.Position =
_transform.InverseTransformPoint(point);
                                buoyancyVoxel.IsOnColliderEdge =
ColliderTools.IsPointAtColliderEdge(_colliders[k], point, _voxelSize);
                                buoyancyVoxel.HadPassedWater = true;

                                voxelList.Add(buoyancyVoxel);

                                break;
                            }
                        }
                    }
                }
            }

            return voxelList;
        }
        private void OnTriggerEnter(Collider otherCollider) {
            if (!_isReady) {
                return;
            }

            if (otherCollider.CompareTag(FluidVolume.DynamicWaterTagName)) {
                IDynamicWaterFluidVolume water =
otherCollider.gameObject.GetComponent<FluidVolume>() ??
```

```
otherCollider.gameObject.GetComponent<DynamicWater>();
                if (water != null) {
                    _water = water;

                    RecalculateCache();
                }
            }
        }
        private void OnTriggerStay(Collider otherCollider) {
            if (!_isReady) {
                return;
            }

            if (_water == null ||
(otherCollider.CompareTag(FluidVolume.DynamicWaterTagName) && otherCollider !=
_water.Collider)) {
                IDynamicWaterFluidVolume water =
otherCollider.gameObject.GetComponent<FluidVolume>() ??

otherCollider.gameObject.GetComponent<DynamicWater>();
                if (water != null) {
                    _water = water;

                    RecalculateCache();
                }
            }
        }
        private void OnTriggerExit(Collider otherCollider) {
            if (!_isReady) {
                return;
            }

            if (_water != null &&
otherCollider.CompareTag(FluidVolume.DynamicWaterTagName) && otherCollider ==
_water.Collider) {
                _water = null;
            }
        }

        private void FixedUpdate() {
            // Happens on assembly reload
            if (_recompileMarker == null) {
                RecalculateVoxels();
                _water = null;
                _recompileMarker = new RecompiledMarker();
```

```
            }

            // Failsafe
            if (_water == null || _density < 0.01f || !_isReady) {
                _rigidbody.drag = _dragNonFluid;
                _rigidbody.angularDrag = _angularDragNonFluid;
                return;
            }

            float invDoubleVoxelSize = 1f / (2f * _voxelSize);
            DynamicWater dynamicWater = _water as DynamicWater;
            bool solverCanInteract = dynamicWater != null &&
dynamicWater.Solver.CanInteract;

            for (int i = 0; i < _voxelsLength; i++) {
                Vector3 wp =
_transform.TransformPoint(_buoyancyVoxels[i].Position);
                float waterLevel = _water.GetWaterLevel(wp.x, wp.y, wp.z);
                if (waterLevel != float.NegativeInfinity && (wp.y - _voxelSize /
1f < waterLevel)) {
                    Vector3 velocity = _rigidbody.GetPointVelocity(wp);
                    float k = (waterLevel - wp.y) * invDoubleVoxelSize + 0.5f;
                    if (_buoyancyVoxels[i].IsOnColliderEdge) {
                        if (!_buoyancyVoxels[i].HadPassedWater && (k < 1f && k >
0f)) {

                            if (solverCanInteract) {
                                float force =
FastFunctions.FastVector3Magnitude(velocity) * _splashForceFactorNormalized;
                                if (force > _maxSplashForceNormalized) {
                                    force = _maxSplashForceNormalized;
                                }

                                if (force > 0.0075f) {
                                    _water.CreateSplash(wp, _voxelSize, force);
                                }
                            }

                            _buoyancyVoxels[i].HadPassedWater = true;
                        } else {
                            _buoyancyVoxels[i].HadPassedWater = false;
                        }
                    }

                    k = (k > 1f) ? 1f : (k < 0f) ? 0f : k;
                    _subMergedVolume += k;
```

```
                    Vector3 archimedesForce;
                    archimedesForce.x = k * _voxelArchimedesForce.x;
                    archimedesForce.y = k * _voxelArchimedesForce.y;
                    archimedesForce.z = k * _voxelArchimedesForce.z;

                    // Applying the local force
                    _rigidbody.AddForceAtPosition(archimedesForce, wp,
ForceMode.Impulse);
                }
            }

            // Normalizing the submerged volume
            // 0 - object is fully outside the water
            // 1 - object is fully submerged
            _subMergedVolume /= _voxelsLength;

            const float threshold = 0.01f;

            // Sending the message to other components
            if (_subMergedVolumePrev < threshold && _subMergedVolume >=
threshold) {
                SendMessage("OnFluidVolumeEnter", _water,
SendMessageOptions.DontRequireReceiver);
            } else if (_subMergedVolumePrev >= threshold && _subMergedVolume <
threshold) {
                SendMessage("OnFluidVolumeExit", _water,
SendMessageOptions.DontRequireReceiver);
            }

            _subMergedVolumePrev = _subMergedVolume;

            // Calculating the drag
            _rigidbody.drag = Mathf.Lerp(_rigidbody.drag, _subMergedVolume >
0.0001f ? _dragNonFluid + _dragInFluid : _dragNonFluid, 8f * Time.deltaTime);
            _rigidbody.angularDrag = Mathf.Lerp(_rigidbody.angularDrag,
_subMergedVolume > 0.0001f ? _angularDragNonFluid + _angularDragInFluid :
_angularDragNonFluid, 8f * Time.deltaTime);
        }

        private void OnDrawGizmos() {
            Gizmos.DrawIcon(transform.position,
"DynamicWater/BuoyancyForce.png");
            if (!Application.isEditor || _buoyancyVoxels == null) {
                return;
            }
```

```
            Vector3 gizmoSize = Vector3.one * _voxelSize;
            Gizmos.color = new Color(Color.yellow.r, Color.yellow.g,
Color.yellow.b, 0.5f);

            foreach (var p in _buoyancyVoxels) {
                Gizmos.DrawCube(transform.TransformPoint(p.Position),
gizmoSize);
            }

            Gizmos.color = Color.red;
            Gizmos.DrawSphere(GetComponent<Rigidbody>().worldCenterOfMass,
_voxelSize / 2f);
        }
    }
#if !UNITY_3_5
}
#endif

#define ARBITRARY_ROTATIONS
```

```
using System;
using System.Collections.Generic;
using LostPolygon.DynamicWaterSystem;
using UnityEngine;

#if !UNITY_3_5
namespace LostPolygon.DynamicWaterSystem {
#endif
    [AddComponentMenu("Lost Polygon/Dynamic Water System/Dynamic Water")]
    [RequireComponent(typeof(MeshFilter))]
    [RequireComponent(typeof(MeshRenderer))]
    [RequireComponent(typeof(BoxCollider))]
    public class DynamicWater : FluidVolume, IDynamicWaterSettings,
IDynamicWaterMeshSettings {
        /// <summary>
        /// The tag and layer name used by clickable plane collider.
        /// </summary>
        public const string PlaneColliderLayerName =
"DynamicWaterPlaneCollider";
        public const string DynamicWaterObstructionTag =
"DynamicWaterObstruction";

        public const string DynamicWaterObstructionInvertedTag =
"DynamicWaterObstructionInverted";

        public bool UsePlaneCollider = true;

        public bool UpdateWhenNotVisible = false;

        public int Quality {
            get {
                return _quality;
            }
            set {
                if (_quality == value) {
                    return;
                }

                _quality = Mathf.Clamp(value, 4, 256);
                _resolution = Mathf.Clamp(Mathf.RoundToInt(_quality *
MaxResolution() / 256f), 4, MaxResolution());
                if (_resolution % 2 == 1) {
                    _resolution++;
                }

                PropertyChanged();
```

```
    }
}

public Vector2Int GridSize {
    get {
        return _grid;
    }
}

public float NodeSize {
    get {
        return _nodeSize;
    }
}


public bool CalculateNormals {
    get {
        return _calculateNormals;
    }
    set {
        _calculateNormals = value;
    }
}
public bool UseFastNormals {
    get {
        return _useFastNormals;
    }
    set {
        _useFastNormals = value;
    }
}

public bool SetTangents {
    get {
        return _setTangents;
    }
    set {
        _setTangents = value;
    }
}


public bool UseObstructions {
    get {
        return _useObstructions;
```

```
    }
    set {
        _useObstructions = value;
    }
}

public bool MeshBakeObstructionData {
    get {
        return _meshBakeObstructionData;
    }
    set {
        _meshBakeObstructionData = value;
        if (_meshBakeObstructionData == value) {
            return;
        }

        _meshBakeObstructionData = value;
        PropertyChanged();
    }
}
```

```
        /// <summary>
        /// Gets a value indicating whether the obstruction field data will
eroded (expanded or shrinked).
        /// </summary>
        /// <remarks>
        /// This can be used for dealing with edge artifacts that can occur when
using obstruction data.

        public int ObstructionDataErosion {
            get {
                return _obstructionDataErosion;
            }
            set {
                if (_obstructionDataErosion == value) {
                    return;
                }

                _obstructionDataErosion = value;
                PropertyChanged();
            }
        }
        public float Speed {
            get {
                return _speed;
            }
            set {
                _speed = Mathf.Clamp(value, 0f, MaxSpeed());
            }
        }


        public float Damping {
            get {
                return 1f - _damping;
            }
            set {
                _damping = 1f - Mathf.Clamp01(value);
            }
        }

        public DynamicWaterSolver Solver {
            get {
                return _waterSolver;
            }
        }
```

```
        public virtual BoxCollider PlaneCollider {
            get {
                return _planeCollider;
            }
        }
        public Texture2D ObstructionMask {
            get {
                return _obstructionMask;
            }
            set {
                if (_obstructionMask != value) {
                    _obstructionMask = value;
                    PropertyChanged();
                }
            }
        }

        private MeshFilter _meshFilter;
        private MeshRenderer _meshRenderer;
        private DynamicWaterSolver _waterSolver;
        private bool _prevIsDirty = true;
        private DynamicWaterMesh _waterMesh;
        private DynamicWaterMesh _waterMeshSimple;
        private Vector2 _nodeSizeNormalized;
        private BoxCollider _planeCollider;
        private RecompiledMarker _recompiledMarker;

#if !ARBITRARY_ROTATIONS
    private Vector3 _cachedPosition;
    private Vector3 _cachedLossyScale;
    #endif

        /* Property fields */

        [SerializeField]
        private float _damping = 0.97f;

        [SerializeField]
        private float _speed = 20f;

        [SerializeField]
        private bool _calculateNormals = true;

        [SerializeField]
        private bool _useFastNormals = true;
```

```
        [SerializeField]
        private bool _useObstructions;

        [SerializeField]
        private Texture2D _obstructionMask;

        [SerializeField]
        private bool _setTangents;

        [SerializeField]
        private int _quality = 96;

        [SerializeField]
        private bool _meshBakeObstructionData;

        [SerializeField]
        private int _obstructionDataErosion;

        private int _resolution = 64;
        private Vector2Int _grid;
        private float _nodeSize;

        public struct ObstructionInfo {
            public GameObject GameObject;
            public Collider Collider;
            public ColliderTools.ColliderSpecificTestEnum SpecificTest;

            public ObstructionInfo(GameObject go) {
                GameObject = go;
                Collider = go.GetComponent<Collider>();
                SpecificTest = ColliderTools.ColliderSpecificTestEnum.None;
                if (go.GetComponent<MarkObstructionAsTerrain>() != null) {
                    SpecificTest =
ColliderTools.ColliderSpecificTestEnum.Terrain;
                }
            }
        }
        public override void CreateSplash(Vector3 center, float radius, float
force) {
            if (_waterSolver == null || !_waterSolver.CanInteract ||
!_collider.bounds.Contains(center)) {
                return;
            }

            center = _transform.InverseTransformPoint(center);
```

```
            center = new Vector2(center.x * _nodeSizeNormalized.x, center.z *
_nodeSizeNormalized.y);
            CreateSplashNormalized(center, radius, force);
        }

        public void CreateSplash(Vector3 start, Vector3 end, float radius, float
force) {
            if (_waterSolver == null || !_waterSolver.CanInteract ||
!(_collider.bounds.Contains(start) || _collider.bounds.Contains(end))) {
                return;
            }

            start = _transform.InverseTransformPoint(start);
            end = _transform.InverseTransformPoint(end);

            start = new Vector2(start.x * _nodeSizeNormalized.x, start.z *
_nodeSizeNormalized.y);
            end = new Vector2(end.x * _nodeSizeNormalized.x, end.z *
_nodeSizeNormalized.y);

            CreateSplashLine(new Vector2Int(start), new Vector2Int(end), radius,
force);
        }

        public override float GetWaterLevel(float x, float y, float z) {
            if (_waterSolver == null) {
                return 0f;
            }

#        if ARBITRARY_ROTATIONS
            Vector3 projected;
            Vector3 projectedNormalized;
            projected.x = x;
            projected.y = y;
            projected.z = z;
            projected = _transform.InverseTransformPoint(projected);

            projectedNormalized.x = projected.x * _nodeSizeNormalized.x;
            projectedNormalized.z = projected.z * _nodeSizeNormalized.y;
            projectedNormalized.y =
_waterSolver.GetFieldValue(projectedNormalized.x, projectedNormalized.z);

            projected.y = projectedNormalized.y;
            projected = transform.TransformPoint(projected);

            return projected.y;
```

```
#        else
            Vector3 projected;
            projected.x = (x - _cachedPosition.x * _cachedLossyScale.x) *
_nodeSizeNormalized.x;
            projected.z = (z - _cachedPosition.z * _cachedLossyScale.z) *
_nodeSizeNormalized.y;
            projected.y = _waterSolver.GetFieldValue(projected.x, projected.z);

            if (projected.y != float.NegativeInfinity) {
                projected.y = _cachedPosition.y + _cachedLossyScale.y *
projected.y;
            }

            return projected.y;
#        endif
        }
        public override float GetWaterLevel(Vector3 position) {
            return GetWaterLevel(position.x, position.y, position.z);
        }
        public int MaxResolution() {
            const int maxSide = 65535;

            float maxResolution = Mathf.Sqrt(maxSide * Mathf.Max(_size.x,
_size.y) / Mathf.Min(_size.x, _size.y));
            int maxResolutionInt = Mathf.FloorToInt(maxResolution) - 1;

            return maxResolutionInt;
        }
        public float MaxSpeed() {
            Vector2Int grid = SizeToGridResolution(Size, _resolution);
            float speedCoeff = (grid.x + grid.y) / 128f;
            float maxSpeed = LinearWaveEqueationSolver.MaxDt /
(Time.fixedDeltaTime * speedCoeff);

            return maxSpeed;
        }
        public void RecalculateObstructions() {
            // Initializing the obstruction array
            byte[] fieldObstruction = new byte[_grid.x * _grid.y];
            for (int i = 0; i < fieldObstruction.Length; i++) {
                fieldObstruction[i] = 255;
            }

            bool obstructionTextureReadable = true;
```

```
            Vector2 invGrid = new Vector2(1f / _grid.x * _collider.size.x, 1f /
_grid.y * _collider.size.z);
            Vector2 invSize = new Vector2(1f / _size.x, 1f / _size.y);
            GameObject[] obstructionsGO =
GetGameObjectsWithTagInBounds(DynamicWaterObstructionTag, _collider.bounds);
            ObstructionInfo[] obstructions =
GetObstructionInfoArray(obstructionsGO);
            GameObject[] obstructionsInvertedGO =
GetGameObjectsWithTagInBounds(DynamicWaterObstructionInvertedTag,
_collider.bounds);
            ObstructionInfo[] obstructionsInverted =
GetObstructionInfoArray(obstructionsInvertedGO);

            for (int j = 0; j < _grid.y; j++) {
                for (int i = 0; i < _grid.x; i++) {
                    int index = j * _grid.x + i;

                    float normX = i * invGrid.x;
                    float normZ = j * invGrid.y;

                    float normX1 = normX * invSize.x;
                    float normZ1 = normZ * invSize.y;

                    // Getting the obstruction value from obstruction mask
                    byte obstructionMaskValue = byte.MaxValue;
                    if (obstructionTextureReadable) {
                        try {
                            if (_obstructionMask != null) {
                                obstructionMaskValue =
                                    (byte)
Mathf.RoundToInt(_obstructionMask.GetPixelBilinear(normX1, normZ1).r * 255f);
                                fieldObstruction[index] = obstructionMaskValue;
                            }
                            if (obstructionMaskValue == byte.MinValue) {
                                continue;
                            }
                        } catch (Exception e) {
                            obstructionTextureReadable = false;

                            Debug.LogError(e, this);
                        }
                    }
                    for (int k = 0; k < obstructions.Length; k++) {
                        ObstructionInfo obstruction = obstructions[k];

                        if (obstruction.Collider != null) {
```

```
                            Vector3 point = _transform.TransformPoint(normX, 0f,
normZ);
                            if (obstruction.Collider) {
                                if (obstruction.Collider.bounds.Contains(point)
&&
ColliderTools.IsPointInsideCollider(obstruction.Collider, point,
obstruction.SpecificTest)) {
                                    fieldObstruction[index] = byte.MinValue;
                                    goto REPEAT;
                                }
                            }
                        }
                    }

                    for (int k = 0; k < obstructionsInverted.Length; k++) {
                        ObstructionInfo obstruction = obstructionsInverted[k];

                        if (obstruction.Collider != null) {
                            Vector3 point = _transform.TransformPoint(normX, 0f,
normZ);
                            if (obstruction.Collider) {
                                if (
!(obstruction.Collider.bounds.Contains(point) &&
ColliderTools.IsPointInsideCollider(obstruction.Collider, point,
obstruction.SpecificTest))) {
                                    fieldObstruction[index] = byte.MinValue;
                                    goto REPEAT;
                                }
                            }
                        }
                    }


                    REPEAT:
                    ;
                }
            }

            fieldObstruction = ErodeObstructionField(_obstructionDataErosion,
fieldObstruction);
```

```
            _waterSolver.FieldObstruction = fieldObstruction;
        }

        public byte[] ErodeObstructionField(int iterations, byte[] field) {
            if (iterations == 0) {
                return field;
            }
            bool erodeInwards = iterations > 0;
            iterations = iterations < 0 ? -iterations : iterations;

            byte[] field2 = new byte[field.Length];
            Array.Copy(field, field2, field.Length);

            byte[] fieldRead = field;
            byte[] fieldWrite = field2;

            for (int n = 0; n < iterations; n++) {
                for (int j = 0; j < _grid.y; j++) {
                    for (int i = 0; i < _grid.x; i++) {
                        if (i <= 0 || j <= 0 || i >= _grid.x - 1 || j >= _grid.y
- 1) {
                            continue;
                        }

                        int index = j * _grid.x + i;
                        bool isSolid = fieldRead[index] == byte.MinValue;

                        if (erodeInwards) {

                            if (!isSolid) {
                                continue;
                            }

                            int solidNeighbourCount = (fieldRead[index - 1] ==
byte.MinValue ? 1 : 0) +
                                                      (fieldRead[index + 1] ==
byte.MinValue ? 1 : 0) +
                                                      (fieldRead[index +
_grid.x] == byte.MinValue ? 1 : 0) +
                                                      (fieldRead[index -
_grid.x] == byte.MinValue ? 1 : 0);

                            int nonSolidNeighboursCount = (fieldRead[index - 1]
!= byte.MinValue ? 1 : 0) +
                                                          (fieldRead[index + 1]
!= byte.MinValue ? 1 : 0) +
```

```
                                                    (fieldRead[index +
_grid.x] != byte.MinValue ? 1 : 0) +
                                                    (fieldRead[index -
_grid.x] != byte.MinValue ? 1 : 0);

                            int nonSolidNeighboursSum = ((int) fieldRead[index -
1] != (int) byte.MinValue ? (int) fieldRead[index - 1] : 0) +
                                                        ((int) fieldRead[index +
1] != (int) byte.MinValue ? (int) fieldRead[index + 1] : 0) +
                                                        ((int) fieldRead[index +
_grid.x] != (int) byte.MinValue ? (int) fieldRead[index + _grid.x] : 0) +
                                                        ((int) fieldRead[index -
_grid.x] != (int) byte.MinValue ? (int) fieldRead[index - _grid.x] : 0);

                            if (solidNeighbourCount != 4) {
                                fieldWrite[index] = (byte)
(nonSolidNeighboursSum / nonSolidNeighboursCount);
                            }
                        } else {

                            if (isSolid) {
                                continue;
                            }

                            int solidNeighbourCount = (fieldRead[index - 1] ==
byte.MinValue ? 1 : 0) +
                                                      (fieldRead[index + 1] ==
byte.MinValue ? 1 : 0) +
                                                      (fieldRead[index +
_grid.x] == byte.MinValue ? 1 : 0) +
                                                      (fieldRead[index -
_grid.x] == byte.MinValue ? 1 : 0);

                            if (solidNeighbourCount >= 1 && solidNeighbourCount
!= 4) {
                                fieldWrite[index] = byte.MinValue;

                                byte val = (byte) (fieldRead[index] / (4 -
solidNeighbourCount) * 3);

                                if (fieldRead[index - 1] != byte.MinValue) {
                                    fieldRead[index - 1] += val;
                                }
                                if (fieldRead[index + 1] != byte.MinValue) {
                                    fieldRead[index + 1] += val;
                                }
```

```
                                if (fieldRead[index - _grid.x] != byte.MinValue)
{
                                    fieldRead[index - _grid.x] += val;
                                }
                                if (fieldRead[index + _grid.x] != byte.MinValue)
{
                                    fieldRead[index + _grid.x] += val;
                                }
                            }
                        }
                    }
                }

                Array.Copy(fieldWrite, fieldRead, field.Length);

                if (n >= iterations - 1) {
                    continue;
                }

                fieldRead = fieldRead == field2 ? field : field2;
                fieldWrite = fieldWrite == field2 ? field : field2;
            }

            return fieldWrite;
        }
        public ObstructionInfo[] GetObstructionInfoArray(GameObject[]
gameObjects) {
            ObstructionInfo[] result = new ObstructionInfo[gameObjects.Length];
            for (int i = 0; i < gameObjects.Length; i++) {
                result[i] = new ObstructionInfo(gameObjects[i]);
            }

            return result;
        }

        protected GameObject[] GetGameObjectsWithTagInBounds(string searchTag,
Bounds bounds) {
            GameObject[] obstructionsTemp =
GameObject.FindGameObjectsWithTag(searchTag);
            List<GameObject> obstructionList = new
List<GameObject>(obstructionsTemp.Length);
            foreach (GameObject obstruction in obstructionsTemp) {
                Collider obstructionCollider = GetComponent<Collider>();
                if (obstructionCollider != null &&
bounds.Intersects(obstructionCollider.bounds)) {
                    obstructionList.Add(obstruction);
```

```
                }
            }

            return obstructionList.ToArray();
        }

        protected override void UpdateCollider() {
            if (_collider == null) {
                return;
            }

            // We don't want the collider to obstruct view in Editor mode
            if (Application.isPlaying) {
                _collider.center = new Vector3(_size.x / 2f, 0f, _size.y / 2f);
                _collider.size = new Vector3(_size.x, _depth * 2f, _size.y);
            } else {
                _collider.center = new Vector3(_size.x / 2f, -_depth / 2f,
_size.y / 2f);
                _collider.size = new Vector3(_size.x, _depth - 0.1f, _size.y);
            }

            _collider.isTrigger = true;
        }

        protected void CreatePlaneCollider() {
            GameObject planeColliderObject = null;

            // Trying to find an existing collider
            foreach (Transform child in transform) {
                if (child.CompareTag(PlaneColliderLayerName)) {
                    planeColliderObject = child.gameObject;
                    break;
                }
            }

            // If we haven't found any - creating a new one
            if (planeColliderObject == null) {
                planeColliderObject = new GameObject(PlaneColliderLayerName);
                planeColliderObject.tag = PlaneColliderLayerName;
                planeColliderObject.layer =
LayerMask.NameToLayer(PlaneColliderLayerName);
                planeColliderObject.transform.parent = _transform;
                planeColliderObject.transform.rotation = transform.rotation;
            }

            // Attaching and setting up the BoxCollider
```

```
            BoxCollider bc = planeColliderObject.GetComponent<BoxCollider>();
            if (bc == null) {
                bc = planeColliderObject.AddComponent<BoxCollider>();
            }

            bc.size = new Vector3(_collider.size.x, 0f, _collider.size.z);
            bc.center = _collider.center;
            bc.isTrigger = true;

            _planeCollider = bc;

            planeColliderObject.transform.localPosition = Vector3.zero;
        }

        private void Start() {
            _recompiledMarker = new RecompiledMarker();
            Initialize();
        }

        private void FixedUpdate() {
            if (!Application.isPlaying) {
                return;
            }

            if (_recompiledMarker == null) {
                Initialize();
                _recompiledMarker = new RecompiledMarker();
            }

            if (Application.isEditor) {
                UpdateComponents();
            }

#if !ARBITRARY_ROTATIONS
        _cachedPosition = _transform.position;
        _cachedLossyScale = _transform.lossyScale;
        #endif

            StepSimulation();
        }

        private void OnDestroy() {
            ClearMeshes();
        }

        private void StepSimulation() {
```

```
            if (_waterSolver == null || (!_meshRenderer.isVisible &&
!UpdateWhenNotVisible)) {
                return;
            }

            _waterSolver.StepSimulation(_speed, _damping);

            if (_waterSolver.IsDirty) {
                _waterMesh.IsDirty = _waterSolver.IsDirty;
                if (_meshRenderer.isVisible) {
                    float[] field = _waterSolver.Field;
                    byte[] fieldObstruction = _waterSolver.FieldObstruction;
                    _waterMesh.UpdateMesh(field, fieldObstruction);
                }
            }


            if (!_meshBakeObstructionData && (_prevIsDirty !=
_waterSolver.IsDirty)) {
                _meshFilter.mesh = _waterSolver.IsDirty ? _waterMesh.Mesh :
_waterMeshSimple.Mesh;
            }

            _prevIsDirty = _waterSolver.IsDirty;
        }

        private void UpdateComponents() {
            _meshFilter = gameObject.GetComponent<MeshFilter>();
            _meshRenderer = gameObject.GetComponent<MeshRenderer>();
        }

        private void UpdateProperties() {

            _resolution = Mathf.RoundToInt(_quality * MaxResolution() / 256f);
            GridMath.CalculateGrid(_resolution, Size, out _grid, out _nodeSize);

            if (Application.isPlaying) {
                _waterSolver = null;

                DynamicWaterSolver[] solverComponents =
GetComponents<DynamicWaterSolver>();
                foreach (DynamicWaterSolver solverComponent in solverComponents)
{
                    if (_waterSolver == null) {
                        _waterSolver = solverComponent;
                    } else {
```

```
                    Destroy(solverComponent);
                }
            }

            if (solverComponents.Length > 1) {
                Debug.LogWarning(
                    "More than one DynamicWaterSolver component present.
Using the first one, others are destroyed",
                    this);
            }


                _waterSolver =
gameObject.AddComponent<DynamicWaterSolverSimulation>();
            }

            _waterSolver.Initialize(_grid);


            if (_useObstructions) {
                RecalculateObstructions();
            }
        }


        ClearMeshes();


        const int simpleMeshResolution = 6;
        _waterMeshSimple = new DynamicWaterMesh(simpleMeshResolution, Size,
this);

        _waterMesh = new DynamicWaterMesh(_resolution, Size, this,
_waterSolver != null ? _waterSolver.FieldObstruction : null);

        _meshFilter = gameObject.GetComponent<MeshFilter>();
        _meshFilter.mesh = _waterMesh.Mesh;


        _nodeSizeNormalized.x = 1f / Size.x * _grid.x;
        _nodeSizeNormalized.y = 1f / Size.y * _grid.y;


        UpdateCollider();
```

```
            if (Application.isPlaying && UsePlaneCollider) {
                CreatePlaneCollider();
            }

            _prevIsDirty = true;
        }


        private Vector2Int SizeToGridResolution(Vector2 size, int resolution) {
            float nodeSize = Mathf.Max(size.x, size.y) / resolution;
            Vector2Int grid;
            grid.x = Mathf.RoundToInt(size.x / nodeSize) + 1;
            grid.y = Mathf.RoundToInt(size.y / nodeSize) + 1;

            return grid;
        }
        private void ClearMeshes() {
            if (_meshFilter != null) {
                DestroyImmediate(_meshFilter.sharedMesh);
            }

            if (_waterMeshSimple != null) {
                _waterMeshSimple.FreeMesh();
                _waterMeshSimple = null;
            }

            if (_waterMesh != null) {
                _waterMesh.FreeMesh();
                _waterMesh = null;
            }
        }

        private void CreateSplashNormalized(Vector2 center, float radius, float
force) {
            radius = _size.x < _size.y ? radius / _size.x * _grid.x : radius /
_size.y * _grid.y;

            _waterSolver.CreateSplashNormalized(center, radius, force);
        }

        private void CreateSplashLine(Vector2Int start, Vector2Int end, float
radius, float force) {
            int dx = Math.Abs(end.x - start.x);
            int dy = Math.Abs(end.y - start.y);

            int sx, sy;
```

```
            if (start.x < end.x) {
                sx = 1;
            } else {
                sx = -1;
            }
            if (start.y < end.y) {
                sy = 1;
            } else {
                sy = -1;
            }

            int err = dx - dy;
            bool splashMade = false;
            while (true) {
                if (start.x == end.x && start.y == end.y) {
                    break;
                }

                Vector2 startf;
                startf.x = start.x;
                startf.y = start.y;
                CreateSplashNormalized(startf, radius, force);

                splashMade = true;

                int e2 = 2 * err;

                if (e2 > -dy) {
                    err = err - dy;
                    start.x = start.x + sx;
                }

                if (e2 < dx) {
                    err = err + dx;
                    start.y = start.y + sy;
                }
            }
            if (!splashMade) {
                CreateSplashNormalized(new Vector2(start.x, start.y), radius,
force);
            }
        }

        private void OnDrawGizmos() {
            if (!Application.isEditor) {
```

```
                return;
            }

            Collider collider = GetComponent<Collider>();
            Gizmos.DrawIcon(new Vector3(collider.bounds.center.x,
transform.position.y + 0.1f, collider.bounds.center.z),
"DynamicWater/FluidVolume.png");
            Gizmos.color = new Color(0f, 0f, 1f, 0.1f);
            if (_collider != null) {
                Gizmos.matrix = transform.localToWorldMatrix;
                Gizmos.DrawCube(new Vector3(Size.x / 2f, -Depth / 2f, Size.y /
2f), new Vector3(Size.x, Depth, Size.y));
            }
        }
    }
#if !UNITY_3_5
}
```

```
using System.Collections;
using UnityEngine;
using LostPolygon.DynamicWaterSystem;

#if !UNITY_3_5
namespace LostPolygon.DynamicWaterSystem {
#endif
    [AddComponentMenu("Lost Polygon/Dynamic Water System/Splash Zone")]
    [RequireComponent(typeof(BoxCollider))]
    public class SplashZone : MonoBehaviour {
        public DynamicWater Water;

        public float DropsPerSecond = 10f;
        public float RadiusMin = 0.1f;
        public float RadiusMax = 0.2f;
        public float ForceMin = 0.3f;
        public float ForceMax = 0.8f;
        public bool AutoStart = true;

        private bool _isRaining;
        private BoxCollider _collider;

        public bool IsRaining {
            get {
                return _isRaining;
            }

            set {
                if (_isRaining != value) {
                    _isRaining = value;

                    if (_isRaining) {
                        StartRain();
                    } else {
                        StopRain();
                    }
                }
            }
        }

        private void StartRain() {
            if (Water == null) {
                return;
            }

            StopRain();
```

```
            StartCoroutine("DoMakeSplash");
            _isRaining = true;
        }

        private void StopRain() {
            if (Water == null) {
                return;
            }

            StopCoroutine("DoMakeSplash");
            _isRaining = false;
        }

        private void Start() {
            if (Water == null) {
                Debug.LogError("Water field is not set, SplashZone disabled",
this);
                enabled = false;
                return;
            }

            _collider = GetComponent<BoxCollider>();
            _collider.isTrigger = true;

            if (AutoStart) {
                StartRain();
            }
        }

        private IEnumerator DoMakeSplash() {
            while (true) {
                if (Water == null) {
                    break;
                }
                Vector3 point = new Vector3(
                    Random.Range(-0.5f, 0.5f),
                    0.5f,
                    Random.Range(-0.5f, 0.5f));
                point = transform.TransformPoint(point);
                point.y = Water.transform.position.y;

                Water.CreateSplash(point, Random.Range(RadiusMin, RadiusMax),
Random.Range(ForceMin, ForceMax));
```

```
                yield return new WaitForSeconds(1f / Mathf.Clamp(DropsPerSecond,
0f, 100f));
            }
        }

        private void OnDrawGizmos() {
            if (!Application.isEditor) {
                return;
            }

            Gizmos.DrawIcon(transform.position, "DynamicWater/SplashZone.png");

            Gizmos.color = new Color(1f, 0f, 0f, 0.2f);
            if (GetComponent<Collider>() != null) {
                Gizmos.matrix = transform.localToWorldMatrix;
                Gizmos.DrawCube(Vector3.zero, Vector3.one);
            }
        }
    }
#if !UNITY_3_5
}
```