АНОТАЦІЯ

Магістерська робота містить 97 сторінок, 8 таблиць, 9 рисунків, список використаної література з 22 найменувань, 3 додатка.

Актуальність теми полягає в тому, що в сучасному світі індустрія комп'ютерних ігор розвивається пропорційно тому, як збільшується обчислювальна сила комп'ютерів. З кожним роком ігри стають більш схожими на реальний світ. Розвивається графічна, фізична складова. Та по справжньому живими ігри може зробити лише достатньо потужний штучний інтелект, який зможе взаємодіяти з гравцем, середовищем та іншими об'єктами, якими керує штучний інтелект.

Об'єктом дослідження роботи є середовище, в якому будуть існувати різні одиниці штучного інтелекту. Одиниця штучного інтелекту — це окремий об'єкт в віртуальному середовищі, яким керує окрема одиниця штучного інтелекту. Дослідження такого середовища, де буде перебування, взаємодія, виживання таких одиниць.

Метою розробки є розробка такого віртуального середовища, в якому зможе бути значна кількість таких одиниць з штучним інтелектом, які будуть взаємодіяти, розвиватися, виживати в цьому віртуальному середовищі.

При розробці було використано MVC (модель-вид-контролер) архітектуру, робота написана за допомогою мови програмування Java та технологій LWJGL та LibGDX.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, ВІРТУАЛЬНЕ СЕРЕДОВИЩЕ, JAVA, LIBGDX, LWJGL, MVC.

ABSTRACT

Thesis contains 97 pages, 8 tables, 9 images, list of used literature with 22 titles, 3 applications.

The relevance of the topic is that in today's world, the computer gaming industry is evolving in proportion to how the computing power of computers is increasing. Every year, the games become more like the real world. The graphic, physical component develops. But truly live games can only be made by a sufficiently powerful artificial intelligence that can interact with the player, the environment and other objects controlled by artificial intelligence.

The object of the study is the environment in which there will be different units of artificial intelligence. Artificial Intelligence Unit is a separate object in a virtual environment managed by a separate artificial intelligence unit. Investigation of the environment where there will be stay, interaction, survival of such units.

The purpose of the development is to develop such a virtual environment, in which there can be a significant number of such units with artificial intelligence that will interact, develop, survive in this virtual environment.

The design used the MVC (model-viev-controller type) architecture, the work was written using the Java programming language and LWJGL and LibGDX technologies.

Keywords: ARTIFICIAL INTELLIGENCE, VIRTUAL ENVIRONMENT, JAVA, LIBGDX, LWJGL, MVC.

ЗМІСТ

ВСТУП

На саму можливість говорити про поняття «Штучний інтелект» величезний вплив справила поява механістичного матеріалізму, який починається з роботи Рене Декарта «Міркування про метод" (1637) і відразу за цим роботи Томаса Гоббса «Людська природа». Рене Декарт припустив, що тварина – це якийсь складний механізм, тим самим сформувавши механістичну теорію.

Технологічними передумовами до виникнення науки ШІ стало те, що в 1623 році Вільгельм Шикард побудував першу механічну цифрову обчислювальну машину, за якою пішли машини Блеза Паскаля і Лейбніца. Лейбніц також був першим, хто описав сучасну двійкову систему числення, хоча до нього цією системою періодично захоплювалися багато великих вчених.

У XVIII столітті завдяки розвитку техніки і, в особливості, годинникових механізмів інтерес до подібних винаходів зріс ще сильніше. В середині 1750-х років австрійський винахідник Фрідріх фон Кнаус сконструював серію машин, які вміли писати пером досить довгі тексти.

Досягнення в механіці XIX століття сприяли новому поштовху винаходів в напрямку до сучасного розуміння штучного інтелекту. У 1830-х роках англійський математик Чарльз Беббідж придумав концепцію складного цифрового калькулятора – аналітичної машини, яка могла б розраховувати ходи для гри в шахи. У 1832 році С. Н. Корсаков представив принцип розробки наукових методів і пристроїв для посилення можливостей розуму і запропонував серію «інтелектуальних машин», в конструкції яких, вперше в історії інформатики, застосував перфоровані карти.

Стрімкий розвиток обчислювальної сили комп'ютера дає змогу розробляти більш корисніше і краще програмне забезпечення, одно із напрямів таких галузей — це ігрова індустрія. З кожним роком ігри стають більш

реалістичними, вони стають кращими зі сторони комп'ютерної графіки та фізики. Штучний інтелект також одна із сторін ігрової індустрії.

Розвиток напрямку штучного інтелекту дасть іграм більшого занурення в ігровий світ, зробить світ більше живим, подібним до реального світу.

З самих перших днів зародження ігрової індустрії, починаючи з таких ігор, як Pong і Pac-Man, засоби штучного інтелекту стали невід'ємною частиною практично любої комп'ютерної гри. Звісно зараз розробники не приділяють стільки часу на створення засобів штучного інтелекту, но без них все ще неможливо обійтись при створенні розважальних ігор. А в зв'язку тим, що сьогодні інші аспекти розважальних ігор (такі як графіка, фізика) досягли дивовижної степені розвитку, увага розробників все з більшою і більшою степеню концентрується на розробці штучного інтелекту[19].

В нашому світі іноді таких створінь, які наділені штучним інтелектом називають аніматорами. Аніматор — це автономні створіння зі штучним тілом, які знаходяться в віртуальному світі. Перед програмістами, які розробляють засоби штучного інтелекту, стоїть задача наділити аніматорів унікальними навичками і можливостями, які дозволяють взаємодіяти їм зі своїм віртуальним середовищем.

Отже, за мету було поставлено розробити віртуальне середовище, в якому буде відбуватися взаємодія тварин, наділених штучним інтелектом, що поможе розробити та вдосконалити певні методики створення штучного інтелекту, який буде застосовуватися в створенні комп'ютерних ігор або в симуляціях з використанням штучного інтелекту.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

Предметна область — область знань, впливу чи діяльності, яка підлягає вивченню з метою розробки програмного забезпечення, яке покращить хід речей в цій області.

Проблемна область — це певна сукупність задач і питань, котра вивчається дослідниками.

У цій сфері виділяють 6 проблем:

1) Представлення знань;

2) Маніпулювання знаннями;

Для того, щоб знаннями можна було користуватись для рішення задач, система штучного інтелекту повинна:

- оперувати знаннями;
- поповнювати знанням;
- класифікувати знаннями;
- звертатись до знань;
- формувати на основі цих знань абстрактні поняття;
- здійснювати правдоподібний вивід інформації;
- користуватися моделями міркувань, котрі орієнтують людський розум.

3) Проблема спілкування:

- розуміння тексту;
- синтез і розуміння мовлення;
- розробка процедур когнітивної графіки.

4) Проблема навчання;

5) Проблема поведінки.

У якості предметної області розглядається задача розробки штучного інтелекту. В наслідок цього завдання полягає у розробці алгоритмів дій, які штучний інтелект буде виконувати при певних обставинах.

1.1.2 Постанова задачі

Після аналізу предметної області, визначення основних функцій і проблем вирішено розробити подобу до живої екосистеми, яка буде містити віртуальних тварин підконтрольними штучним інтелектом, які будуть існувати і взаємодіяти у віртуальному середовищі, і буде мати такі можливості:

- Можливість створити нову екосистему;
- Можливість вказати кількість і різновидність тварин під час створення екосистеми;
- Можливість переміщати камеру в екосистемі;
- Можливість видалення тварини з екосистеми;
- Можливість добавити тварину в екосистему;
- Можливість побачити інформацію, параметри і характеристики тварини;
- Можливість побачити наявну кількість тварин в екосистемі;
- Можливість вивести графік кількості різних видів тварин в екосистемі.

Дана система буде працювати на операційних системах сімейства Windows, Linux та MacOS.

1.1.3 Пошук акторів та варіантів використання

Для моделювання системи найважливішим аспектом є опис її динамічної поведінки, тобто опис поведінки компонентів системи в будь-який довільний змінний момент часу.

В рамках мови UML всі уявлення про модель складної системи фіксуються у вигляді спеціальних графічних конструкцій, що отримали назву діаграм [1]. У UML всього існує п'ять діаграм, що дозволяють моделювати

відповідний характер, і однією із них є діаграма варіантів використання або діаграма прецедентів (use case diagram) – ця діаграма є найбільш загальною концептуальною моделлю складної системи, яка описує функціональне призначення системи.

Знаючи, що діаграма варіантів використання має динамічний характер, для здійснення взаємодії необхідна наявність певних зовнішніх факторів – актори. Діаграма прецедентів складається з акторів (actor), варіантів використання (use case) та їхніх відношень (relationship). При цьому актором може бути людина, технічний пристрій, програма (штучний інтелект) чи будь-яка інша система, яка може служити джерелом впливу на модельовану систему так, як визначить буде визначено розробником. У свою чергу, варіант використання служить для опису сервісів, які система надає акторові. Іншими словами, кожен варіант використання визначає певний набір дій, який чинять системою при діалозі з актором [2].

Дана діаграма застосовується для моделювання системи або підсистеми програми, тому для моделювання цілісної системи часто використовують декілька діграм, які фіксують певну функціональність системи.

Метою побудови діаграми прецедентів є:

- Загальний аналіз вимог до системи;
- Високорівневе моделювання системи;
- Визначення зовнішніх та внутрішніх чинників, що впливають на систему;
- Показати взаємодію між вимогами та акторами.

Перед тим, як побудувати діаграму варіантів використання, слід визначити наступні елементи:

- Функціональні можливості, що мають бути представлені у варіантах використання;
- Актори;
- Відноження між варіантами використання та акторами.

Окремий варіант використання позначається на діаграмі еліпсом, всередині якого (або під ним) міститься його коротка назву або ім'я у формі дієслова з пояснювальними словами.

Мета варіантів використання полягає в тому, щоб визначити закінчений аспект або фрагмент поведінки деякої сутності без розкриття внутрішньої структури цієї сутності. В якості такої сутності може виступати вихідна система або будь-який інший елемент моделі, який володіє власною поведінкою, подібно підсистемі або класу в моделі системи [3].

Актор – це роль, що виконується сутностями (людина, штучний інтелект тощо), які взаємодіють з системою і представляє собою будь-яку зовнішню по відношенню до модельованої системи сутність, що взаємодіє з системою і використовує її функціональні можливості для досягнення певних цілей або вирішення приватних завдань. При цьому актори служать для позначення узгодженості безлічі ролей, які можуть грати користувачі в процесі взаємодії з проектованою системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

- асоціації;
- включення;
- розширення;
- узагальнення.

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме — за допомогою відношень включення, розширення і узагальнення [4].

Для даної системи симулювання віртуальної системи головним актором є користувач. Його основні функції можна зобразити з допомогою діаграми варіантів використання. Конкретна система розробляється для вільного користування. Згідно вимог, це додаток для настільного комп'ютера в якому буде

відбуватися симуляція подоби екосистеми. У даній системі не планується взаємодія між користувачами, тому достатньо припустити, що система не має складних архітектурних рішень в плані багатокористувацьких можливостей. Зокрема система призначена для не достатньо кваліфікованих користувачів і повинна бути достатньо зрозумілою для таких користувачів які зможуть самостійно оволодіти інтерфейсом конкретного додатку. Другим актором буде істота, яка буде керуватися з допомогою штучного інтелекту, вона буде взаємодіяти з віртуальним навколишнім середовищем, і іншими істотами. Тобто в роботі системи будуть присутні лише двоє акторів: Користувач, якому буде доступна частина функціональності, яка необхідна для комфортного спостереження за віртуальним середовищем, та істоти, котрі керуються штучним інтелектом та існують в цьому середовищі.

1.1.4 Опис варіантів використання

Як було зазначено у попередньому підрозділі, в системі існують двоє акторів — Користувач і Істота. Користувач — це користувач системи, який має доступ до інструментів впливу на симульовану віртуальну екосистему та має можливість спостерігати за процесами які відбуваються в ній. Істота — це актор системи який буде керуватися за допомогою штучного інтелекту і взаємодіяти безпосередньо з віртуальним середовищем і іншими істотами в системі.

До варіантів використання Користувача відносяться: створення екосистеми, налаштування екосистеми, переміщення камери в екосистемі, додавання істоти в екосистему, видалення істоти з екосистеми, перегляд параметрів екосистеми, перегляд параметрів істоти з екосистеми.

До варіантів використання Істоти, яка буде контролюватися з допомогою штучного інтелекту будуть доступні наступні варіанти використання: перегляд області в екосистемі, переміщення по екосистемі, з'їсти їжу в радіусі досяжності,

випити рідину з водойми, ріст, розмножитися з іншою Істотою свого виду, вбити істоту іншого виду, з'їсти істоту іншого виду, померти.

Для того, щоб детальніше ознайомитись з функціональними можливостями акторів системи було створено діаграму варіантів використання, з описом ключових варіантів використання системи.

На рисунку 1.1 зображено діаграму варіантів цих акторів.

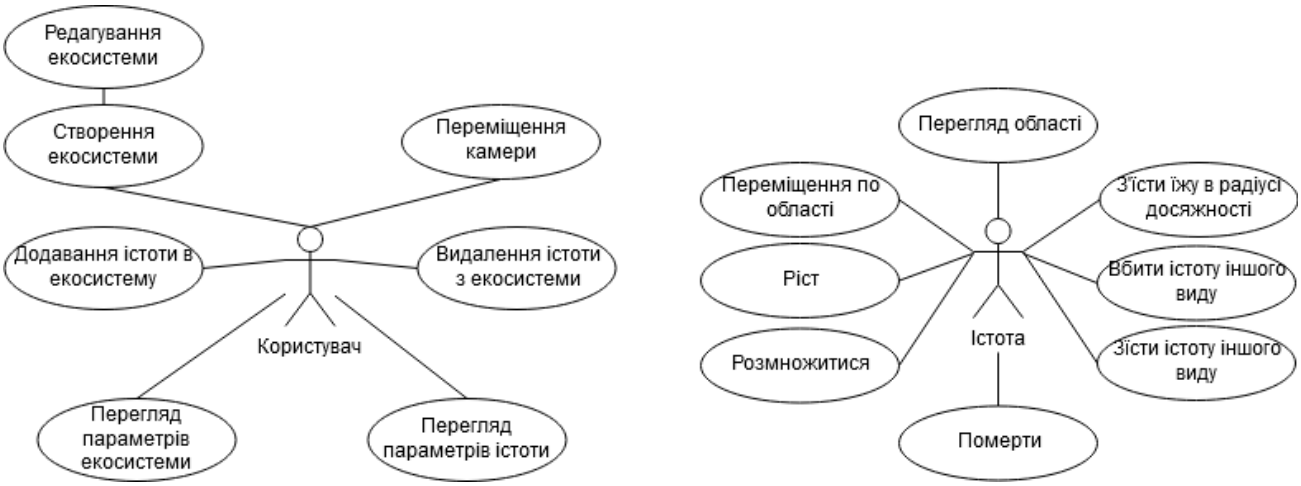


Рисунок 1.1 — Діаграма варіантів використання системи

З описом ключових варіантів використання системи можна ознайомитись в таблиці 1.1.

Кожен з акторів має унікальні варіанти використання, які не доступні іншому актору цієї системи.

Таблиця 1.1 — Опис ключових варіантів використання

Код	Найменування	Основний сценарій	Альтернативний сценарій
ВК1	Створити екосистему	1. Користувач натискаючи кнопку “Створити екосистему” переходить у відповідне вікно, де користувачеві пропонують створити екосистему з налаштуваннями по замовчуванню. 2. Користувач натискає кнопку “Створити нову екосистему”, після чого створюється екосистема з заданими параметрами, а користувач переходить до перегляду екосистеми.	2.а Користувач натискає кнопку “Задати параметри екосистеми” після чого виконується варіант використання “Налаштування системи”.
ВК2	Налаштування екосистеми	1. Під час перебування у вікні “Створення екосистеми” можна нажати кнопку “Налаштування системи”. 2. Система запропонує ввести налаштування екосистеми, такі як розмір, кількість звірів, різновидність звірів, особливість ландшафту і тд.	
ВК3	Переміщення камери	1. Користувач натискає відведені кнопки які сприяють переміщенню камери. 2. Камера переміщується у відповідному напрямку в залежності від натиснутої кнопки.	2.а Якщо камера дійшла до границі де закінчується світ — нічого не відбувається.

Продовження таблиці 1.1

Код	Найменування	Основний сценарій	Альтернативний сценарій
ВК4	Додавання істоти в екосистему	1. Користувач натискає відповідний елемент інтерфейсу. 2. З боку появляться відповідні елементи інтерфейсу які дозволяють вибрати вид істоти. 3. Користувач вибирає істоту і клацає курсором на вільній комірці світу, в якій і появляться вибраний вид істоти.	3.а Користувач вибирає зайняту або непрохідну комірку світу, після чого нічого не відбувається.
ВК5	Видалення істоти з екосистеми	1. Користувач з допомогою камери находить бажану істоту. 2. Користувач натискає відповідну кнопку на інтерфейсі. 3. Користувач натискає мишкою на бажаній істоті, після чого вона зникає зі світу.	
ВК6	Перегляд параметрів екосистеми	1. Користувач натискає на кнопку меню. 2. В меню з'являється список кількості істот і видів істот. Відображається графік видів істот, в якому зображується кількість істот з початку запуску симуляції екосистеми до теперішнього часу.	

Продовження таблиці 1.1

Код	Найменування	Основний сценарій	Альтернативний сценарій
ВК7	Перегляд параметрів істоти	1. Користувач наводить курсором на бажану істоту. 2. Натискаючи курсором по істоті з'являється список усіх характеристик істоти, здоров'я, вік, стать, голод, спрага, вид, покоління, стан штучного інтелекту і тп.	
ВІ1	Перегляд області	1. Зі світу в передається масив даних, в якому містяться об'єкти в видимій для штучного інтелекту області. 2. Штучний інтелект вирішує які подальше він буде виконувати дії.	
ВІ2	Переміщення по області	1. Істота, яка підконтрольна штучним інтелектом вибирає вільну, прохідну клітинку світу, котра розташована зверху, знизу, чи з боку від свого розташування. 2. Здійснюється переміщення Істоти в віртуальному просторі екосистеми.	
ВІ3	З'їсти їжу в радіусі досяжності	1. Після перегляду області і виявлення їжі в радіусі досягнення Істоти — Істота вирішує поглинати їжу чи ні. 2. Якщо Істота вирішила поглинути їжу, одиниця їжі зникає з простору симульованої екосистеми.	3.а Нарахування балів ситності істоти не відбувається, якщо кількість балів перейшла максимально дозволений поріг.

Продовження таблиці 1.1

Код	Найменування	Основний сценарій	Альтернативний сценарій
		3. Після поглинання їжі, Істоті нараховуються бали ситності.	
BI4	Ріст	1. Після проходження певного проміжку часу — Істоті нараховуються бали росту. 2. Якщо у Істота досягла певної кількості балів — вона стає дорослою і в неї з'являються додаткові можливості.	2.а Якщо у Істоти накопичилось надто багато балів росту, тоді Істота випадкову перевірку на смерть, після успішності перевірки — виконується варіант використання “Померти”.
BI5	Розмножитися	1. Після проходження певного періоду часу, істоті яка стала дорослою нараховуються “бали розмноження”. 2. Після нарахування певної кількості балів — Істота має змогу розмножитись з іншою істотою цього виду, яка попала в радіус перегляду області. 3. Після успішного виконання умов, система сама використовує варіант використання “Додавання істоти в екосистему”.	
BI6	Вбити істоту іншого виду	1. Істота, вид якої має характеристику “Хижак” може знищити істоту іншого виду в радіусі досягнення, вибравши Істоту, “Хижак” наносить певну кількість шкоди вибраній Істоті, після чого вона втрачає певну	

Продовження таблиці 1.1

Код	Найменування	Основний сценарій	Альтернативний сценарій
		кількість одиниць здоров'я, еквівалентну кількості нанесеної шкоди. 2. Після декількох спроб вбити Істоту, в неї закінчуються одиниці здоров'я, і вона виконує варіант використання “Померти”.	
BI7	З'їсти іншу істоту	1. Істота, вид якої має характеристику “Хижак” може з'їсти недавно знищену мертву істоту в радіусі дії. В процесі цього виконується варіант використання “З'їсти їжу в радіусі досягнення”. 2. Після першого кроку, мертва Істота пропадає зі середовища симульованої екосистеми, а Істоті “Хижак” нараховуються бали ситності.	1.a Нарахування балів ситності істоти не відбувається, якщо кількість балів перейшла максимально дозволений поріг.
BI8	Померти	1. При досягненні певних умов, Істота виконує цей варіант використання і лишається контролю штучним інтелектом, стоячи на місці, ставши “Трупом”. 2. З певним часом, або з допомогою інших обставин - “Труп” Істоти пропадає середовища симульованої екосистеми.	

З допомогою інформації, отриманої з даних таблиць і рисунків, можна ознайомитись з

1.2 Проектування програмної системи

1.2.1 Управління розробкою програмного забезпечення (англ. Software project management) – це особливий вид управління проектами, в рамках якого відбувається планування, відстеження і контроль за проектами по розробці програмного забезпечення. Ключовим моментом в управлінні проектом по розробці програмного забезпечення — це правильний вибір методів його розробки. Наявність різних методологій і підходів до розробки передбачають використання різних способів до моделювання даного процесу.

Для розробки даного проекту було обрано модель швидку розробку додатків (rapid application development model). У випадку швидкої моделі розробки, програмне забезпечення поступово розробляється з блоків конструкцій.

Використання цієї моделі розробки програмного забезпечення припускає проведення аналіз предметної області для вивчення потреб замовника, аналізу можливостей цієї моделі для її реалізації. Дана модель використовується для розробки не дуже складних і вимогливих систем, де головна мета — реалізація функцій системи. При чому вимоги не визначені відразу і повністю. Тому розробка такої системи відбувається ітераційним шляхом, поступовим еволюціюванням і розвитком деяких варіантів систем-прототипу, на яких і відбувається перевірка реалізації вимог. Іншими словами, цей процес по своїй суті є ітераційним, з етапами розробки, які повторюються, починаючи з змінених вимог і завершуючи готовою розробкою [4].

Розвиток цієї моделі розробки програмного забезпечення в рамках усього життєвого циклу розробки продукту було зображено на рисунку нижче (рисунок 1.2).

Швидка модель розробки (Rapid Application Development)

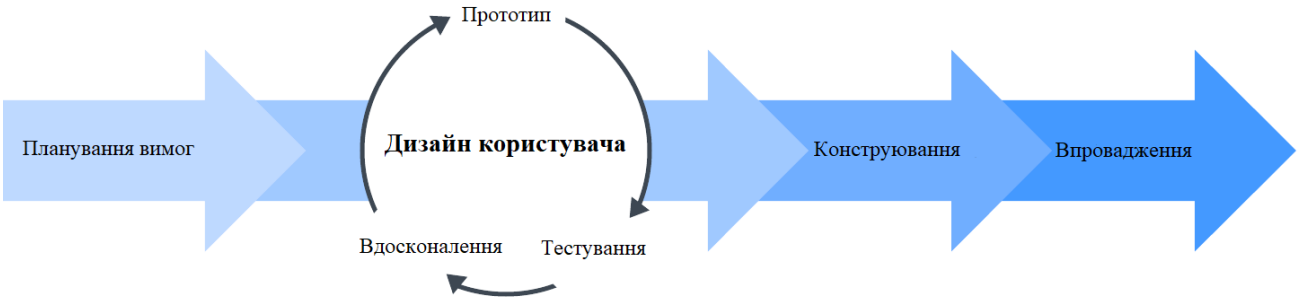


Рисунок 1.2 — Модель швидкої розробки програмного забезпечення

Швидка модель розробки додатків — це гнучка стратегія управління проектами, популярна в розробці програмного забезпечення. Ключова перевага такого підходу полягає у великій гнучкості проекту, що робить його привабливим вибором для розробників, які працюють у середовищі, що швидко розвивається, на кшталт розробки програмного забезпечення.

Ця модель зменшує час планування, та дозволяє керівникам проектів та зацікавленим сторонам точно вимірювати прогрес та спілкуватися в режимі реального часу щодо вирішення проблем чи інших змін.

Швидка модель розробки додатків чітко дотримується чотирьох фаз розробки:

1. Планування вимог;
2. Дизайн користувача;
3. Конструювання;
4. Впровадження.

Друга фаза у цій моделі розробки програмного забезпечення повторюватиметься декілька раз.

Фаза 1: Планування вимог

Фаза планування вимог веде до визначення цілей проекту. Хоча ця фаза є скороченою у порівняно з іншими методологіями управління проектами, вона є важливим кроком для досягнення остаточного успіху проекту.

Під час цього етапу розробки, клієнти (кінцеві користувачі програмного забезпечення) та члени команди розробки спілкуються, щоб визначити цілі та очікування щодо проекту, а також поточні та потенційні проблеми, які потребують вирішення.

Основний розділи цього етапу включають:

- Дослідження поточної проблеми;
- Визначення вимог до проекту;
- Доведення вимог до затвердження кожного зацікавленого боку.

Важливо, щоб кожен мав можливість оцінити цілі та очікування щодо проекту і добре обміркувати їх. Отримавши схвалення від кожного ключового зацікавленого учасника і розробника, команди в процесі обговорень можуть уникнути неправильних комунікацій та дорогих змін у процесі розробки проекту. Після цього настає наступна фаза проекту — фаза розробки дизайну користувача.

Фаза 2: Фаза дизайну

В фазі дизайну, коли проект був схвалений на засіданні команди розробників і майбутніх користувачів (заказника) — настає час перейти до розробки, створивши дизайн користувача в процесі циклу ітерацій прототипу розробки.

Ця фаза і є головною особливістю моделі швидкої розробки програмного забезпечення і те, що відрізняє її від усіх інших стратегій управління проектами. Під час цієї фази, клієнти співпрацюють з розробниками, щоб забезпечити

задоволення їх потреб на кожному кроці процесу проектування. Це майже як розробка настоюваного програмного забезпечення, де користувачі можуть протестувати кожен прототип продукту на кожному етапі з різними елементами інтерфейсу, щоб переконатися, що кінцевий продукт буде відповідати їх побажанням і очікуванням.

Усі помилки та різні особливості відпрацьовуються в цьому ітераційному процесі розробки. Розробник розробляє прототип, клієнт (кінцевий користувач) тестує його, а потім вони збираються, щоб повідомити про те, що відповідало їх очікуванням, а що ні.

Цей метод дає розробникам можливість з кожним циклом розробити більш якісніший продукт і переконатися, що вони йдуть в правильному напрямку. Цей цикл повторюється, поки на виході прототип не досягне задовільної якості.

Фаза 3: Конструювання

У фазі конструювання беруться прототипи та варіанти розробки з попереднього етапу розробки, та перетворюються у робочу модель.

Оскільки більшість проблем та змін були вирішені у минулій фазі проектування, розробники можуть побудувати остаточну модель швидше, ніж могли, дотримуючись традиційного підходу до управління проектами.

Фаза конструювання розділяється на кілька менших етапів

- Підготовка до швидкої розробки;
- Розробка програмної архітектури;
- Написання коду додатка;
- Інтеграція та тестування системи.

Команда з розробки програмного забезпечення працює на цьому етапі разом, щоб переконатися, що все працює безперебійно, а кінцевий результат задовольняє очікуванням та потребам клієнта.

Ця фаза є важливою, оскільки клієнт все ще може вплинути упродовж усього процесу. Клієнт може запропонувати нові вимоги, зміни або навіть нові ідеї, які можуть вирішити певні проблеми в міру їх виникнення.

Фаза 4: Впровадження

Це етап впровадження готового продукту до запуску. Він включає впровадження, тестування а також навчання користувачів.

Всі остаточні зміни вносяться, поки розробники та клієнти продовжують шукати помилки в системі.

Модель швидкої розробки програмного забезпечення — одна з найуспішніших моделей розробки програмного забезпечення, яка має чисельні переваги як для команд з розробки програмного забезпечення, та і для її клієнтів.

Архітектурна модель проектованої системи відповідає стилю модель-вид-контролер (Model-view-controller, MVC). Модель-вид-контролер полягає, що схема даних додатка, користувацького інтерфейсу і керуючої логіки розділена на три окремих компонента: модель, вид, контролер. Вони реалізовані таким образом, що модифікація кожного компонента може здійснюватися незалежно одне від одного.

Основна мета застосування MVC полягає в розділенні даних і бізнес-логіки від візуалізації (зовнішнього вигляду). За рахунок такого поділу підвищується можливість повторного використання програмного коду і спрощується супровід програмного забезпечення (зміни зовнішнього вигляду, наприклад, не вплинуть на функціонал в середині системи).

Модель (Model) — представляє собою об'єктну модель певної предметної області, включає в себе дані і методи роботи з цими даними, реагує на запити контролера, повертаючи дані чи змінюючи свій стан, при цьому модель не містить в собі інформації, як дані можна зобразити на пристрої виведення, а також не “спілкується” з користувачем безпосередньо.

Вид (View) – відповідає за відображення інформації (візуадізацію), одні і ті ж дані можуть представлятися різними способами, наприклад, колекцію об'єктів можна зобразити як і в табличному вигляді, графіком чи навіть з списком, з допомогою декількох різних компонентів виду (view).

Контролер (Controller) – забезпечує зв'язок між користувачем і системою, використовує модель і уявлення для реалізації необхідної реакції на дії користувача, як правило, на рівні контролера здійснюється фільтрація отриманих даних і авторизація (перевіряються права користувача на виконання дій або отримання певної інформації).

1.2.2 Побудова UML діаграми діяльності програмної системи

Під час моделювання поведінки проектованої чи аналізованої системи з'являється необхідність не лише уявити процеси зміни її станів, але і висвітлити особливості логічної і алгоритмічної реалізації виконуваних цією системою операцій. Традиційно для цієї мети використовувалися блок-схеми або структурні схеми алгоритмів. Такі схеми акцентують увагу на певному послідовному виконанні необхідних дій або елементарних операцій, котрі в загальній сумі приведуть до отримання необхідного результату.

Для зображення процесу використання операцій у мові UML застосовуються діаграми діяльності. Використовувана в них графічна нотація діаграм діяльності багато чим подібна на нотацію діаграм станів, оскільки на діаграмах так же присутні позначення переходів і станів. Кожен стан на цій діаграмі відповідає за виконання певної елементарної операції, а наступна операція відбувається лише при завершенню попередньої операції. Графічно діаграма діяльності зображається у вигляді графа діяльності, вершини якого це стан дії, а дуги — переходи з одного стану на інший.

У такому випадку, діаграму діяльності можна рахувати як окремий вид діаграми станів. Вони дають змогу реалізувати у мові UML характер

синхронного і процедурного управління. Метамодель UML надає для цього потрібні терміни і семантику [5].

Діаграма діяльності — це технологія, яка дозволяє описати логіку процедур, бізнес-процеси і потоки робіт. В багатьох випадках вони нагадують блоки-схеми, але принципова різниця між діаграмами діяльності і нотацією типу блок-схеми заключається в тому, що перші підтримують паралельні процеси.

В мові UML діяльність (activity) це певна сукупність окремих обчислень, які виконуються автоматично. Під час цього процесу окремі елементи обчислення мають можливість приводити до певного результату чи дії (action). На діаграмі діяльності зображується послідовність і логіка переходу з одного кроку діяльності до іншого. Результат в наслідок проходження через систему може спровокувати зміну стану системи чи повернення певного значення.

Стан дії (action state) це спеціальний випадок стану з певними вхідними діями, а також хоча б одним переходом. Такий перехід неявно вказує, що вхід на цю дію вже завершився. Стан такої дії не зміг би мати внутрішніх переходів, оскільки цей стан є елементарний. Звичайне використання такого стану полягає в моделюванні одного такого виконання процедури (алгоритму) або потоку управління.

Графічно такий стан дії виглядає як фігура, яка нагадує прямокутник, бічні сторони котрого змінені на опуклі дуги. Всередині такої фігури записується вираз певної дії (action-expression), який повинен бути унікальним в середині одної такої діаграми діяльності [5].

При створенні діаграми діяльності використовують тільки такі переходи, котрі спрацьовують зразу ж після завершення діяльності чи виконання необхідної дії. Такий перехід переведе діяльність в наступний стан зразу, як тільки в попередньому стані дія дійде до кінця. Такий перехід на діаграмі буде зображуватися суцільною лінією зі стрілкою.

Графічно в діаграмі діяльності розгалуження буде позначатись з допомогою невеликого ромба, всередині такого ромба немає тексту. У такий ромб може входити всього одна стрілка від такого стану дії, після виконання котрого керуючий потік має бути продовжений з допомогою одною з взаємовиключних гілок. Вхідна стрілка має бути приєднана до лівої чи верхньої вершини розгалуження. Вихідних стрілок повинно бути мінімум дві або більше, але для кожної з цих стрілок повинна бути вказана умова в формі булевого вираження [6].

В загалом дії на діаграмі діяльності робляться над різними об'єктами. Такі об'єкти ініціюють виконання певних дій, або визначають результат таких дій. Такі об'єкти вони або ініціюють виконання цих дій, або впливають на певний результат таких дій. В такому разі дії визначають виклики, які в процесі передаються від одного об'єкта діяльності в інший. Оскільки в такому випадку об'єкти грають певну роль в розумінні процесу діяльності, з'являється необхідність зобразити їх на діаграмі діяльності.

Таким чином, можна сказати, що призначення діаграми діяльності є демонстрація того, які дії користувач може зробити в системі.

Для даної системи можна вивести такі основні дії, які може виконувати користувач:

1) створення екосистеми;

2) налаштування екосистеми;

3) завантаження екосистеми;

4) збереження екосистеми;

5) переміщення камери екосистеми;

6) додавання тварини в екосистему;

7) видалення тварини з екосистеми;

8) перегляд характеристик тварини екосистеми;

9) перегляд графіку подій в екосистемі;

10) Цикл симуляції.

Дія “створення екосистеми” дає користувачеві можливість створити екосистему, таким чином система запропонує користувачеві створити екосистему з параметрами по замовчуванню, або вибрати крім цього пункт “налаштування”.

“Налаштування екосистеми” дозволяє користувачеві в меню створення екосистеми налаштувати необхідну кількість тварин, особливості ландшафту екосистеми і інші властивості системи.

“Завантаження екосистеми” - після першого створення екосистеми користувач має змогу завантажити створену екосистему в минулій сесії.

“Збереження екосистеми” - ця дія відбувається після виходу з екосистеми під час її симуляції.

“Переміщення камери по екосистемі” - користувач може переміщати камеру в екосистемі під час її симуляції.

“Додавання тварини в екосистему” дає змогу під час симуляції добавити істоту певного виду в область віртуального світу.

“Видалення тварини з екосистеми” дає змогу видалити не бажану тварину з простору екосистеми.

“Перегляд характеристик тварини в екосистемі” дає змогу переглянути інформацію про істоту яка з находиться в тій чи іншій позиції в просторі екосистеми.

“Перегляд графіку подій в екосистемі” - подає статистику подій в якому показується кількість тварин відносно часу з допомогою графіку, який виводиться на екран.

“Цикл симуляції” - це цикл в якому система симулює всі дії які відбуваються в екосистемі.

Переглянути діаграму діяльності для розроблювальної симуляції екосистеми можна на рисунку 1.3.

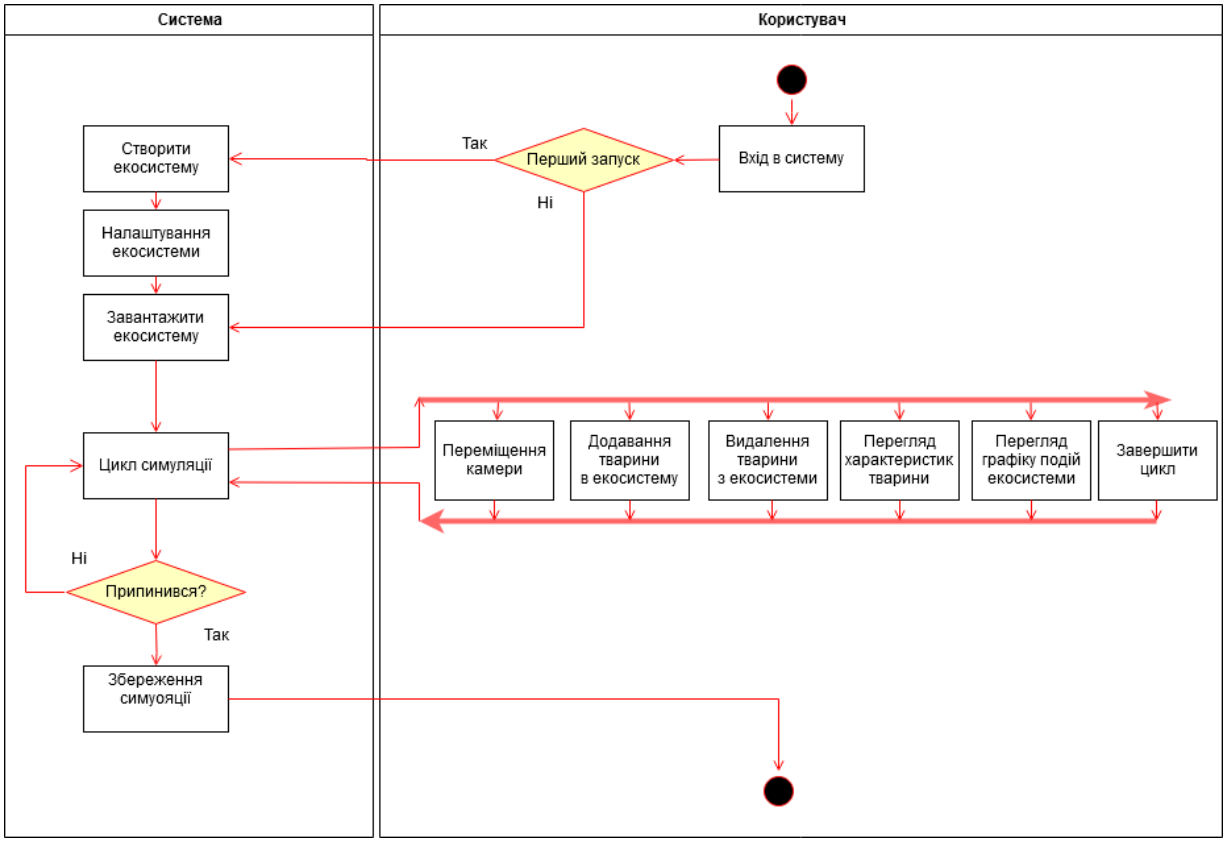


Рисунок 1.3 — Діаграма діяльності

На цій діаграмі зображаються можливі дії користувача у додатку. Початок роботи як і у більшості додатків для настільного персонального комп'ютера починається з запуску. Далі йде перше розгалуження яке визначає чи був проведений перший запуск додатка, в залежності від цього фактору відбувається або створення нової екосистеми, або завантаження старої. Під час першого запуску система пропонує користувачеві створити екосистему за умовчанням. Користувач може згодитись з цим варіантом, а може налаштувати певні параметри власноруч, такі як кількість істот, кількості водойм і суші в світі, різновидності істот і інші параметри. Так чи інакше, відбувається завантаження або старої екосистеми, або нової, котра щойно створилась. Дальше відбувається зациклення циклу в якому відбуваються ходи всіх істот, якими керує штучний

інтелект, обмірковування ходів і дій. На даному етапі окрім цього відбувається зчитування вводу, і користувач може зробити певну кількість дій.

А саме таких дій як:

- переміщення камери;
- додавання тварин в екосистему;
- видалення тварин з екосистеми;
- перегляд характеристик тварин;
- перегляд графіку подій в екосистемі;
- завершити цикл.

Користувач може пересувати камеру по віртуальному світі, яка буде показувати певну область з віртуального світу, ландшафт, істот, рослин. Користувач може навівши курсором на тварину побачити характеристики цієї тварини, такі як спрага, здоров'я, голод, вік. Користувач також має змогу видалити цю тварину з просторів світу, після чого вона безслідно зникне. Користувач також може додати в світ істоту потрібного виду, яка одразу буде дорослою.

Під час цього циклу користувач може викликати меню, в якому буде зображуватися графік кількості тварин відносно часу з початку створення світу, популяція різних видів різних тварин і інформація про те, скільки їх є в поточний момент. Також в цьому меню є кнопка “Вихід”, яка завершуватиме цикл в якому відбуваються основні дії додатка. Поточний стан світу зі всіма істотами, рослинами і ландшафтом збережеться, і додаток зафіксує, що перший запуск відбувся. Після цього додаток закриється.

1.2.3 Моделювання архітектури системи

Як впливає із технічного завдання, розроблювальна система повинна бути додатком для настільного комп'ютера для операційних систем сімейства Windows, Linux та MacOS. Вирішено будувати додаток з допомогою архітектури модель-вид-контролер (MVC).

Тому архітектурно програма буде складатися з трьох частин. Модель — в якій буде відбуватися сама програма екосистеми зі всіма її мешканцями, також буде відбуватися мислення штучного інтелекту і вибір дій. Вид — це візуальна частина, в яка буде зображувати екосистему на екранах монітора, та ввід — частина з допомогою якої користувач впливає на програму.

Візуальна частина програми буде створена з допомогою фреймворку під назвою LibGDX. LibGDX – це фреймворк, який призначений для створення ігор та додатків, він створений з допомогою мови програмування Java, деякі компоненти були розроблені з використанням мови програмування C і C++ (для більш швидшої роботи). Він дозволяє писати додатки для різних платформ використовуючи один і той самий код для всіх платформ.

Архітектура цього фреймворку дозволяє розробнику писати код, тестувати розробку на власному комп'ютері, а потім з легкістю перенести додаток на інші операційні системи. Для цього фреймворк використовує модулі для збірки додатка під кожну окрему платформу, а також незалежний модуль, який містить в собі основний код додатка.

В перелік особливостей цього фреймворку входять:

- Підтримка Windows, Linux, Mac OS X, Android, iOS а також браузери з підтримкою WebGL. Також підтримуються 32 і 64 розрядні версії операційних систем;
- Гнучкість фреймворку можна збільшити з допомогою різних розширень
- Вбудований набір віджетів для побудови графічного інтерфейсу — як для ігор, так і для програм;

- Можливість легко реалізувати логіку поведінки ефектів;
- Вставки з мови програмування C++ надають високу продуктивність додатку.

Цей фреймворк використовує сторонні бібліотеки, такі як LWJGL(Lightweight Java Game Library), OpenGL, SoundTouch Audio Library, FreeType, mpg123, Vobis, Box2D, OpenAL.

1.3 Конструювання системи

1.3.1 Вибір мови та середовища розробки

Як відомо, розроблювальна система розділена на три компонента, які взаємодіють між собою. Для полегшення розробки і подальшої підтримки додатка, навчання і взаємодії розробників різних модулів було вирішено використовувати для розробки мову програмування Java у всіх компонентах системи.

Java – це мова програмування загального призначення, яка орієнтована на класи і є об'єктно-орієнтованою мовою програмування. Ця мова програмування розроблена таким чином, щоб мати якомога найменше залежностей від реалізації. Вона призначена для того, щоб розробник додатків коли пише код один раз, він зможе запускати цей код де завгодно, тобто скомпільований код Java може запускатися на всіх платформах, які підтримують Java без необхідності перекомпіляції, тому що програми Java компілюються в байт-код, який може працювати на будь якій віртуальній машині Java (JVM) незалежно від основної архітектури комп'ютера. Синтаксис мови програмування Java схожий на синтаксис C та C++, але в ній є менше об'єктів низького рівня, ніж в C і C++. Станом на 2019 рік, Java була однією з найпопулярніших мов програмування, що використовується згідно GitHub, зокрема для веб-додатків клієнт-сервер,

яких було зафіксовано понад 9 мільйонів. Популярність призвела до розробки багатьох фреймворків лоя цієї мови програмування.

Java спочатку була розроблена Джеймсом Гослінгом у компанії Sun Microsystems (яка була придбана компанією Oracle) і випущена в 1995 році, як основний компонент платформи Java Sun Microsystems. Оригінальні та модернізовані Java компілятори, віртуальні машини та бібліотеки класів спочатку були випущені компанією під власною ліцензією. Станом на травень 2007 року, відповідно до специфікацій компанія яка розробила Java передала більшість своїх технологій, які стосувалися цієї мови програмування в публічний доступ під видом ліцензії GNU.

Останньою версією є Java 13, випущена у вересні 2019 року, та Java 11 – версія довготривалої підтримки, яка підтримується і по цей час, ця версія була випущена 25 вересня 2018 року. Oracle також випустила оновлення для застарілої версії Java 8 LTS в січні 2019 року для комерційного використання.

Програми на Java транслюються в байт-код Java, який може виконуватись в віртуальній машині Java (JVM) – віртуальною машиною, яка обробляє байт-код і передає інструкції в обчислювальний пристрій як інтерпретатор.

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і обладнання, що дозволяє виконувати додаток Java на різних пристроях, для яких існує відповідна віртуальна машина. Другою важливою особливістю технології Java — це гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Любі операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкційованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне завершення програми.

До недоліків концепції віртуальної машини відносять зниження швидкодії. Ряд вдосконалень можуть зменшити цей недолік, тобто пришвидшити виконання програми на мові Java:

- застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT технологія) з можливістю збереження версій класу в машинному коді;
- широке застосування платформно-орієнтованого коду в стандартних бібліотеках;
- апаратні засоби, котрі забезпечують пришвидшену обробку байт-кода (наприклад, технологія Jazelle, яка підтримується деякими процесорами архітектури ARM).

Виконання різних задач в середньому на мові програмування Java необхідно в півтора-два рази більше часу, чим для мови C чи C++, в деяких випадках Java справляється швидше, а в окремих специфічних випадках в 7 разів повільніше. З іншої сторони, для більшості випадків використання пам'яті Java машиною було в 10-30 раз більше, чим в такій же програмі написаній на мові програмування C/C++.

Всередині Java існує декілька основних сімейств технології:

- Java SE – Java Standard Edition, основне видання Java, вміщає в себе компілятори, API, та віртуальну машину Java. Підходить для розробки різних додатків, в першу чергу — для настільних систем;
- Java EE – Java Enterprise Edition, являє собою набір специфікацій для створення програмного забезпечення на рівні підприємства;
- Java ME – Java Micro Edition, створена для використання в пристроях з обмеженою кількістю обчислювальних ресурсів, наприклад таких як мобільний телефон чи КПК;
- Java Card – це технологія, яка надає безпечне середовище для пристроїв, що працюють зі смарт-картами та інших пристроях з дуже обмеженим обсягом пам'яті і можливостями обробки даних.

Для цього проекту було вибрано Java SE сімейство Java, що дасть змогу користатися всіма особливостями мови програмування Java без серйозних обмежень.

Для реалізації візуальної частини додатка був вибраний фреймворк LibGDX. Цей фреймворк являє собою сукупністю бібліотек для зручнішого використання графіки, вводу, взаємодії з файловою системою, ігрової фізики та звукової частини. Він може бути використаний як і для ігор так і для додатків де необхідна детальніша візуалізація об'єктів.

В графічній частині фреймворк використовує бібліотеку Lightweight Java Game Library (LWJGL). LWJGL — це відкрита графічна бібліотека, основною цілю якою є надання простого і легкого програмного інтерфейсу для створення комп'ютерних ігор на мові програмування Java.

LWJGL являє собою високопродуктивну крос-платформову бібліотеку, яка широко застосовується в розробці комп'ютерних ігор та мультимедійних додатків. Вона використовує OpenGL, OpenAL і OpenCL та забезпечує платформову незалежний доступ до різноманітних маніпуляторів, таких як геймпади, рулі та джойстики.

Основною цілю бібліотеки є створення технології, котра б дозволяла Java розробникам отримувати доступ до ресурсів, до яких доступ в поточний момент утруднений, або зовсім відсутній з Java платформи.

LWJGL доступна під ліцензією BSD. Будучи відкритою і безкоштовною, вона є основою багатьох ігрових рушіїв та бібліотек.

OpenGL (Open Graphics Library) – специфікація, яка визначає незалежний від платформи програмний інтерфейс(який незалежний від мови програмування) для створення додатків, які використовують двовимірну чи тривимірну комп'ютерну графіку.

Він включає в себе більше як 300 функцій для рисування складних тривимірних сцен із простих примітивів. Використовується для створення

комп'ютерних ігор, систем автоматизованого проектування, віртуальної реальності, візуалізації наукових досліджень.

OpenAL (Open Audio Library) – крос-платформовий програмний інтерфейс для роботи з аудіо даними. Ключовою особливістю являється робота зі звуком в тривимірному просторі і використання EAX (Environmental Audio Extensions) ефектів.

OpenCL (Open Computing Language – відкрита мова обчислень) — це фреймворк який був розроблений для створення комп'ютерних програм, зв'язаних з паралельними обчисленнями на різноманітних графічних і центральних процесорах. В OpenCL входять мови програмування, які основані на стандарті мови програмування C, C99 та інтерфейс програмування додатків. OpenCL забезпечує паралелізм на рівні інструкцій і на рівні даних і є здійсненням техніки GPGPU. OpenCL являється повністю відкритим стандартом, його використання не обкладається додатковими ліцензійними відрахуваннями.

Як середовище розробки було обрано IntelliJ IDEA Community Edition. IntelliJ IDEA — це інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python. Середовище розробки було розроблено компанією JetBrains. Середовище розробки було розроблено з використанням мови програмування Java.

Перша версія середовища з'явилась в грудні 2001 року і швидко здобула популярність як перше середовище для мови програмування Java з широким набором інтегрованих інструментів для рефакторинга, які давали можливість програмістам швидко реорганізувати вихідні тексти програм. Дизайн середовища орієнтований на продуктивність роботи програміста, дозволяючи сконцентруватись на функціональних задачах, в той час як середовище розробки бере на себе виконання рутинних операцій.

Починаючи з версії 9.0, середовище розробки стало доступним в двох редакціях: Community Edition та Ultimate Edition. Community Edition являє

собою повністю вільну версію, яка доступна широкому кругу програмістів під ліцензією Apache 2.0, в ній реалізована підтримка Java SE, Kotlin, Groovy, Scala, а також інтеграція з найбільш популярними системами управління версій. В редакції Ultimate Edition, доступній по комерційній ліцензії доступні всі функції, що є в Community Edition, але окрім цього реалізована підтримка Java EE, UML-діаграм, підрахунок покриття коду, а також підтримка інших систем управління версіями, підтримка різних мов програмування та фреймворків.

1.3.2 Реалізація основних класів та методів

У даній частині було зосереджено увагу на практичній реалізації розроблювальної системи. Необхідно навести приклад лістингів основних методів представлення даних.

Для реалізації системи була використана мова Java. Отже архітектурна складова додатку була розділена на три частини модель, вид та контролер.

Розглянемо частину з якої починається запуск програми. Фреймворк LibGDX зроблений так, щоб можна було скомпілювати проект для різних середовищ, таких як Android, iOS, Web, Desktop. В даному проекті додаток запускається в режимі Desktop, тому початок виконання програми відбувається в класі “DesktopLauncher”. Це спеціальний клас в якому відбувається налаштування певних характеристик додатку, і це клас з якого відбувається звернення до універсального ядра LibGDX.

В цьому класі задається розмір вікна додатку, назва вікна додатку, вибирається чи буде повноекранний режим, вибирається режим версії OpenGL, налаштовується режим роботи вертикальної синхронізації, вказується шлях на іконку додатка.

Давайте розглянемо як цей процес буде відбуватися з середини додатку переглянувши фрагмент коду додатку, який розкриває особливості початку додатка на персональному комп'ютері.

Лістинг 1.1 — код десктопної частини ядра додатку

```
public class DesktopLauncher
{
     public static void main (String[] arg)
     {
          LwjglApplicationConfiguration config = new
LwjglApplicationConfiguration();
          config.title = "Ecosys Sim";
          config.width = 840;
          config.height = 680;
          config.fullscreen = false;
          config.gles30ContextMajorVersion = 1;
          config.backgroundFPS = 60;
          config.vSyncEnabled = true;
          config.allowSoftwareMode = false;
          config.addIcon("foxL.png", Files.FileType.Local);
          new LwjglApplication(new GameMain(), config);
     }
}
```

З даного лістингу видно, що спочатку створюється файл конфігурації Lightweight Java Game Library. З початку він створюєтеся з стандартними налаштуваннями, які задаються в його конструкторі. Далі можна змінити певні поля класу які в решті решт вплинуть на запуск програми та змінять певні характеристики графічно.

Далі створюється клас “LwjglApplication”, в який приймає в себе універсальне ядро програми, клас “LwjglApplicationConfiguration” з налаштуваннями. Після чого програма спочатку створює необхідні об'єкти і класи для роботи, а потім зациклюється в циклі програми в якому і відбувається вся сутність додатка.

Клас “GameMain” успадковує клас “ApplicationAdapter”, який являє собою шаблоном універсального ядра системи. Він містить собі такі методи як “create” “resize(int width, int height)”, “render()”, “pause()”, “resume()” та “dispose()”.

Метод “create()” активізовується лише один раз при першому запуску програми. Він дозволяє задати параметри полів, класів на початку запуску програми, та підготувати її до подальшої експлуатації.

Метод “render()” - це цикл додатку в якому відбуваються основні події системи, її основний функціонал. В цьому циклі відбувається показ елементів інтерфейсу, здійснюється зчитування даних з усіх контролерів вводу, використовуються певні елементи аудіо системи та функціюють основні механізми системи.

Метод “resize(int width, int height)” активізовується під час зміни габаритів вікна додатка. В цьому методі слід задати нові розміри елементам інтерфейсу які знаходяться в середині додатку, щоб вони відповідали новим розмірам вікна додатка.

Метод “pause()” - це так званий стан паузи додатка, він виконується в різних системах по різному, наприклад в системах Windows – це згортання вікна, в системах типу Android – це перехід на домашній екран після натискання кнопки “Дім”.

Метод “resume()” виконується між переходами системи зі стану паузи в нормальний стан, коли додаток розгортається з трею в системах Windows, чи знову відкривається в системах Android.

Метод “dispose()” виконується під час завершального етапу програми, коли вона виключається. В цьому методі звільнюються ресурси системи, такі як оперативна пам'ять, пам'ять графічної підсистеми, при необхідності зберігаються дані.

Давайте розглянемо лістинг класу “MainMenu”. Цей клас створений, щоб зобразити головне меню додатку програми. В ньому присутні кнопки, які дозволяють створити екосистему, та вийти з додатка. Також на задньому плані зображується пуста екосистема, в котрій лише є поверхня екосистеми без жодних істот на ній (Лістинг 1.2).

Лістинг 1.2 — Код головного меню додатку

```
public class MainMenu
{
    Button createB, exitB;
    boolean isShowing, worldMustBeCreated;
    float logoPosX, logoPosY;
    Texture logo;
    World menuWorld;
    boolean moveCamUp, moveCamRight;


    TextureManager textureManager;
    MainMenu(TextureManager txtM)
    {
        menuWorld= new World(18,txtM);
        moveCamUp = moveCamRight = true;
        worldMustBeCreated = false;
        isShowing = true;
        textureManager = txtM;
        createB = new Button(txtM.menuT[0],txtM.menuT[1]);
        exitB = new Button(txtM.menuT[2],txtM.menuT[3]);
        createB.setPos((Gdx.graphics.getWidth()/2)-
(createB.getSizeX()/2),(Gdx.graphics.getHeight()/2)-
(createB.getSizeY()/2));
        exitB.setPos((Gdx.graphics.getWidth()/2)-
(exitB.getSizeX()/2),(int)(createB.getPosY()-
(exitB.getSizeY()*1.5f)));
        logo = txtM.menuT[4];
        logoPosX = (Gdx.graphics.getWidth()/2) -
(logo.getWidth()/2);
        logoPosY = (Gdx.graphics.getHeight()*0.75f) -
(logo.getHeight()/2);
        menuWorld.fillWorldforMenu();
    }

    public void render(SpriteBatch batch)
    {
        if(isShowing)
        {
            menuWorld.render(batch);
            createB.renderButton(batch);
            exitB.renderButton(batch);
```

Продовження лістингу 1.2

```
            batch.draw(logo,logoPosX,logoPosY);
            if(moveCamUp)
```

```
            {
                menuWorld.moveCameraUp();
                if (menuWorld.cameraPosY >
((menuWorld.worldSize+0.9f)* menuWorld.textureSize) -
Gdx.graphics.getHeight())
                    moveCamUp = false;
            }
            else
            {
                menuWorld.moveCameraDown();
                if (menuWorld.cameraPosY <= menuWorld.textureSize)
                    moveCamUp = true;
            }
            if(moveCamRight)
            {
                menuWorld.moveCameraRight();

if(menuWorld.cameraPosX>((menuWorld.worldSize+0.9f)*menuWorld.textu
reSize)-Gdx.graphics.getWidth())
                    moveCamRight = false;
            }
            else
            {
                menuWorld.moveCameraLeft();
                if(menuWorld.cameraPosX<=menuWorld.textureSize)
                    moveCamRight = true;
            }
        }
    }

    public void listner()
    {
        if(createB.wasPushed())
        {
            isShowing = false;
            createWorld();
        }
        if(exitB.wasPushed()) Gdx.app.exit();
    }

    public boolean createNow()
    {
        if(worldMustBeCreated)
        {
            worldMustBeCreated = false;
            return true;
```

Продовження лістингу 1.2

```
        }
```

```
        return false;
    }

    public boolean isShowing()
    {
        return isShowing;
    }
    public void showingFalse()
    {
        isShowing = false;
    }
    public void showingTrue()
    {
        isShowing = true;
    }
    private void createWorld()
    {
        worldMustBeCreated = true;
    }
}
```

В даному лістингу видно, що спочатку створюється світ з розміром 18 клітинок, в цей світ передається посилання на менеджер текстур, також посилання на цей менеджер зберігається в самому класі, щоб спростити доступ до нього програмісту. Далі створюються дві кнопки, це кнопка “Створити екосистему” та “Вихід”, в кожну з цих кнопок передаються два посилання, одне це текстура кнопки в стані спокою, а друге це текстура кнопки в той час коли на неї натискають. Далі вираховуються координати цих кнопок відносно розширення сторін екрану, перша кнопка розраховується так, щоб вона була в самому центрі вікна додатку, друга, щоб вона була дещо нижче від першої кнопки. Далі передається посилання на логотип додатку, який знаходиться в центрі вищої половини додатку. Далі викликається метод з класу, який являє собою світ. В цьому методі описано як повинен виглядати світ екосистеми який призначений для показу в головному меню додатка.

Далі в цьому класі описується метод який буде зображати меню в самому додатку. Він зображує меню, якщо воно активно і в ньому відбувається анімація переміщення камери по світу, який створений для показу в меню. Камера

переміщається по світу в верх і в праву сторону до тих пір, поки вона не досягне країв світу, а тоді вона переміщується в протилежному напрямку.

Наступний метод цього класу створений для того, щоб зчитувати інформацію про взаємодію з інтерфейсом меню, натисканням кнопок. Якщо перша кнопка була натиснута, то здійснюється вхід в основну частину додатка та створюється основний світ. При натисканні другої кнопки, яка називається “Вихід” - здійснюється вихід з додатку, та завершуються усі процеси додатка, вікно додатка пропадає з екрану комп'ютера.

Наступний метод дає команду створити світ, він виконуватиметься при натисканні першої кнопки додатку.

Розглянемо літинг живої сутності екосистеми, яка буде діяти, якщо вона буде жива, чи без діяти, якщо та буде мертвою.

Лістинг 1.3 — Код живої сутності

```
public class LivingEntity{
    public int colourMaterialIndex;
    public Species species;
    public Material material;

    public Coord coord;
    [HideInInspector]
    public int mapIndex;
    [HideInInspector]
    public Coord mapCoord;

    protected bool dead;

    public virtual void Init (Coord coord) {
        this.coord = coord;
        transform.position = Environment.tileCentres[coord.x,
coord.y];
```

Продовження лістингу 1.3

```
        var meshRenderer =
transform.GetComponentInChildren<MeshRenderer> ();
        for (int i = 0; i<meshRenderer.sharedMaterials.Length; i++)
```

```
        {
            if (meshRenderer.sharedMaterials[i] == material) {
                material = meshRenderer.materials[i];
                break;
            }
        }
    }

    protected virtual void Die (CauseOfDeath cause) {
        if (!dead) {
            dead = true;
            Environment.RegisterDeath (this);
            Destroy (gameObject);
        }
    }
}
```

В цьому лістингу зображено одиницю живої сутності системи, вона займається діяльністю в живому стані, та видаляється з системи, коли помирає.

Давайте розглянемо клас який являє з себе тварину, яка буде існувати в просторах віртуальної екосистеми в лістингу нижче (Лістинг 1.4)

Лістинг 1.4 — Клас живої тварини

```
public class Animal extends LivingEntity
{
    public const int maxViewDistance = 10;

    [EnumFlags]
    public Species diet;

    public CreatureAction currentAction;
    public Genes genes;

    float timeBetweenActionChoices = 1;
    float moveSpeed = 1.5f;
    float timeToDeathByHunger = 200;
    float timeToDeathByThirst = 200;
    float drinkDuration = 6;
    float eatDuration = 10;
```

Продовження лістингу 1.4

```
    float criticalPercent = 0.7f;

    // Visual settings:
```

```
    float moveArcHeight = .2f;

    [Header ("State")]
    public float hunger;
    public float thirst;

    protected LivingEntity foodTarget;
    protected Coord waterTarget;

    bool animatingMovement;
    Coord moveFromCoord;
    Coord moveTargetCoord;
    Vector3 moveStartPos;
    Vector3 moveTargetPos;
    float moveTime;
    float moveSpeedFactor;
    float moveArcHeightFactor;
    Coord[] path;
    int pathIndex;

    float lastActionChooseTime;
    const float sqrtTwo = 1.4142f;
    const float oneOverSqrtTwo = 1 / sqrtTwo;

    public void Init (Coord coord) {
        base.Init (coord);
        moveFromCoord = coord;
        genes = Genes.RandomGenes (1);
        ChooseNextAction ();
    }



    protected void Update () {
        hunger += Time.deltaTime * 1 / timeToDeathByHunger;
        thirst += Time.deltaTime * 1 / timeToDeathByThirst;
        if (animatingMovement) {
            AnimateMove ();
        } else {
            HandleInteractions ();
            float timeSinceLastActionChoice = Time.time -
lastActionChooseTime;
            if (timeSinceLastActionChoice >
timeBetweenActionChoices) {
                ChooseNextAction ();
```

Продовження лістингу 1.4

```
            }
        }
        if (hunger >= 1) {
```

```
            Die (CauseOfDeath.Hunger);
        } else if (thirst >= 1) {
            Die (CauseOfDeath.Thirst);
        }
    }
    protected void ChooseNextAction () {
        lastActionChooseTime = Time.time;
        bool currentlyEating = currentAction ==
CreatureAction.Eating && foodTarget && hunger > 0;
        if (hunger >= thirst || currentlyEating && thirst <
criticalPercent) {
            FindFood ();
        }
        else {
            FindWater ();
        }
        Act ();
    }
    protected void FindFood () {
        LivingEntity foodSource = Environment.SenseFood (coord,
this, FoodPreferencePenalty);
        if (foodSource) {
            currentAction = CreatureAction.GoingToFood;
            foodTarget = foodSource;
            CreatePath (foodTarget.coord);
        } else {
            currentAction = CreatureAction.Exploring;
        }
    }
    protected virtual void FindWater () {
        Coord waterTile = Environment.SenseWater (coord);
        if (waterTile != Coord.invalid) {
            currentAction = CreatureAction.GoingToWater;
            waterTarget = waterTile;
            CreatePath (waterTarget);
        } else {
            currentAction = CreatureAction.Exploring;
        }
    }
    protected int FoodPreferencePenalty (LivingEntity self,
LivingEntity food) {
        return Coord.SqrDistance (self.coord, food.coord);
    }
    protected void Act () {
        switch (currentAction) {
```

Продовження лістингу 1.4

```
            case CreatureAction.Exploring:
                StartMoveToCoord (Environment.GetNextTileWeighted
(coord, moveFromCoord));
```

```
                break;
            case CreatureAction.GoingToFood:
                if (Coord.AreNeighbours (coord, foodTarget.coord))
{
                    LookAt (foodTarget.coord);
                    currentAction = CreatureAction.Eating;
                } else {
                    StartMoveToCoord (path[pathIndex]);
                    pathIndex++;
                }
                break;
            case CreatureAction.GoingToWater:
                if (Coord.AreNeighbours (coord, waterTarget)) {
                    LookAt (waterTarget);
                    currentAction = CreatureAction.Drinking;
                } else {
                    StartMoveToCoord (path[pathIndex]);
                    pathIndex++;
                }
                break;
        }
    }
    protected void CreatePath (Coord target) {
        if (path == null || pathIndex >= path.Length ||
(path[path.Length - 1] != target || path[pathIndex - 1] !=
moveTargetCoord)) {
            path = EnvironmentUtility.GetPath (coord.x, coord.y,
target.x, target.y);
            pathIndex = 0;
        }
    }
    protected void StartMoveToCoord (Coord target) {
        moveFromCoord = coord;
        moveTargetCoord = target;
        moveStartPos = transform.position;
        moveTargetPos = Environment.tileCentres[moveTargetCoord.x,
moveTargetCoord.y];
        animatingMovement = true;

        bool diagonalMove = Coord.SqrDistance (moveFromCoord,
moveTargetCoord) > 1;
        moveArcHeightFactor = (diagonalMove) ? sqrtTwo : 1;
        moveSpeedFactor = (diagonalMove) ? oneOverSqrtTwo : 1;
        LookAt (moveTargetCoord);
    }
```

Продовження лістингу 1.4

```
    protected void LookAt (Coord target) {
        if (target != coord) {
            Coord offset = target - coord;
```

```
            transform.eulerAngles = Vector3.up * Mathf.Atan2
(offset.x, offset.y) * Mathf.Rad2Deg;
        }
    }

    void HandleInteractions () {
        if (currentAction == CreatureAction.Eating) {
            if (foodTarget && hunger > 0) {
                float eatAmount = Mathf.Min (hunger, Time.deltaTime
* 1 / eatDuration);
                eatAmount = ((Plant) foodTarget).Consume
(eatAmount);
                hunger -= eatAmount;
            }
        } else if (currentAction == CreatureAction.Drinking) {
            if (thirst > 0) {
                thirst -= Time.deltaTime * 1 / drinkDuration;
                thirst = Mathf.Clamp01 (thirst);
            }
        }
    }

    void AnimateMove () {
        moveTime = Mathf.Min (1, moveTime + Time.deltaTime *
moveSpeed * moveSpeedFactor);
        float height = (1 - 4 * (moveTime - .5f) * (moveTime - .5f))
* moveArcHeight * moveArcHeightFactor;
        transform.position = Vector3.Lerp (moveStartPos,
moveTargetPos, moveTime) + Vector3.up * height;
        if (moveTime >= 1) {
            Environment.RegisterMove (this, coord, moveTargetCoord);
            coord = moveTargetCoord;

            animatingMovement = false;
            moveTime = 0;
            ChooseNextAction ();
        }
    }
    void OnDrawGizmosSelected () {
        if (Application.isPlaying) {
            var surroundings = Environment.Sense (coord);
            Gizmos.color = Color.white;
            if (surroundings.nearestFoodSource != null) {
```

Продовження лістингу 1.4

```
                Gizmos.DrawLine (transform.position,
surroundings.nearestFoodSource.transform.position);
            }
            if (surroundings.nearestWaterTile != Coord.invalid) {
```

```
                Gizmos.DrawLine (transform.position,
Environment.tileCentres[surroundings.nearestWaterTile.x,
surroundings.nearestWaterTile.y]);
            }

            if (currentAction == CreatureAction.GoingToFood) {
                var path = EnvironmentUtility.GetPath (coord.x,
coord.y, foodTarget.coord.x, foodTarget.coord.y);
                Gizmos.color = Color.black;
                for (int i = 0; i < path.Length; i++) {
                    Gizmos.DrawSphere
(Environment.tileCentres[path[i].x, path[i].y], .2f);
                }
            }
        }
    }
}
```

В даній частині програмного коду зображується реалізація класу живої тварини світу. Тварина наділяється такими параметрами як максимальна дальність бачення, гени, час між діями, швидкість переміщення, час до смерті від голоду, час до смерті від спраги, час до пиття води, час до вживання їжі, критичний фактор, параметр, який відповідає, що тварина може їсти, пройдений час, фактор швидкості, фактор ширини.

Наступний метод “Update” оновлює стан тварини, вирішує чи та буде переміщатись та в якому напрямку, вирішує чи тварина буде шукати їжу чи воду. Якщо тварина сильно спрагла чи сильно голодна — вона помирає.

Метод “FindFood” дозволяє тварині найти шлях до найближчої одиниці їжі в межах радіусу видимості тварини, якщо тварина не бачить їжі в радіусі видимості — вона буде досліджувати віртуальне середовище в пошуках їжі. Коли тварина находить їжу, вона з'їдає її.

Метод “FindWather” дає змогу тварині шукати водойму в радіусі видимості тварини, якщо тварина не бачить водойми, вона буде досліджувати місцевість в пошуках води. Якщо тварина находить водойму, то вона стає на сусідню клітинку, та випиває воду з цієї водойми.

Наступний метод “Act” вирішує, що вона буде робити, коли настав час реагувати на середовище, вона буде або досліджувати, або йти до їжі, або переміщатись до водойми.

Метод “CreatePath” створює шлях по якому тварина добереться до потрібної їй цілі, якщо вона зараз в даний момент не має в своїй пам'яті іншого шляху до необхідного ресурсу, чи для простого дослідження.

Метод “StartMoveToCoord” бере дані записані з допомогою минулого методу, і дає змогу переміщати тварину по врахованому шляху паралельно перевіряючи чи зник, або змістився об'єкт, до якого вона пересувається.

Наступний метод “LookAt” дає попередньому методу бачення цілі, та стеження за нею під час її переміщення до неї, та з врахуванням відстані до цієї цілі.

Метод “HandleInteractions” дозволяє тварині взаємодіяти з цілю, якщо тварина з'їдає їжу на шляху, їй нараховуються бали їжі, а також, якщо вона випиває воду з водойми, тоді нараховуються бали спраги.

В даній програмі існує два види тварин, одні це зайці, які харчуються морквою, другі це лисиці, які харчуються зайцями. Для знаходження їжі і води, система використовує методи з класу “Environment”.

Тому варто розглянути лістин окремих методів “SenseWater”, “SenseFood” та “SensePotentialMates”, які є методами класу “Environment”.

Лістинг 1.5 — частина програмного коду класу “Environment”

```
    public static Coord SenseWater (Coord coord) {
        var closestWaterCoord = closestVisibleWaterMap[coord.x,
coord.y];
        if (closestWaterCoord != Coord.invalid) {
            float sqrDst = (tileCentres[coord.x, coord.y] -
tileCentres[closestWaterCoord.x, closestWaterCoord.y]).sqrMagnitude;
```

Продовження лістингу 1.5

```
            if (sqrDst <= Animal.maxViewDistance *
Animal.maxViewDistance) {
                return closestWaterCoord;
```

```
            }
        }
        return Coord.invalid;
    }
    public static LivingEntity SenseFood (Coord coord, Animal self,
System.Func<LivingEntity, LivingEntity, int> foodPreference) {
        var foodSources = new List<LivingEntity> ();
        List<Species> prey = preyBySpecies[self.species];
        for (int i = 0; i < prey.Count; i++) {
            Map speciesMap = speciesMaps[prey[i]];
            foodSources.AddRange (speciesMap.GetEntities (coord,
Animal.maxViewDistance));
        }
        foodSources.Sort ((a, b) => foodPreference (self,
a).CompareTo (foodPreference (self, b)));
        for (int i = 0; i < foodSources.Count; i++) {
            Coord targetCoord = foodSources[i].coord;
            if (EnvironmentUtility.TileIsVisibile (coord.x, coord.y,
targetCoord.x, targetCoord.y)) {
                return foodSources[i];
            }
        }
        return null;
    }
    public static List<Animal> SensePotentialMates (Coord coord,
Animal self) {
        Map speciesMap = speciesMaps[self.species];
        List<LivingEntity> visibleEntities = speciesMap.GetEntities
(coord, Animal.maxViewDistance);
        var potentialMates = new List<Animal> ();
        for (int i = 0; i < visibleEntities.Count; i++) {
            var visibleAnimal = (Animal) visibleEntities[i];
            if (visibleAnimal != self &&
visibleAnimal.genes.isMale != self.genes.isMale) {
                if (visibleAnimal.currentAction ==
CreatureAction.SearchingForMate) {
                    potentialMates.Add (visibleAnimal);
                }
            }
        }
        return potentialMates;
    }
```

Розглянувши даний лістинг, можна спостерігати те, що в якості параметрів в дані методи передається передається клас “coord”, в якому містяться координати. Це координати обєкта, який вирішив знайти їжу, водойм, чи іншу істоту для подальшого розмноження і створення потомства. В методі де істота вирішила знайти собі їжу — відбувається перебирання координат їжі, вказівник на їжу яка доступна в радіусі видимості додається в список типу List. Потім, якщо цей список не пустий — він сортується, і в решті решт найближчий об'єкт “їжа” повертається цим методом, а коли такі об'єкти відсутні в радіусі огляду істоти — повертається значення null.

В методі “SenseWater” такий пошук є значно простішим, в радіусі видимості вираховується лінія до одної з найближчих водойм, і повертаються координати однієї з таких водойм, в інакшому випадку повертається значення яке позначає відсутність водойми в радіусі зору істоти.

Метод “SensePotentialMates” бере з точки знаходження звіра радіус його зору, і перевіряється на присутність в цьому радіусі істот такого самого виду та жіночої статі. Всі такі істоти які знаходяться в радіусі зору додаються в список, потім в кінці функції метод повертає цей список. В деяких випадках цей список може бути порожнім.

Далі варто розглянути метод який провокує створення простору екосистеми, та заповнює її різними об'єктами. Такий метод ініціалізації середовища зображений в лістингу 1.6.

Лістинг 1.6 – Частина програмного коду, яка відповідає за ініціалізацію простору

```
 void Init (int size, Gparams params) {
        TerrainGenerator terrainGenerator = new
TerrainGenerator(size, params);
        terrainData = terrainGenerator.Generate ();

        tileCentres = terrainData.tileCentres;
        walkable = terrainData.walkable;
```

Продовження лістингу 1.6

```
        size = terrainData.size;

        int numSpecies = System.Enum.GetNames (typeof
(Species)).Length;
        preyBySpecies = new Dictionary<Species, List<Species>>();
        predatorsBySpecies=new Dictionary<Species,List<Species>>();

        speciesMaps = new Dictionary<Species, Map> ();
        for (int i = 0; i < numSpecies; i++) {
            Species species = (Species) (1 << i);
            speciesMaps.Add (species, new Map (size,
mapRegionSize));

            preyBySpecies.Add (species, new List<Species> ());
            predatorsBySpecies.Add (species, new List<Species> ());
        }
```

Даний лістинг, викликається кожного разу, коли створюється віртуальний простір симульованої екосистеми. З самого початку важливо, щоб була інформація про ландшафт простору, тому в першу чергу запускається генератор ландшафту. Генератор ландшафту генерує ландшафт відповідно до розмірів заданими на вході метода ініціалізації віртуального простору. Також він отримує необхідні параметри, які вказують на кількість суші, водойм, трави, земні та інші особливості ландшафту, які слід врахувати при створенні ландшафту.

Дальше створюються об'єкти для травоїдних істот, які вона зможе вживати, щоб продовжити своє існування та отримати шанс створити нове покоління популяції виду цих істот.

Після цього, в симуляції починають створювати необхідну кількість істот різних видів, що додаються в випадкову частину прохідної області екосистеми, та вносяться у список істот.

Дані лістинги дозволяють зрозуміти основну частину програми, її основні механізми, реалізацію створення екосистеми та реалізацію штучного інтелекту, та істот з таким інтелектом, які будуть взаємодіяти в цьому віртуально симульованому просторі, та симулювати життєво необхідні процеси, такі які відбуваються в реальних живих тваринах на нашій планеті.

2 ПУБЛІКАЦІЯ І ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Розгортання програмної системи та системні вимоги

Програмна система може бути завантажена на пристрій на базі операційної системи типу Windows, Linux та MacOS та допомогою різних онлайн сервісів цифрової дистрибуції. Цим сервісам відповідають такі площадки для продажу додатків як “Steam” та “itch.io” . Для цього потрібно ввести в пошуковому розділі одного з таких сервісів слова “Ecosys sim” та натиснути кнопку “Завантажити”.

Для того щоб можна було публікувати додаток у одному з таких онлайн сервісів цифрової дистрибуції необхідно здійснити ряд дій.

Steam – це онлайн-сервіс цифрового поширення комп'ютерних ігор і програм, розроблений і підтримуваний компанією Valve. Steam виконує роль засобу технічного захисту авторських прав, платформи для багатокористувацьких ігор і потокового стримінгу відео з гри, а також соціальної мережі для гравців. Програмний клієнт Steam також забезпечує установку і регулярне оновлення ігор, хмарні збереження ігор, текстову і голосовий зв'язок між гравцями. Щоб користувач зміг завантажити додаток, він повинен встановити на своєму комп'ютері клієнт цієї платформи та бути зареєстрованим та авторизованим в системі через цей клієнт.

Щоб викласти додаток в магазині, перш за все потрібно здійснити відповідні приготування, які вимагає цей сервіс цифрової дистрибуції:

1. Заповнити електронні документи;
2. Внести внесок за додаток;
3. Надати банківську і податкову інформацію, а також підтвердити свою особистість;
4. Отримавши доступ до Steamworks, почати готувати продукт до випуску, вказавши дані, опис додатку та його ціну;

5. Дочекатися перевірки додатку яка займатиме від 1 до 5-ти днів;
6. Публікувати додаток.

Після виконання цих дій, додаток буде опублікований на даній прощадці цифрової дистрибуції, і користувач буде мати можливість придбати чи завантажити додаток.

Itch.io — це онлайн-сервіс для розміщення, продажу та завантаження інді-ігор. Сервіс було запущено в березні 2013 року і станом на липень 2019 року в сервісі знаходяться більше як 100 тисяч ігор.

Щоб завантажити додаток в цей сервіс, необхідно авторизуватися, та виконати наступні кроки:

1. Потрібно перейти по інтерфейсу по пунктах Dashboard-->Projects-->Create;
2. Заповнити анкету в якій буде вказана інформація про гру, завантажити скріншоти, іконку для магазина і сам додаток;
3. Нажати кнопку “Save”.

Після виконання цих дій — додаток одразу ж зявиться в даному сервісі цифрової дистрибуції.

Для запуску цієї програми необхідно, щоб настільний персональний комп'ютер відповідав мінімальним системним вимогам, або був потужніший за ці вимоги:

- Операційна система: Windows XP, Ubuntu 9.04, MacOS 10.5;
- Центральний процесор: Pentium 3 1.4 ГГц / Amd Athlon XP ;
- Відеокарта: ATI Radeon X300 Seties / NVIDIA GeForce MX 200 32MB;
- Сховище інформації: 2 гігабайта вільного простору.

Додаток гарантовано запускатиметься як і на старих настільних комп'ютерах так і на сучасному новому обладнанні.

2.3 Опис типових схем використання системи

При запуску програми, відкриється вікно додатку в якому буде зображуватися основний вміст додатку (див. рис. 2.1)

Рисунок 2.1 — Стартове вікно додатку

В даному вікні присутні кнопки створення екосистеми, та вихід з додатку. Натискання по першій кнопці, яка в перекладі з англійської мови перекладається як “Нова Екосистема” призведе до переходу в меню, в якому користувач зможе налаштувати певні характеристики екосистеми та почати симуляцію віртуальної екосистеми.

Натискання на кнопку, яка перекладається як “Вихід” призведе до негайного виходу з програми.

З верху рисунка рисується логотип програми, а на задньому фоні зображується порожня екосистема, в якій присутній лише ландшафт, крім цього переміщується камера по горизонталі і вертикалі. Камера змінює напрям переміщення, якщо вона досягне до одної з границь екосистеми.

Отже припустимо, що користувач натиснув на кнопку “Нова Екосистема”. В такому випадку в вікні додатка появиться інтерфейс з кнопками, в якому можна буде налаштувати нову екосистему (див. рис. 2.2).

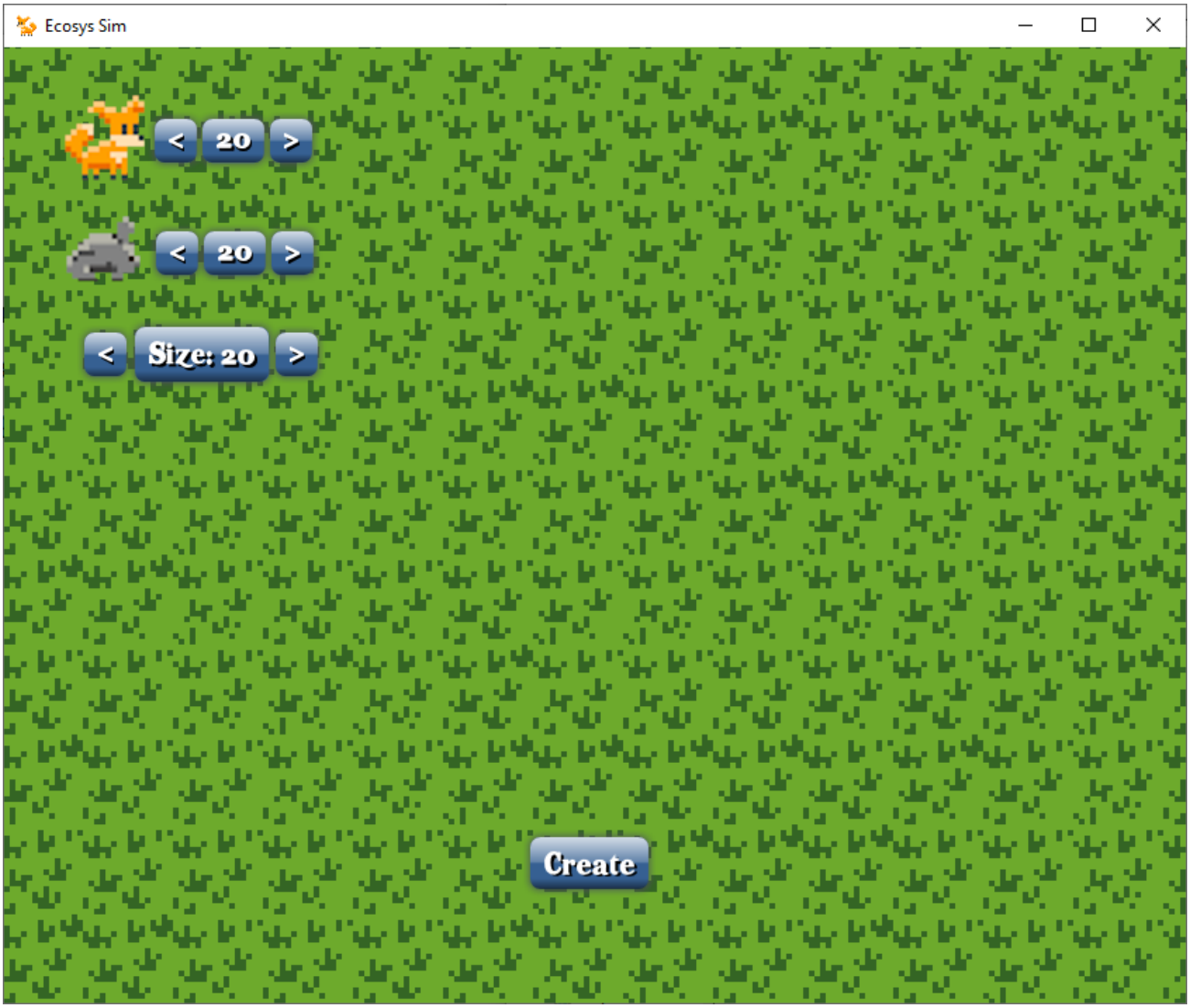


Рисунок 2.2 — Інтерфейс створення нової екосистеми

Отже в даному вікні можна налаштувати такі характеристики екосистеми, як кількість тих чи інших істот, та вказати розміри екосистеми. Перший рядок відповідає за кількість істот типу “Лисиць” в екосистемі, другий рядок відповідає за кількість “Зайців” в екосистемі. Третій рядок відповідає за розмір простору віртуального середовища.

Натискаючи на кнопки зі стрілочками “Вправо” та “Вліво” відбувається зміна значень полів, біля яких стоять ці кнопки. При натисканні кнопки “Вліво” - число поля зменшується на одиницю, доки не досягне нуля в налаштуваннях істоти певного виду. Число поля “Розмір” не може зменшитись менше 15. При натисканні кнопки “Вправо”, число поля збільшується на одиницю. Якщо число, яке відповідає за кількість певних істот досягло 50, то при натисканні кнопки — воно не збільшуватиметься, для поля, яке відповідає за розмір світу — стоїть обмеження в 150 клітинок розміру світу. Після налаштування, користувач повинен натиснути кнопку “Створити”, щоб створився віртуальний простір зі всіма заданими істотами та розміром екосистеми.

Вданому додатку існують два типи істот, це “Лисиці” та “Зайці”, ці істоти наділені певним штучним інтелектом, який надає змогу “Жити” своїм життям в просторах віртуального середовища екосистеми.

Істота типу “Заєць” має середню швидкість пересування, та має низькі характеристики спраги та голоду, ця істота харчується такими об'єктами, як “Морква” та п'є воду з водойми. Істота може створювати нове потомство приблизно кожні 2 хвилини.

Істота типу “Лисиця” має високу швидкість пересування, та має високі характеристики спраги та голоду, ця істота харчується об'єктами інших видів, які наділені штучним інтелектом, та які харчуються не живими об'єктами, в даному випадку це “Зайці”, також вона п'є воду з водойми. Ця істота може створювати нове потомство приблизно кожні 5 хвилин.

Отже будемо вважати, що користувач вказав параметри розміру екосистеми 40, кількість “Лисиць” 2 а кількість “Зайців” 14. Після вказування налаштувань користувач повинен нажати кнопку “Створити”, щоб система почала діяти далі.

Після цього створиться нова екосистема з відповідними налаштуваннями. Вікно програми показуватиме ліву-нижню область симульованої екосистеми, що зображено на рисунку 2.3.

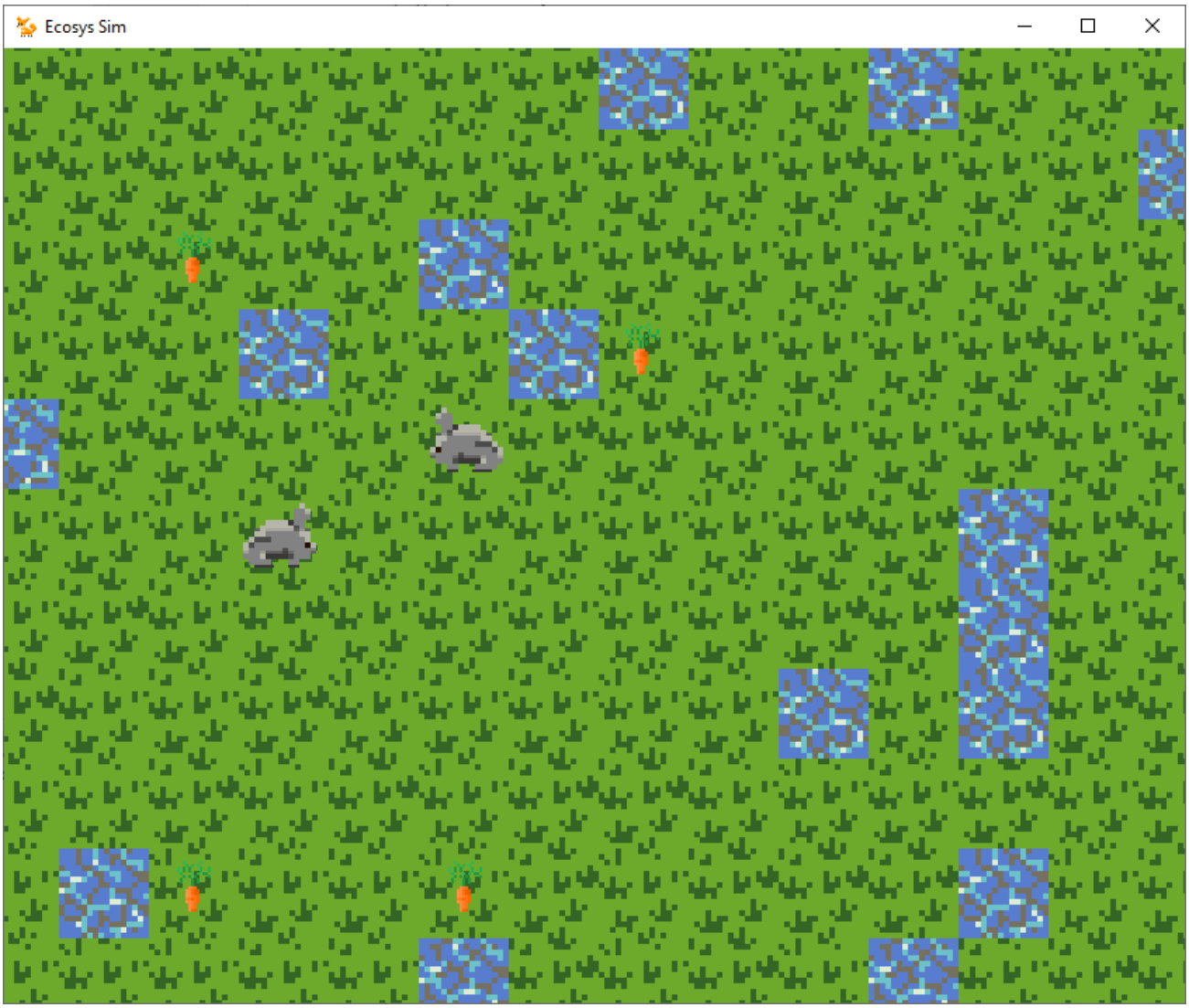


Рисунок 2.3 — Екосистема з “Зайцями” та їжею

Отже після створення екосистеми можна побачити у вікні додатка певну ділянку цієї екосистеми, в даному випадку вона містить 4 морквин, які може з'їсти істота, та власне два “Зайця”. Морква це об'єкт екосистеми, який може з шансом раз в хвилину з'явитись у клітинці в вірогідністю 3%.

Щоб відбулася певна послідовність подій в екосистемі потрібно почекати їх здійснення. В наступному рисунку зображено сусідню область екосистеми після двох хвилин активності (рис. 2.4).

Рисунок 2.4 — Екосистема з “Лисицею” та мертвив “Зайцем”

Щоб зсунути камеру, використовуються кнопки “W” для зсуву вверх, “D” для зсуву вправо, “S” для зсуву вниз, та “A” для зсуву камери вліво. Швидкість камери можна змінити з допомогою таких клавіш як “+” пришвидшити і “-” сповільнити.

Після двох хвилин очікування в сусідній області з правої сторони істота “Лисиця” вполювала одного “Зайця”, оскільки показник голоду цієї істоти досягнув певної точки, вона вимушена харчуватися об'єктами підконтрольними штучним інтелектом типу “Заєць”. На місці вбитого “Зайця” з'явився труп істоти, яким можуть харчуватися хижі істоти декілька раз. Хижа істота, яка вполювала “Зайця” задовольнила свої потреби голоду та перевелась в пасивний режим, в якому вона не переслідує інших істот. З допомогою такого трупа вбитого “Зайця”, хижа істота може проіснувати 5 хвилин, оскільки вона поїдає цей труп не одразу а з деяким часом та шматочками, в залежності від голоду хижої істоти.

Після проходження певного проміжку часу, популяція кроликів збільшилась майже у двічі, а популяція “Лисиць” становила 3 особи. Через нестачу їжі, деякі кролики почали вимирати від голоду, а хижакам не треба було полювати на кроликів, щоб втамувати свій голод.

Тоді з певним проміжком часу, популяція кроликів почала спадати, а популяція хижаків — збільшуватись. Це детально видно на наступному рисунку вікна програми (рис. 2.5).

Ще через деякий проміжок часу, приблизно 40 хвилин реального часу, популяція хижаків зросла настільки, що вполювала всіх кроликів і почала помалу вимирати. З цим і завершується дана симуляція екосистеми з такими конкретними параметрами і настройками істот їх видів.

Щоб вийти з додатку можна натиснути на хрестик в правій-верхній частині вікна, або натиснути кнопку “Escape”, після чого додаток негайно завершиться.

Рисунок 2.5 — Збільшення популяції “Лисиць”

2.3 Тестування системи

Тестування програмного забезпечення — це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Під час цієї процедури можуть оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники;
- правильна відповідність для усіх можливих вхідних даних;
- виконання функцій за прийнятний час;

- практичність;
- сумісність з програмними забезпеченням та операційними системами;
- відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів так і для замовників [21].Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершено). Процес розробки зазвичай передбачає коли та як буде відбуватися тестування.

В цій роботі тестування проводилось в ручному режимі, під час якого був виявлений виліт програми поли камера в екосистемі доходить до верхньої межі. Тестування слід проводити відразу після впровадження нової функції в додаток, щоб пізніше можна було легше виявити причину. Такий підхід тестування називається поетапним тестуванням.

Та на противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно [22].

3 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

Метою цього розділу дипломної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності симульованої екосистеми для систем типу Windows, Linux та MacOS.

Для здійснення розрахунків було використано реальні дані проведення науково-дослідної роботи, був описаний технологічний процес розробки із зазначенням трудомісткості кожної операції, враховано суму матеріальних затрат, та витрат на оплату праці основного і допоміжного персоналу, включно з відрахуванням на соціальні заходи, обчислено додаткові затрати, а також було визначено економічну ефективність та термін окупності продукту.

3.1 Планування стадій та етапів проектування програмного забезпечення

Відповідно до ст. 1 Закону № 3792, комп'ютерна програма – це набір інструкцій у вигляді слів, цифр, кодів, схем, символів чи у будь-якому іншому вигляді, виражених у формі, придатній для зчитування комп'ютером, які приводять його в дію для досягнення певної мети або результату. В такому самому значенні розуміється цей термін і в контексті податкового обліку (пп.8.2.2 Закону про прибуток). Статтею 8 Закону № 3792 передбачено, що комп'ютерні програми є об'єктом авторського права [7].

Діяльність з програмування — це процес розробки програми, який є представлений послідовністю таких кроків: формулювання вимог до програми та розробка алгоритмів; кодування (запис алгоритму на необхідній мові програмування), відлагодження коду, тестування програмного продукту, доопрацювання програми, розробка довідкової системи.

Проектування програмного забезпечення складається з послідовних, цілеспрямованих, взаємоповязаних та взаємообумовлених етапів, на які можна

розділити весь часовий відрізок, який відводиться на розробку програмного продукту (таблиця 3.1).

Таблиця 3.1 — Етапи розробки проекту

Найменування		Вид роботи	
стадії	Етапу	шифр	зміст роботи
1 Підготовча стадія	1.1 Вивчення стану предметної області	1.1.1	Дослідження проблеми
		1.1.2	Вибір платформи для реалізації проекту
		1.1.3	Вивчення та аналіз аналогічних розробок
		1.1.4	Економічне обґрунтування доцільності виконання проекту
	1.2 Розробка технічного завдання	1.2.1	Складання технічного завдання
		1.2.2	Узгодження технічного завдання із зацікавленими сторонами
		1.2.3	Складання плану та розрахунок розробки
2 Технічна пропозиція	2.1 Аналіз технічного завдання та техніко-економічне обґрунтування проекту	2.1.1	Доведення техніко-економічного обґрунтування
		2.1.2	Аналіз технічного завдання та визначення пріоритетних аспектів розробки
		2.1.3	Доведення та уточнення загального обсягу робіт, строків виконання та витрат
3 Теоретична розробка	3.1 Теоретичне вивчення задачі	3.1.1	Дослідження технічних особливостей інформаційної системи
		3.1.2	Визначення переліку технологій, які використовуватимуться при розробці та мови програмування
		3.1.3	Визначення форматів даних та запитів і їх узгодження із розробниками проекту
		3.1.4	Розробка алгоритмів роботи програми на високому рівні
		3.1.5	Розробка структури систем забезпечення та схеми взаємодії її компонент
4 Практична реалізація	4.1 Реалізація окремих модулів ПЗ	4.1.1	Розробка алгоритмів роботи програми на низькому рівні
		4.1.2	Розробка окремих класів та компонування їх у модулі
		4.1.3	Розробка графічного користувацького інтерфейсу для системи
	4.2 Відладка ПЗ	4.2.1	Автономна відладка окремих модулів системи

		4.2.2	Комплексна відладка ПЗ системи

Продовження таблиці 3.1

Найменування		Вид роботи	
	4.3 Розробка субмодуля	4.3.1	Розробка структурної схеми субмодуля
		4.3.2	Розробка функціональної схеми субмодуля
		4.3.3	Розробка алгоритму функціювання субмодуля
		4.3.4	Обґрунтування вибору елементного базису та розробка схеми
5 Доробка всього комплексу	5.1 Тестування всієї системи	5.1.1	Тестування системи у реальних умовах
		5.1.2	Доробка систем із урахуванням результатів тестування
	5.2 Підготовка документації по системі	5.2.1	Підготовка звіту про розробку системи
		5.2.2	Підготовка технічної документації
		5.2.3	Розробка довідкової системи, допоміжної і супроводжувальної документації, запис CD/DVD диска
6 Заключна стадія	6.1 Ознайомлення зацікавлених осіб з проектом	6.1.1	Підготовка презентації
		6.1.2	Демонстрація системи

Головна мета планування процесу розробки продукту — це визначення ресурсів які будуть необхідні на всіх етапах розробки програмного забезпечення (далі ПЗ).

Традиційний процедурний підхід для розробки ПЗ в основі якого лежать алгоритми, процедури та функції передбачає розробку ПЗ як монолітного композиту, що в подальшому вимагає великих витрат на супровід та модернізацію проекту.

Об'єктно-орієнтований підхід, який ґрунтується на основі об'єктів певних класів, котрі описують певну область, описують методи та володіють певними атрибутами, орієнтовані на варіанти використання та покроковий, покомпонентний процес розробки.

Підготовча стадія, технічна пропозиція і заключна стадія при розробці з об’єктно-орієнтованим підходом залишаються незмінними. Стадії теоретичної розробки, практичної реалізації та модернізація всього комплексу програмного продукту описані з точки зору специфіки об'єктно-орієнтованого підходу із

врахуванням життєвого циклу розробки ПЗ і представлені із уточненням фахівців залучених у кожний робочий процес розробки (таблиця 3.2) [8].

При створенні ПЗ за об'єктно-орієнтованим підходом необхідно залучити більшу кількість фахівців, більшу увагу приділяти покроковій компонентній розробці, що може збільшити робочий час і витрати. Однак цей підхід значно скорочує витрати на подальший супровід і модернізацію, що в загальному призведе до зменшення витрат усього проекту.

Таблиця 3.2 — Робочі процеси життєвого циклу розробки ПЗ

№	Робочі процеси	Діяльності	Співробітники	Артефакти
1	“Визначення вимог”	Ідентифікація актантів (А) і варіантів використання (ВВ), Модель ВВ Описи (формалізація) ВВ і сценаріїв реалізації, Визначення пріоритетності ВВ Створення прототипів інтерфейсів користувача (ІК)	Системний аналітик Специфікатор ВВ	Модель ВВ Актанти Глосарій ВВ Прототип ІК
2	“Проектування”	Проектування архітектури, Визначення вузлів та мережевих конфігурацій Визначення систем та їх інтерфейсів Визначення підсистем та зв’язків між ними Визначення інтерфейсів підсистем Визначення архітектурно значущих класів та класів проектування,Визначення загальних механізмів проектування Проектування ВВ Проектування класів Визначення вимог до реалізації програмного продукту	Архітектор Інженер з ВВ	Модель проектування Модель розміщення Опис архітектури Проекти реквізитів ВВ Класи проектування

Продовження таблиці 3.2

№	Робочі процеси	Діяльності	Співробітники	Артефакти
3	“Реалізація”	Модель реалізації, Компоненти, Інтерфейси компонентів, Опис архітектури План збирання системи, Реалізація архітектури, Ітеративна реалізація класів Проектування з генерацією програмного коду, Збірка системи, Планування білдів Реалізація білдів і системи в цілому	Системний інтегратор Інженер-програміст	Модель реа-лізації Опис архітекту-ри План збірки Компоне-нти Підсист-еми реалізації Інтерфейси
4	“Тестування”	Модель тестування Тестові приклади Процедура тестування Тестові компоненти План і оцінка тестування Розробка тестів Тестування цілісності Тестування системи Оцінка результатів тестування	Інженер з тестування Системний тестер	Модель тестування Тестові приклади План тестування Оцінюв-ання тестів

Терміни розробки залежать від замовника, від наданої необхідності інформації (зразків довідників, документів), рівня та порядку оплати та інших умов від замовлення, і можуть становити від 2 тижнів до 6 місяців і більше (за даними ІТ компаній)

Вартість складання технічного завдання зазвичай складає до 10% від планувальної вартості розробки і становить від 200 до 5000 гривень (за даними ІТ компаній). Роботу зі складанням технічного завдання веде керівник проекту разом із програмістами та проконсультувавшись із замовником.

Для визначення загальної тривалості проведення робіт дані витрат часу для окремих операцій техпроцесу було зведено в таблицю (таблиця 3.3).

Таблиця 3.3 — Тривалість та трудомісткість розробки та реалізації програмного проекту

Найменування роботи	Виконавець, посада, спеціальність	К-сть виконав-ців	Тривалість виконання роботи, дні
Дослідження предметної області, вибір середовища для реалізації проекту, вивчення та аналіз аналогічних розробок	Керівник проекту Програміст	2	1
Економічне обгрунтування доцільності виконання проекту, складання ТЗ, узгодження ТЗ із зацікавленими сторонами, складання плану та розрахунок розробки	Керівник проекту Програміст	2	1
Доведення техніко-економічного обґрунтування, аналіз технічного завдання та визначення пріоритетних аспектів розробки, доведення та уточнення загального обсягу робіт, строків виконання та витрат	Керівник проекту Програміст	2	1
Теоретична розробка процедурний підхід	Програміст	2	2
Теоретична розробка об'єктно-орієнтований підхід	Системний аналітик, специфікатор ВВ, архітектор, інженер з ВВ	3	3
Практична реалізація процедурного підходу	Програміст	2	4
Практична реалізація об'єктно-орієнтованого підходу	Системний інтегратор, інженер-програміст	3	5
Тестування програмного забезпечення	Інженер з тестування	1	4
Доробка всього комплексу програмного забезпечення	Програміст	1	3
Заключна стадія	Керівник проекту Програміст	2	1

Як для процедурного, так і для об’єктно-орієнтованого підходу, підготовча і заключна стадії та технічна пропозиція будуть виконуватися однаково, теоретична розробка і практична реалізація для об’єктно-орієнтованого підходу

вимагатимуть залучення більшої кількості ІТ-спеціалістів, що може збільшити тривалість виконання роботи.

3.2 Розрахунок витрат на реалізацію проекту та оцінка економічної ефективності

Оцінка вартості комп'ютерних програм базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації. Оподаткування заробітної плати програмістів підприємства-розробника, яким встановлено певний посадовий оклад, відбувається аналогічно оподаткуванню зарплати працівників інших галузей господарства.

Враховуючи залежність якості кінцевого програмного продукту від кваліфікації програмістів, розробники часто йдуть на додаткові заходи їх стимулювання,які найчастіше втілюються у:

- в певній системі преміювання (за виконання графіку розробки чи його дострокове виконання, оптимізація етапів розробки тощо). Премії оподатковуються аналогічно витратам на оплату праці;

- авторській винагороді за кожну продану одиницю програми. Сума не обкладається внесками до пенсійного фонду та фондів соціального страхування, а використання авторських прав оформлюється відповідними договорами (ліцензійний авторський договір).

Разом з тим до створення ПЗ можуть залучатися також позаштатні програмісти як зареєстровані, так і не зареєстровані підприємцями. В обох випадках співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Щодо оподаткування виплат за договором підряду, то все залежить від того, чи зареєстрований виконавець підприємцем:

- якщо виконавець за договором підряду не зареєстрований фізичною особою-підприємцем, то виплати за таким договором оподатковуються ПДФО за ставкою 15 %, також нараховуються (у розмірі 33,2 %) та утримуються (у розмірі 2 %) внески до Пенсійного фонду (п. 1 та п. 4 ст. 4 Закону № 400). Оподаткування провадиться у межах максимальної величини, що дорівнює 15 розмірам прожиткового мінімуму, встановленого для працездатних осіб;

- якщо ж виконавець за договором підряду є фізичною особою-підприємцем, яка сплачує єдиний податок чи знаходиться на загальній системі оподаткування, виплати, що здійснюються на його користь в рамках договору підряду, не оподатковуватимуться ПДФО.

Вважатимемо, що усі програмісти працюють у штаті підприємства-розробника і їм встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість і основна заробітна плата кожного учасника технічного процесу представлено у таблиця 3.4.

Таблиця 3.4 — Сума часткових заробітних плат

	Місячний оклад, грн.	Денна зар. плата, грн	Трудомісткість, людино-дні		Основна заробітна плата, грн.	
			Процедурний підхід	Об'єктно-орієнтований підхід	Процедурний підхід	Об'єктно-орієнтований підхід
Керівник проекту	17800	712	4	4	2848	2848
Інженер-програміст	14600	584	17	33	9928	19272
Інженер-тестувальник	9800	392	4	4	1568	1568
Дизайнер	7500	300	3	3	900	900
Всього			28 днів	44 днів	15244 грн.	24588 грн.

Визначимо витрати на основну заробітну плату. Вони складають суму заробітних плат за кожну із робіт. Витрати на основну зарплату для процедурного та об’єкт-орієнтованого підходів:

Додаткова заробітна плата обчислюється за формулою $ЗП_{дод}=ЗП_{осн}\cdot 0.2$.

Обчислимо витрати на додаткову зарплату для двох підходів:

$ЗП_{осн(п)}= 2848 + 9928 + 1568 + 900 = 15244$ (грн.);

$Зп_{осн(о)}= 2848 + 19272 + 1568 + 900 = 24588$ (грн.);

Це сума часткових заробітних плат за кожну роботу, і вона приведена в таблиці 3.4.

Додаткова заробітна плата обчислюється за формулою:

$ЗП_{дод}= 0,2 \cdot ЗП_{осн}$.

$Зп_{дод(п)}= 15244 \cdot 0,2 = 3050$ (грн.);

$Зп_{дод(о)}= 24588 \cdot 0,2 = 4918$ (грн.);

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$ФЗП = Зп_{осн}+ЗП_{дод}$.

$ФЗП_{(п)} = 15244 + 3050 = 18294$ (грн.);

$ФЗП_{(о)} = 24588 + 4918 = 29506$ (грн.);

З цієї суми утримуються обов’язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, який складає 15% від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов’язкове соціальне страхування та податкової соціальної пільги.

Таким чином обов'язкові відрахування на заробітну плату для працівників складають:

Утримання єдиного соціального внеску:

$Відр_{ЄСВ1} = 0{,}036 \cdot ФЗП = 0{,}036 \cdot 18294 = 659$ (грн.);

$Відр_{ЄСВ2} = 0{,}036 \cdot ФЗП = 0{,}036 \cdot 29506 = 1062$ (грн.);

Податок на доходи фізичних осіб:

$Відр_{ПДФО1} = 0{,}15 \cdot (ФЗП - Відр_{ЄСВ1}) = 0{,}15 \cdot (18294\text{-}659) = 2645{,}25$ (грн.);

$Відр_{ПДФО2} = 0{,}15 \cdot (ФЗП - Відр_{ЄСВ2}) = 0{,}15 \cdot (29506\text{-}1062) = 4266{,}6$ (грн.);

$Відр_1 = Відр_{ЄСВ1} + Відр_{ПДФО1} = 659 + 2645{,}25 = 3304{,}25$ (грн.);

$Відр_2 = Відр_{ЄСВ2} + Відр_{ПДФО2} = 1062 + 3304{,}25 = 4366{,}25$ (грн.);

Законом № 1621 підрозділ 10 розділу XX Перехідних положень Податковий кодекс України від 02 грудня 2010 року № 2755-VI зі змінами та доповненнями (далі – ПКУ) доповнено пунктом 16, яким тимчасово, до 1 січня 2015 року, встановлено військовий збір.

Водночас Законом України від 28 грудня 2014 року № 71-VIII «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» (далі – Закон № 71), який набрав чинності з 01.01.2015, оподаткування військовим збором подовжено до набрання чинності рішенням Верховної Ради України про завершення реформи Збройних Сил України (пункт 16 підрозділ 10 розділу XX Перехідних положень ПКУ) [35].Водночас Законом України від 28 грудня 2014 року № 71-VIII «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» (далі – Закон № 71), який набрав чинності з 01.01.2015, оподаткування військовим збором подовжено до набрання чинності рішенням Верховної Ради України про завершення реформи Збройних Сил України (пункт 16 підрозділ 10 розділу XX Перехідних положень ПКУ) [9].

Платниками збору в ромірі 1,5% від заробітної плати є особи, визначені пунктом 162.1 статті 162 цього ПКУ (підпункт 1.1 пункт 16 підрозділу 10 ПКУ) [10].

Військовий збір з фізичних осіб:

$ВЗФО = 0{,}015 \cdot ФЗП$;

$ВЗФО_1 = 0{,}015 \cdot 18294 = 274{,}41$ (грн.);

$ВЗФО_2 = 0{,}015 \cdot 29506 = 442{,}59$ (грн.);

Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3%, для підприємців розмір єдиного внеску залежно від класу професійного ризику виробництва становить від 36,76 % до 49,70 %. Зокрема, видання програмного забезпечення – 36,77%) [11].

Нарахування на фонд оплати праці (ФОП):

$ФОП_{ЄСВ} = 0{,}3677 \cdot ФЗП$

$ФОП_{ЄСВ1} = 0{,}3677 \cdot 18294 = 6726{,}70$ (грн.)

$ФОП_{ЄСВ2} = 0{,}3677 \cdot 29506 = 10849{,}35$ (грн.)

Всього витрат:

$В_{ЗП1} = ЗП_1 + ФОП_{ЄСВ1} = 18294 + 6726{,}70 = 25020{,}7$ (грн.);

$В_{ЗП2} = ЗП_2 + ФОП_{ЄСВ2} = 29506 + 10849{,}35 = 40355{,}35$ (грн.);

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни;

$$M_{Bi} = q_i \cdot p_i$$

де q_i — кількість витраченого матеріалу i-го класу; p_i — ціна матеріалу i-го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{B\,i}$$

Проведені розрахунки занесено в таблицю 3.5:

Таблиця 3.5 — Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Фактично витрачено матеріалів	Ціна 1-ці, грн.	Загальна сума витрат, грн.
1	SCRUM-дошка	шт	1	500,00	500,00
2	Папір для друку	листів	120	0,25	30,00
3	Чорнила для принтера	шт	1	65,00	65,00
4	Маркери	шт	4	14,00	56,00
Всього					651,00

Згідно постанов Національної комісії, що здійснює державне регулювання у сфері енергетики (згідно Постанов Національної комісії, що здійснює державне регулювання у сфері енергетики (НКРЕ) від 22.01.2001р.№47 зі змінами і доповненнями, від 06.12.2002р. № 1358, від 22.10.2004р. № 1030, від 28.04.2016р. № 755 на підставі Закону України від 03.12.1999р № 1274-XIV “Про внесення змін до Закону України “Податкового кодексу України”, згідно Закону України “Про міський електричний транспорт” від 29 червня 2004 року № 1914 – IY; також затвердженого постановою Кабінету Міністрів України від 01.06.2011р. № 869, Порядку розрахунку роздрібних тарифів на електричну енергію, тарифів на розподіл електричної енергії (передачу електричної енергії місцевими (локальними) електромережами), тарифів на постачання

електричної енергії за регульованим тарифом, затвердженого постановою Національної комісії, що здійснює державне регулювання в сфері енергетики та комунальних послуг (НКРЕКП) "Про ринкове формування роздрібних тарифів на електричну енергію, що відпускається для кожного класу споживачів, крім населення, на території України", ця сума становить 2,50 грн/кВт.г. [12].

Витрати на електроенергію одиниці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії. S = 2,50грн/кВт.г.

$З_{е1} = 1,5 \cdot 224 \cdot 2,5 = 840$ (грн.);

$З_{е2} = 1,5 \cdot 352 \cdot 2,5 = 1320$ (грн.);

Розрахунок суми амортизаційних відрахувань.

Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Амортизація за час (період) використання обладнання (комп’ютера) складає долю затрат, які припадають на дослідницьку роботу (написання програми), і визначається:

$$A = \frac{С_Б * N_A * T_{ФАК}}{Т_{год}}$$

де $С_Б$ — балансовна вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, $Т_{год}$ — річний робочий фонд часу, год.; $Т_{ФАК}$ — фактичний час роботи обладнання по написанню програми, год.

Таблиця 3.6 — Перелік обладнання необхідного для розробки

Найменування	Кількість, шт.	Ціна за одиницю продукту, грн.	Сума, грн.
Комп’ютер	4	15775	63100
Принтер	1	2000	2000
Хостинг	1	995	995
Доменне ім'я	1	366	366
Всього			66461

$$A_1 = \frac{66461 * 0{,}6 * 224}{2016} = 4430{,}75(\text{грн.}) A_2 = \frac{66461 * 0{,}6 * 352}{2016} = 6962{,}60(\text{грн.})$$

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$НВ = 0{,}5 \cdot ЗП_{осн}.$$

$$НВ_п = 0{,}5 \cdot 18294 = 9147 \text{ грн.};$$

$$НВ_о = 0{,}5 \cdot 29506 = 14753 \text{ грн.}$$

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво

$$Св = В_{зп} + З_{мв} + З_е + А + НВ;$$

$$Св_1 = 25020{,}7 + 651 + 840 + 4430{,}75 + 9147 = 40089{,}45 \text{ (грн.)};$$

$Св_2 = 40355,35 + 651 + 1320 + 6962,60 + 14753 = 64041,95$ (грн.).

Приймемо прибуток на рівні 27%. Для нових інноваційних продуктів, що користуються високим попитом на ринку.

Отже, вартість розробленого програмного забезпечення:

$В_1 = Св_1 + 0,27 \cdot Св_1 = 40089,45 + 0,27 \cdot 40089,45 = 50913,60$ (грн.);

$В_2 = Св_2 + 0,27 \cdot Св_2 = 64041,95 + 0,27 \cdot 64041,95 = 81333,27$ (грн.)

Економічна ефективність (*Е*) полягає у відношенні результату виробництва до затрачених ресурсів:

$$Е = \frac{П}{Св}$$

де *П* — прибуток, (*П = В — Св*); Св — собівартість.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку.

$Е_п = (50913,60 - 40089,45) / 40089,45 = 0,27;$

$Е_о = (81333,27 - 64041,95) / 64041,95 = 0,27.$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Т_{ок}$):

$$Т_{ок} = \frac{1}{Е}$$

У нашому випадку $Т_{ок1} = Т_{ок2} = 1 / 0,27 = 3,7$ років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Загальна вартість пропонованих робіт по розробці програмного продукту становить 18294 грн. для першого варіанту та 29506 грн. для другого. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,27, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3,7 року.

3.3 Визначення витрат на супровід та модернізацію програмного продукту та уточнений аналіз ефективності вкладених інвестицій

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за процедурним методом 55% від початкових витрат, а за об'єктно-орієнтованим – 25%.

Собівартість модернізації:

$СвМ_п = 0,55 \cdot Св_п = 0,55 \cdot 40089,45 = 22049,20$ грн.,

$СвМ_о = 0,25 \cdot Св_о = 0,25 \cdot 64041,95 = 16010,48$ грн.

Вартість модернізації для споживача:

$М_п = 0,5 \cdot В_п = 0,55 \cdot 50913,60 = 28002,50$ грн.,

$М_о = 0,25 \cdot В_о = 0,25 \cdot 81333,27 = 20333,32$ грн.

Таким чином, уже після першої модернізації, загальні витрати на створення і супровід ПЗ для виробника за об'єктно-орієнтованим методом менші, ніж за процедурним, навіть якщо його собівартість є дещо дорожчою.

$ЗВ_{п\ (вир)} = 40089,45 + 22049,20 = 62138,65$ грн.,

$ЗВ_{о\ (вир)} = 64041,95 + 16010,48 = 80052,43$ грн.

Як і для споживача:

$ЗВ_п = 50913,60 + 28002,50 = 78916,1$ грн.,

$ЗВ_о = 81333,27 + 20333,32 = 101666,60$ грн.

Річна економія витрат за всіма можливими напрямами і додатковими витратами, пов'язаними з супроводом і тільки одноразовою модернізацією (у розрахунку на одиницю продукції) при об’єктно-орієнтованому методі порівняно із процедурним:

$\Delta C_{(вир)} = ЗВ_{1(вир)} - ЗВ_{2(вир)} = 68091,95 - 80052,43 = -11960,48$ грн,

$\Delta C = ЗВ_1 - ЗВ_2 = 78916,1 - 101666,60 = -22750,5$ грн,

Визначення ЧПД відбувається за формулою:

$$ЧПД = \sum_{i=1}^{i} ГП_i \, a_{ТВi} - \sum_{i=1}^{i} ІК_i \, a_{ТВi};$$

де $Гп_i$ — грошовий потік і-го розрахункового року;

$ІК_i$ – сума інвестицій і-го розрахункового року;

$\alpha_{ТВi}$ – коефіцієнт дисконтування (коефіцієнт приведення інвестицій і грошового потоку до теперішньої вартості).

Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$a = \frac{1}{(1+i)n}$$

де i — ставка дисконтування або норма дисконту, $i = 0,27$;

n — час або кількість періодів (років), протягом якого планується отримувати дохід.

$$a_0 = 1, a_1 = \frac{1}{1+0,27} = 0,79$$

Вважатимемо, що обидва програмних продукти однаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми однаково вплинуть на розмір його додаткових доходів на вкладений капітал, Тому приймемо цю величину за постійну, а порівняння дохідності двох проектів проведемо тільки за витратами.

Усі результати розрахунків, приведених в розділі зведені у таблицю 3.7

Таблиця 3.7 — Результати розрахунків

№ п.п.	Назва	Процедурний підхід	Об’єкто-орієнтова-ний підхід
1	Зарплата основна, грн	18294	29506
2	Зарплата додаткова, грн	3050	4918
3	Фонд заробітньої плати (1+2)	21344	34424
4	Обов’язкові відрахування на заробітну плату (4.1+4.2), грн	3304,25	4366,25
4.1	Утримання єдиного соціального внеску (3,6%), грн	659	1062
4.2	Податок на доходи фізичних осіб (15%), грн	2645,25	3304,25
5	Військовий збір (1.5%), грн	264,5	3304,25
6	Відрахування на ФОП (36,77%), грн	6726,70	10849,35
7	Разом на оплату праці (3+6), грн	28070,7	45273,35
8	Матеріальні витрати, грн	651	651
9	Електроенергія, грн	840	1009.2
10	Амортизація, грн	4430,75	6962,60
11	Накладні витрати, грн	9147	14753
12	Разом на ін. витрати (8+9+10+11)	15177,3	23375,8
13	Собівартість, грн	40089,45	64041,95
14	Прибуток, грн	10824,15	17291,33
15	Вартість розробленого ПЗ	50913,60	81333,27
16	Економічна ефективність	0,27	0,27
17	Термін окупності, років	3,7	3,7
18	Собівартість модернізації, грн	22049,20	16010,48
19	Супровід і модернізація, грн	28002,50	20333,32

Загальна вартість пропонованих робіт по розробці програмного продукту становить 78916,10 грн. для першого варіанту та 101666,60 грн. для другого. Оскільки ефективність для обидвох проектів відповідно до встановленого рівня прибутку становить 0,27, що є високим показником, то проводити дані роботи варто і вкладені кошти окупляться за 3,7 року.

При використанні процедурного підходу зменшується кількість працівників, які залучаються у проект, та зменшуються витрати на реалізацію проекту, але для підтримки проекту і його подальшої модернізації все ж в майбутньому потрібні набагато більші витрати. Для об'єктно-орієнтованого підходу потрібна більша кількість працівників, часу і коштів, але на модернізацію і підтримку в загальному потрібно менше ресурсів, і у висновку програма виконана по функціональній парадигмі стає дешевшою для споживача, і приносить економію ресурсів для розробників.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКИ ЖИТТЄДІЯЛЬНОСТІ

4.1 Охорона праці

Тема дипломної роботи передбачає створення системи симуляції екосистеми зі штучним інтелектом для персональних комп'ютерів з операційними системами типу Windows, Linux та MacOS. Розробник системи в процесі розробки інтеграції експлуатує персональні електронно-обчислювані машини. Для розробки системи потрібне офісне приміщення, в якому будуть розміщуватись комп’ютеризовані робочі місця, тому потрібно спланувати приміщення відповідно до державних нормативно-правових актів та законів.

При роботі з комп'ютером користувач піддається дії ряду небезпечних і шкідливих виробничих факторів: електромагнітні поля (діапазон радіочастот: ВЧ, УВЧ і СВЧ), рентгенівське та ультрафіолетове випромінювання, шум і вібрація, запиленість робочої зони, накопичення статичної електрики і інші.

Для зменшення впливу негативних ефектів, потрібно правильно розмістити обладнання та робочі місця в офісі. Список нормативно-правових актів, що регулюють дане питання, є досить широким. Так, обов’язки роботодавця щодо забезпечення працівникам комфортних та безпечних умов для здійснення роботи, а також права працівників на такі умови передбачено частиною 2 ст. 2 та ч. 1 ст. 21 КЗпП, а також ст. 13 Закону України «Про охорону праці».

В Законі України «Про охорону праці» вказано основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні [13].

Для оцінки відповідності робочого приміщення відповідно до актів потрібно заміряти його розміри. Так в приміщенні розміром 12 м^2 (4х3 м) і висотою 3,4 метра можна розмістити не більше двох комп’ютеризованих робочих місць (не менше 6 м^2 на одне робоче місце).

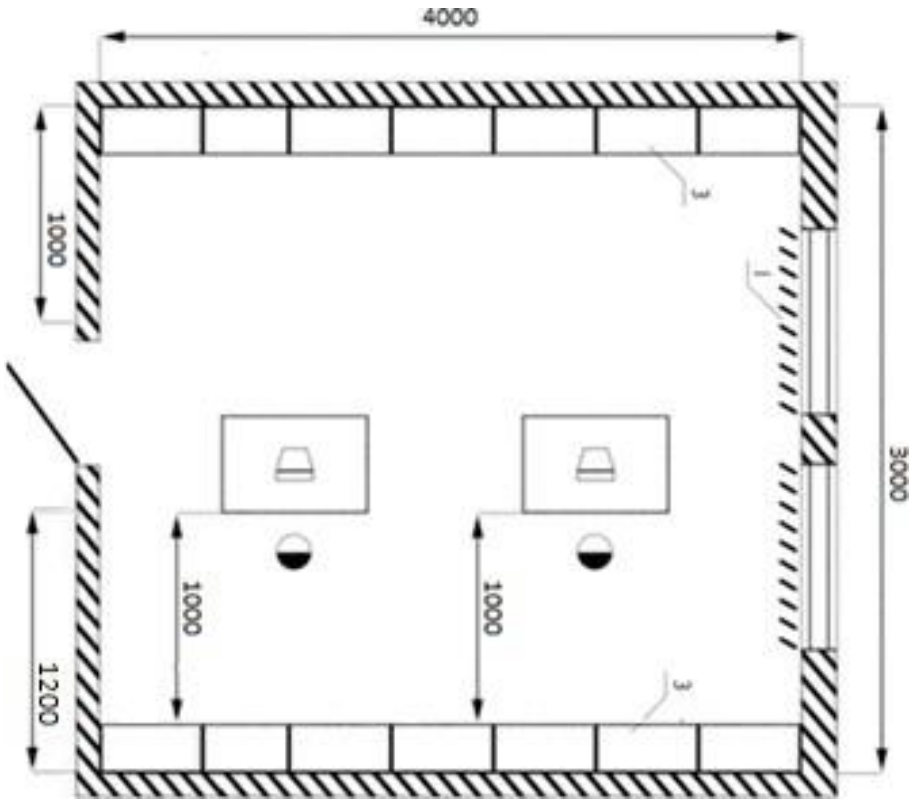
Окрім площі на одне робоче місце в нормативно правових актах вказано потрібний об’єм приміщення – 20 м^3. Об'єм приміщення в даному випадку становить 40,8 м^3 а об'єм, що припадає на одне робоче місце – 20,4 м^3. Таким чином норматив щодо об'єму приміщення на одне робоче місце виконується.

Планування розміщення комп'ютеризованих робочих місць у приміщенні проводимо з урахуванням таких вимог:

- робочі місця розміщуються на відстані не менше 1 м від стіни зі світловими прорізами (вікнами);
- відстань між бічними поверхнями має бути не меншою за 1,2 м;
- відстань між тильною поверхнею одного комп’ютеризованого робочого місця та екраном іншого не повинна бути меншою за 2,5 м;
- прохід між рядами робочих місць має бути не меншим за 1м.

Найкраще розмістити комп'ютеризовані робочі місця рядами вздовж стіни з вікнами. Це дасть змогу виключити дзеркальне відбиття на екрані джерел природного світла (вікон) та потрапляння останніх у поле зору операторів, що погіршує їх зорову роботу.

Наведемо план офісного приміщення для розробки програмного забезпечення з комп'ютеризованими робочими місцями (рис. 4.1):



1 – комп'ютеризоване робоче місце;

2 – сонце захисні жалюзі;

3 – шафи для зберігання;

Рисунок 4.1 – План приміщення з комп'ютеризованими робочими місцями.

Параметри, за якими оцінюється мікроклімат, встановлюється відповідно до пори роки і категорії роботи. Для теплого періоду, температура повинна бути в проміжку +23 °С - +25 °С, вологість 40 - 60 % а швидкість повітря ≤ 0.1 м/с. Для холодного періоду температура повітря повинна бути в проміжку +22 °С - +24 °С. Всі показники задовольняють вимогам для робіт категорії легка 1а і є задовільними для здоров’я людини.

Для визначення потрібної кількості світильників, які повинні забезпечити нормований рівень освітленості, визначається світловий потік, що падає на робочу поверхню.

Приміщення, в яких встановлені персональні комп’ютери, повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28-2018.

Природне освітлення має здійснюватися через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 5%. Розраховується КПО за методикою, викладеною в ДБН В.2.5-28-2018.

Штучне освітлення в приміщеннях з робочими місцями має здійснюватися системою загального рівномірного освітлення. У разі переважної роботи з документами, допускається застосування системи комбінованого освітлення (крім системи загального освітлення додатково встановлюються світильники місцевого освітлення). Зазначення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300- 500лк. Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцеве освітлення.

Приміщення, має 2 вікна з лінійними розмірами: ширина – 1 м, висота – 1,8 м, відповідно площа кожного вікна – 1,8 м^2. Вікна мають регульовані пристрої для відкривання та обладнані жалюзі з можливістю захисту працюючих від прямого попадання сонячних променів і регулювання рівня освітленості в приміщенні. Відблискування поверхонь обмежується за рахунок правильного вибору світильників та розташування робочих місць відносно джерел освітлення. Яскравість відблисків на сучасних моніторах не перевищує 35 кд/м^2. В досліджуваному приміщенні використовується система загального рівномірного штучного освітлення. Мається 2 ряди світильників Л201Б 2х400.3, у кожному з яких знаходиться по чотири лампи типу ЛБ-40.

Споживачі електроенергії: 2 ПЕОМ, 2 дисплея, 4 світильника, 1 кондиціонер. Кожне робоче місце обладнане 4-ма розетками по 220 В. Заземлені конструкції захищені діелектричними сітками від випадкового дотику. Усе електроустаткування має апаратуру захисту від струму короткого замикання.

Отже, можна зробити висновок, що кабінет задовольняє вимогам електробезпеки у приміщенні, в якому встановлені ЕОМ, зображені в НПАОП 0.00-7.15-18.

У приміщенні маються внутрішні джерела постійного шуму – вентилятори блоків ЕОМ, дисководи. Фактичний обмірюваний рівень шуму в робочій зоні задовольняє нормативному рівню шуму (не повинний перевищувати 50 дБА згідно з ДСН 3.3.6.037- 99).

Для забезпечення захисту і досягнення нормованих рівнів комп’ютерних випромінювань необхідно застосування приекранних фільтрів, локальних світлофільтрів (засобів індивідуального захисту очей) та інших засобів захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат.

Приміщення відноситься до зони П-ІІа згідно з НПАОП 0.00-7.15-18 і до категорії пожежної небезпеки В. Горючі рідини, пил та волокна у приміщенні не використовуються і не виділяються.

Ймовірними причинами виникнення пожежі можуть бути несправність електрообладнання, короткі замикання внаслідок виходу з ладу електроустаткування, порушення правил протипожежної безпеки тощо.

4.2 Електробезпека користувачів ПК

Такий елемент як персональний комп'ютер вже давно став важливим елементом оточення більшості людей. Вони використовуються для розваг, для офісної роботи, для спілкування і багато чого іншого. Тому безпека користувачів які користуються персональним комп'ютером є важливим питанням.

Приміщення із робочими місцями користувачів комп’ютерів для забезпечення електробезпеки обладнання, а також для захисту від ураження електричним струмом самих користувачів ПК повинні мати достатні технічні засоби захисту.

Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, перейти на негорючу ізоляцію.

Лінія електромережі для живлення ЕОМ, периферійних пристроїв ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ виконується як окрема групова три-провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занурення) електроприймачів і прокладається від стійки групового розподільчого щита, розподільчого пункту до розеток живлення [17].

У приміщенні, де одночасно експлуатується або обслуговується більше п'яти персональних ЕОМ, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

Неприпустимим є підключення ЕОМ, периферійних пристроїв ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ до звичайної двопровідної електромережі, в тому числі — з використанням перехідних пристроїв.

При розташуванні в приміщенні за його периметром до 5 персональних ЕОМ, використанні три-провідникового захищеного проводу або кабелю в оболонці з негорючого або важкогорючого матеріалу дозволяється прокладання їх без металевих труб та гнучких металевих рукавів.

Електробезпека будівель та приміщень, де розміщені робочі місця операторів (користувачів) ПК, повинна відповідати вимогам Правил безпечної експлуатації електроустановок споживачів, затверджених наказом Держнаглядохоронпраці від 9 січня 1998 р. № 4 (далі — НПАОП 40.1-1.21-98) (п.

1.4 розділу III Правил № 65), але ці правила не поширюється на безпосередньо комп'ютерну техніку, оскільки комп'ютер не є електроустановкою, а є електротехнічним пристроєм, який розміщується в приміщеннях без підвищеної небезпеки, поза межами вибухо- та пожежонебезпечних зон.

На підприємстві інструктаж, навчання й перевірка знань з питань охорони праці та пожежної безпеки здійснюються відповідно до вимог Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнаглядохоронпраці України від 26 січня 2005 р. № 15 (НПАОП 0.00-4.12-05) та Типового положення про інструктажі, спеціальне навчання та перевірку знань з питань пожежної безпеки на підприємствах, в установах та організаціях України, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 29 вересня 2003 р. № 368 (НАПБ Б.02.005-2003). Працівники, які в установленому цими нормативним документами не пройшли навчання, інструктаж та перевірку знань, не повинні допускатись до роботи (пункти 1.4–1.6 розділу I Правил № 65).

Металеві труби та гнучкі металеві рукави повинні бути заземлені. Заземлення повинно відповідати вимогам ДНАОП 0.00-1.21-98 "Правила безпечної експлуатації електроустановок споживачів". Заземлені конструкції, що знаходяться у приміщеннях (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном тощо), мають бути надійно захищені діелектричними щитками або сітками від випадкового дотику.

Конструкція знімної підлоги повинна бути такою, щоб забезпечувались:

— вільний доступ, до кабельних комунікацій під час обслуговування;

— стійкість до горизонтальних зусиль при частково знятих плитах;

— вирівнювання поверхні підлоги за допомогою регулювальних опорних елементів;

— взаємозамінюваність плит.

Є неприпустимими:

— експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості за час експлуатації, ізоляцією; залишення під напругою кабелів та проводів з неізольованими провідниками;

— застосування саморобних продовжувачів, які не відповідають вимогам ПВЕ до переносних електропроводок;

— застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання;

— користування пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має сліди затемнення або випинання;

— підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною та іншими горючими матеріалами, експлуатація їх зі знятими ковпаками (розсіювачами);

— використання електроапаратури та приладів в умовах, що не відповідають вказівкам (рекомендаціям) підприємств-виготовлювачів.

4.3 Висновок до розділу

У даному розділі були розглянуті умови праці для офісного приміщення, у якому програмістами розробляється та експлуатується програмний продукт. Було проведено аналіз шкідливих та небезпечних виробничих чинників, наявних у даному приміщенні. Розміри приміщення та параметри робочих місць відповідають нормам чинного законодавства з охорони праці. Усі параметри приміщення відповідають необхідним нормативам.

ВИСНОВОК

В результаті виконання магістерської роботи була розроблена система симуляції екосистеми для операційних систем типу Windows, Linux та MacOS.

У цій магістерській роботі було досліджено проблему розробки штучного інтелекту, який би дав змогу досягнути дійсно хороших рівнів інтелектуальності штучного інтелекту який би дав змогу розробити якісне віртуальне середовище для комп'ютерних ігор.

Опираючись на вміння програмування у цій магістерській роботі були використані певні технології створення автономних штучних істот наділених штучним інтелектом.

Такі методи застосування технологій штучного інтелекту дадуть змогу спростити розробку штучного інтелекту для віртуального ігрового середовища. Під час розробки магістерської роботи склалося повне розуміння того, що інтелектуальні форми поведінки повинні бути втілені в дійсні створіння, і на основі цього розуміння зародився підхід, який передбачає застосування аніматів. Із цього був виведений висновок, що важливо, перш за все, займатися розробкою засобів штучного інтелекту, а не простим програмуванням і оптимізацією класичних алгоритмів. Люди, які посвятили своє життя створенню ігрових засобів штучного інтелекту повинні надавати увагу всім напрямкам своєї діяльності.

Цей підхід (умовно названий розробкою сучасних ігрових середовищ штучного інтелекту) показує, як втілити на практиці новітні теоретичні досягнення в області ігрових середовищ штучного інтелекту в поєднанні з цим підходом. По мірі того, як перед розробниками ігрових середовищ штучного інтелекту встають все нові і нові задачі (в зв'язку з зі зростаючою необхідністю в забезпеченні інтелектуальності, ефективності і реалістичності), стає все більш важливим запозичення досягнень інших суміжних наукових дисциплін. Підхід

до створення ігрових середовищ штучного інтелекту, який був описаний в даній магістерській роботі, може бути розглянутий в використанні наступних трьох областях досліджень:

- Новітні дослідження штучного інтелекту, котрі спрямовані на вивчення втілених систем, які знаходяться в достатньо реалістичних світах (таких як ігрові персонажі);
- Роботи по створенню активізованих архітектур, які забезпечують управління компонентами, які розподілені в середовищі, які негайно реагуватимуть на зміну обстановки в віртуальному середовищі;
- Методи навчання, які використовуються для моделювання адаптивних форм поведінки, які дозволяють штучним створінням придбати певні інтелектуальні властивості.

Дана магістерська робота показує, що сучасні засоби штучного інтелекту можуть служити додатком до сучасних ідей комп'ютерних іграх. Такі ідеї можуть застосовуватися до певних напрямків втілення штучного інтелекту, це буде сприяти підвищенню правдоподібності та дасть змогу спростити розробку цих систем.

Найбільш адекватно реагувати на безпосередньо оточувальне середовище спроможні саме втілені системи. Використовуючи засоби сприйняття такі системи збирають тільки необхідну їм локальну інформацію, по аналогії з тим, як люди та тварини взаємодіють зі своїм середовищем. Завдяки цьому з'являється можливість автоматично відфільтрувати менш релевантну інформацію, що сприяє спрощенню задачі розробки інтелектуальних форм поведінки, як в теорії, так і на практиці. Процес обчислення дій на основі сприйнятої ситуації робиться майже прямолінійним. По цій причині підвищується ефективність застосування методик навчання, які супроводжуються зменшенням потенціальних проблем і підвищенню продуктивності праці розробників.

Як і живі істоти, аниманти повинні реагувати на стимули, які поступають з зовнішнього середовища. З'являються ситуації, які відносяться до різних типів, тому частіше за все доцільно використовувати для симуляції реакції різні компоненти засобів штучного інтелекту. Ці незалежні компоненти, окремо взяті, можуть бути організовані в виді активізованої архітектури, що дозволяє розширити їх можливості. Кількість можливих типів архітектур вельми велике, але для управління інтелектуальною поведінкою в більшій степені підходять активізуємі, завдяки їх надійності і простоті, тому такі архітектури часто служать в якості фундаменту для більш складних методик.

Втілення окремих компонентів, які являють собою мозок, від компонентів, які являються тілом. Такий метод застосований до ігрової машини означає, що повинні бути проведені явні відмінності між засобами штучного інтелекту та ігровою логікою, і навіть повинні бути втілені в окремо організовуваному моделюванні самого світу гри, це надзвичайно сприяє спрощенню проектованих рішень. Що стосується розробки, то створення анімантів зазвичай втілюється інкрементно, і при цьому кожний раз проводиться перевірка системи з допомогою експериментів. Це — чудова методологія, яка забезпечує реалістичність і надійність. Керувальні архітектури по самій своїй суті являються модульними, що дає спроможність організувати роботу по принципу “розділяй і володарюй” як і в процесі реалізації, так і в процесі тестування системи.

Такі підходи до створення штучного інтелекту значно спростять в майбутньому створення віртуальних середовищ зі штучним інтелектом, які зможуть застосовуватись в певних цілях, як і в іграх так і в певних наукових симуляціях віртуальних середовищ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Канонічні діаграми мови UML — Студопедія [Електронний ресурс] – 2018 – Режим доступу: https://studopedia.com.ua/1_42750_kanonichni- diagrami-movi-UML.html
2. Rumbaugh K. The unified modeling language reference manual [Text] / J. Rumbaugh, I. Jacobson, G. Booch – Boston. : Addison Wesley Longman Inc., 1999 – p 228. - ISBN 0-201-30998-X.
3. Cockburn, 2001. Inside rear cover. Field "Use Case Title", 2009. Page 39.
4. ДСТУ 2844-94 Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення.
5. James Rumbaugh, Ivar Jacobson, Grady Booch (1999). The unified modeling language reference manual. Addison Wesley Longman Inc. ISBN 0-201-30998-X.
6. Unified Modeling Language Superstructure Specification, version 2.1.1. Document formal/2007-02-05, Object Management Group, February 2007.
7. Закон України «Про авторське право і суміжні права» від 23.12.1993№ 3792-XII. – [Електронний ресурс]. – Режим доступу: http://portal.rada.gov.ua/5.6.12
8. Методичні рекомендації по виконанню розділу техніко-економічного обґрунтування дипломних робіт студентами технічних спеціальностей напряму підготовки 8.05010302 «Інженерія програмного забезпечення» освітньо-кваліфікаційного рівня «Магістр» / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.
9. Закон України «Про оподаткування прибутку підприємств» від 28.12.1994 № 334/94-ВР. – [Електронний ресурс]. – Режим доступу: http://portal.rada.gov.ua/6

10. Закон України «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» від 28 грудня 2014 року № 71-VIII. – [Електронний ресурс]. – Режим доступу: http://portal.rada.gov.ua/
11. Закон України «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» від 28 грудня 2014 року № 71-VIII. – [Електронний ресурс]. – Режим доступу: http://portal.rada.gov.ua/
12. Закон України «Про міський електричний транспорт» від 29 червня 2004 року № 1914 – IY. – [Електронний ресурс]. – Режим доступу: http://portal.rada.gov.ua/
13. Закон України «Про авторське право і суміжні права» від 23.12.1993 № 3792-XII. – [Електронний ресурс]. – Режим доступу: http://portal.rada.gov.ua/5.6.12
14. Лайза Криспин, Джанет Грегори Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М. : «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series). — 1000 прим. — ISBN 978- 5-8459-1625-9.
15. Методичні вказівки до виконання магістерської роботи освітнього рівня “магістр” студентами усіх форм навчання для напряму підготовки 121 – “Інженерія програмного забезпечення” / Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид- во ТНТУ імені Івана Пулюя, 2016 – 26 с.
16. Comparing React Native to Cordova | Toptal [Електронний ресурс].– Режим доступу: https://www.toptal.com/mobile/comparing-react-native-to- cordova

17. Гандзюк М. П. Основи охорони праці: підручник. - 4-те вид. / М. П. Гандзюк, Є. П. Желібо, М. О. Халімовський ; за ред. М. П. Гандзюка. - К. : Каравела, 2008.
18. Reis E. The lean startup: how today’s enterpreneurs use continuous innovation to create radically successful businesses – Crown Business, 2011. - ISBN: 9780307887894, 336ст.
19. Алекс Д. Ш. Искусственный интеллект в компьютерных играх / Дж. Шампандар Алекс., 2007. – 745 с.
20. Comparing React Native to Cordova | Toptal [Електронний ресурс].– Режим доступу: https://www.toptal.com/mobile/comparing-react-native-to-cordova
21. Kaner, Cem; Falk, Jack; Nguyen, Hung Quoc (1999). Testing Computer Software, 2nd Ed. New York, et al: John Wiley and Sons, Inc. c. 480. ISBN 0-471-35846-0.
22. Лайза Криспин, Джанет Грегори Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М. : «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series). — 1000 прим. — ISBN 978- 5-8459-1625-9.

Додаток А — Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНО ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою

програмної інженерії

“___ “_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської дипломної роботи

на тему: «Розробка автономного віртуального світу з динамічно підтримуючим штучним інтелектом з використанням MVC рхітектури»

Керівник роботи:

д.ф.-м.н., професор Петрик М. Р.

“___” _____ 2019р.

_____

Виконавець:

студент групи СПм-62

Шпак Павло Михайлович

“___” _____ 2019р.

м. Тернопіль – 2019

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка автономного віртуального світу з динамічно підтримуючим штучним інтелектом з використанням MVC рхітектури».

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Предметом розробки є Автономна екосистема з підтримкою штучного інтелекту для операційних систем типу Windows, Linux та MacOS, для створення якої було використано кросплатформовий фреймворк LibGDX, бібліотеку LWJGL та об'єктно-орієнтовану мову програмування Java. В якості середовища розробки було обрано IntelliJ IDEA Community Edition.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмне забезпечення має виконувати наступні дії:

- створення віртуальної екосистеми;
- можливість взаємодіяти з екосистемою;
- надавати результати симуляції екосистеми у вигляді вихідних даних та графіків.

3.2 Склад та параметри технічних засобів

1) ПК з 1024 Мб оперативної пам‘яті, встановленою системою Windows XP, Vista, Seven, 8, 8.1, 10, MacOS, Linux Ubuntu 14,16,18. Не менше 200 Мб вільного місця на жорсткому диску. Одноядерний процесор з тактовою частотою від 1.4 GHz і більше.

2) наявність встановленої JDK version 8 і вище з можливістю розгорнути java-програми.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати в різних операційних системах (Linux, Windows, MacOS), на яких доступне для встановлення JRE 8.121. Розробку виконувати з використанням бібліотек та технологій мови Java в середовищі програмування IntelliJ IDEA Community Edition 2019.3.1 x64 з використанням JDK 8.

4. СТАДІЇ РОЗРОБКИ

- аналіз предметної області;
- вибір інструментів;
- розробка прикладного програмного забезпечення;
- тестування програмного продукту;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- технічне завдання;
- пояснювальна записка;

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Приймання проводиться спеціально створеною екзаменаційною комісією в термін до:

“__” лютого 2017р.

ДОДАТОК Г

Текст програми на клієнті

Лістинг В.1 — клас “GameMain”

```
public class GameMain extends ApplicationAdapter {
	SpriteBatch batch;
	World world;
	MainMenu mainMenu;
	TextureManager textureManager;

		@Override
	public void create ()
	{
		batch = new SpriteBatch();
		textureManager = new TextureManager();

		world = new World(40, textureManager);
		//world.fillWorld();
		world.fillWorldforMenu();
		world.setCamera(128,90);
		mainMenu = new MainMenu(textureManager);

		System.out.println("Start");
	}

	@Override
	public void render ()
	{
		Gdx.gl.glClearColor(0, 0, 0, 1);
		Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
		batch.begin();

		if(!mainMenu.isShowing()) {
			world.render(batch);
			world.listner();
		}

		mainMenu.render(batch);
		mainMenu.listner();



		if(Gdx.input.isKeyPressed(Input.Keys.ESCAPE)) Gdx.app.exit();

		batch.end();
	}

	@Override
	public void dispose ()
	{
		batch.dispose();
		world.dispose();
		textureManager.dispose();
	}
}
```

Лістинг В.2 — клас “World”

```
public class World
{
    int worldGround[][];
    int worldGType[][];
    int worldSize;
    int cameraPosX, cameraPosY, cameraSpeed;
    int textureSize;

    int counterTime;

    final static int GRASS = 0;
    final static int WATHER = 1;

    TextureManager tManager;

    public World(int size, TextureManager tMan)
    {
        worldGround = new int[size][size];
        worldGType = new int[size][size];
        this.worldSize = size;
        tManager = tMan;
        textureSize = tManager.groundTextures[0][0].getHeight()*2;
        counterTime = 0;
        cameraSpeed = 1;
        cameraPosY = cameraPosX = textureSize;
    }

    void fillWorld()
    {
        for(int x = 0; x < worldSize; x++) {
            for (int y = 0; y < worldSize; y++)
            {
                worldGround[x][y] = 0;
                if(x==0 || y==0 || x==(worldSize-1) || y==(worldSize-1))
                    worldGround[x][y] = WATHER;
                worldGType[x][y] = (int)(3*Math.random());
            }
        }
        worldGround[3][3] = WATHER;
        worldGround[3][4] = WATHER;
        worldGround[4][3] = WATHER;
        worldGround[4][4] = WATHER;
    }
    public void fillWorldforMenu()
    {
        for(int x = 0; x < worldSize; x++) {
            for (int y = 0; y < worldSize; y++)
            {
                worldGround[x][y] = 0;
                if(x==0 || y==0 || x==(worldSize-1) || y==(worldSize-1))
                    worldGround[x][y] = WATHER;
                worldGType[x][y] = (int)(3*Math.random());
                if(Math.random()>0.89f)
                    worldGround[x][y] = WATHER;
```

```
            }
        }
        worldGround[2][2] = WATHER;
        worldGround[3][2] = WATHER;
        worldGround[worldSize-2][worldSize-2] = WATHER;
        worldGround[worldSize-2][worldSize-3] = WATHER;
        worldGround[worldSize-5][worldSize-5] = WATHER;
        worldGround[worldSize-5][worldSize-6] = WATHER;
        worldGround[worldSize-6][worldSize-5] = WATHER;
        worldGround[worldSize-7][worldSize-5] = WATHER;
    }

    public void render(SpriteBatch batch)
    {
        int xP, yP, osX, osY;
        xP = yP = osX = osY = 0;
        int xCap, yCap;
        xCap = Gdx.graphics.getWidth()/textureSize+2;
        yCap = Gdx.graphics.getHeight()/textureSize+2;

        counterTime++;

        if(cameraPosX!= 0)
        {
            xP = cameraPosX / textureSize;
            if( xP > 0) xP--;
            osX = cameraPosX % textureSize;
            xCap+= xP;
        }
        if(cameraPosY!= 0)
        {
            yP = cameraPosY / textureSize;
            if(yP > 0) yP--;
            osY = cameraPosY % textureSize;
            yCap += yP;
        }
        for (int x = xP, x1 = 0; x < xCap; x++, x1++)
        {
            for(int y = yP, y1 = 0; y < yCap; y++, y1++)
            {
                if(y<worldSize && x<worldSize) {

batch.draw(tManager.groundTextures[worldGround[x][y]][worldGType[x][y]],
                            x1 * textureSize - osX,
                            y1 * textureSize - osY,
                            textureSize, textureSize);
                    if (counterTime>20 && worldGround[x][y] == WATHER)
                    {
                        worldGType[x][y]++;
                        if(worldGType[x][y]>2) worldGType[x][y] = 0;
                    }
                }
            }
        }
        if (counterTime>20) counterTime = 0;
```

```
        /*
        counterTime++;
        if(counterTime>120)
        {
            System.out.println(" xP= "+xP+ " xCap= "+xCap);
            counterTime = 0;
        }
         */
    }
    public void setCamera(int x, int y)
    {
        cameraPosX = x;
        cameraPosY = y;
    }
    public void moveCameraRight()
    {
        if (cameraPosX<((worldSize+1)*textureSize)-Gdx.graphics.getWidth())
        cameraPosX += cameraSpeed;
    }
    public void moveCameraLeft()
    {
        if(cameraPosX>textureSize)
            cameraPosX -= cameraSpeed;
    }
    public void moveCameraUp()
    {
        if (cameraPosY<((worldSize+1)*textureSize)-Gdx.graphics.getHeight())
            cameraPosY += cameraSpeed;
    }
    public void moveCameraDown()
    {
        if(cameraPosY>textureSize)
            cameraPosY -= cameraSpeed;
    }

    public void listner()
    {
        if (Gdx.input.isKeyPressed(com.badlogic.gdx.Input.Keys.D))
moveCameraRight();
        if (Gdx.input.isKeyPressed(com.badlogic.gdx.Input.Keys.A))
moveCameraLeft();
        if (Gdx.input.isKeyPressed(com.badlogic.gdx.Input.Keys.W))
moveCameraUp();
        if (Gdx.input.isKeyPressed(com.badlogic.gdx.Input.Keys.S))
moveCameraDown();
        if(Gdx.input.isKeyJustPressed(Input.Keys.PLUS)) cameraSpeed++;
        if(Gdx.input.isKeyJustPressed(Input.Keys.MINUS)&& (cameraSpeed>1))
cameraSpeed--;
    }

    void dispose()
    {

    }
}
```