Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до магістерської кваліфікаційної роботи

магістр
(освітньо-кваліфікаційний рівень)

на тему: Підвищення рівня інтелектуальності поведінки персонажів у стратегічних іграх з урахуванням особливостей місцевості

Виконав: студент II курсу
групи 1ПІ-18м
спеціальності
121 – інженерія програмного забезпечення
(повне найменування вищого навчального закладу)
Любивий. Б. О.
(прізвище та ініціали)
Керівник: к.т.н., доц. Романюк О. В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. Іванчук Я. В.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О. Н.
«___» _____ 2019 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Любивому Богдану Олександровичу

1. Тема роботи: «Підвищення рівня інтелектуальності поведінки персонажів у стратегічних іграх з урахуванням особливостей місцевості», керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "__"_____2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: Жанр гри – тактично-стратегічна в реальному часі; базовий алгоритм керування поведінкою персонажів – flocking AI; стандартна кількість одиниць в загоні гравця – 3, максимальна кількість ворожих одиниць - 150.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та порівняльна характеристика сучасних методів керування поведінкою персонажів у стратегічних іграх; розробка методу керування поведінкою персонажів з використанням матриць небезпеки; розробка програмних засобів реалізації запропонованого методу в стратегічній грі; тестуванння розробленого програмного продукту; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу: титульний слайд; мета, об'єкт, предмет та завдання дослідження; наукова новизна та практична цінність; flocking AI; поняття матриці небезпеки; розробка методу керування поведінкою персонажів з використанням матриці небезпек; сценарії поведінки ворожих персонажів; алгоритм роботи матриці небезпеки; розробка інтерфейсу головного меню;

розробка інтерфейсу вікна гри; тестування роботи методу та програмної реалізації; економічне обґрунтування; заключний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О. В., к.т.н, доцент кафедри ПЗ		
5	Бальзан М.В. к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання та існуючих аналогів, постановка задач дослідження	04.09.2019 – 30.09.2019	Вик.
2	Розробка методу керування ворожими юнітами з використанням «Матриці Небезпеки»	30.09.2019 – 27.10.2019	Вик.
3	Розробка структур і алгоритмів програмного продукту	28.10.2019 – 10.11.2019	Вик.
4	Розробка програмної реалізації модифікованої моделі штучного інтелекту “Flocking AI”	11.11.2019 – 15.11.2019	Вик.
6	Тестування розробленого програмного продукту	16.11.2019 – 21.11.2019	Вик.
7	Економічна частина	17.11.2019 – 20.11.2019	Вик.
8	Оформлення матеріалів до захисту МКР	04.09.2019 – 21.11.2019	Вик.

Студент _____ **Любивий Б.О.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Романюк О.В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У магістерській кваліфікаційній роботі «Підвищення рівня інтелектуальності поведінки персонажів у стратегічних іграх з урахуванням особливостей місцевості» запропоновано поняття матриці небезпеки та розроблено метод керування поведінкою персонажів супротивника в стратегічних іграх, що використовує матрицю небезпеки. Розроблено гру “DPS Log”, в основі роботи якої лежить запропонований метод.

В ході виконання проаналізовано сучасні методи керування поведінкою юнітів та ігри аналоги. Обґрунтовано вибір засобів для розробки програмного забезпечення. До них належить середовище розробки Galaxy Editor, технологія XML та мова програмування скриптів Galaxy. Побудовано моделі та структуру автоматизованої системи, розроблено програмні модулі та протестовано коректну роботу програми.

ABSTRACT

In the master's qualification work on "Enhancing character intelligent behaviors in strategy games based on terrain" both concept of the danger matrix and method of unit behavior control based on danger matrix layer showcased in “DPS Log” – real time strategy game created for this purpose, were developed.

In the course of work, the existing methods of control of the units’ behavior were analyzed, as well as the existing program analogues. Software has been developed and tested to implement and research the developed methods using the following tools: Galaxy script and programming language, Galaxy Editor IDE, XML technology.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сьогодні комп'ютерні ігри відіграють особливу роль в розвитку людини, оскільки можуть допомагати їй у розвитку багатьох навичок. За допомогою комп’ютерних ігор людина несвідомо навчається. Певна гра тренує конкретні навички: бути зібраною та уважною в потрібні моменти; запам’ятовувати великий обсяг інформації; логічно мислити, активізувати розумову діяльність; розвивається образне мислення; йде активний розвиток дрібної моторики рук та очей; розвивається просторове мислення.

Тактична рольова гра (англ. tactical role-playing game) — жанр відеоігор, що поєднує елементи рольових і стратегічних відеоігор [1]. Основний акцент ігрового процесу в тактичних рольових іграх робиться на прийнятті тактичних рішень під час бою. Жанр позбавлений чітких рамок, і багато ігор, які належать до нього, можуть бути віднесені до комп'ютерних рольових ігор або покрокових стратегій.

У тактичних рольових іграх гравець управляє групою персонажів, які володіють набором певних характеристик; обстежує ігровий світ, виконує завдання, вступає в бої з різного роду противниками [2].

Одним із найбільш важливих аспектів стратегічної гри, який має сильний плив на гравця є штучний інтелект ворогів. Саме він, в поєднанні з алгоритмами знаходження шляху впливає на рівень задоволеності гравця та складність гри. Вороги повинні бути швидкими і хижими, або налаштованими і добре захищеними, щоб гравець справді відчув складність гри та мав шанс перевірити свої навички і навчитись чомусь.

Умовно «інтелект» ворога визначається двома частинами: що він робить і як він це робить. Згідно з цим методи керування поведінкою ворожими військами в стратегіях можна поділити на два основних види: механіки руху та взаємодії окремих юнітів (юніт – окрема бойова одиниця, складова частина

війська) з іншими об’єктами у грі, та штучний інтелект, що стоїть за цими діями ворогів.

Механіки руху включають в себе алгоритми знаходження шляху, організації руху юнітів в групах, динамічної зміни руху при появі нових перешкод, при цьому використовуючи мінімальну можливу кількість обчислювальних ресурсів комп’ютера.

Штучний інтелект натомість це нейронна мережа основана на нечіткій логіці, що обраховує параметри конкретної ситуації в конкретний момент часу для прийняття оптимального рішення. Нейронні мережі зазвичай мають декілька шарів обчислювальних блоків-нейроні, деякі з них приховані від користувача, щоб проводити потрібні обчислення, а деякі навпаки виводять свої результати прямо до користувача[3].

Не зважаючи на десятиліття розробок в цій області, штучні інтелекти ворогів все ще далекі від ідеалу, та часто виконані не оптимальним способом, мають серйозні проблеми зі стабільністю в обставинах сильного навантаження кількістю ворожих одиниць на бойовій мапі.

У зв’язку з цим, удосконалення методів, що використовуються в штучному інтелекті ворогів, є актуальною і важливою задачею.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення рівня інтелектуальності поведінки персонажів у сучасних стратегічних іграх, за рахунок розробки нового методу керування поведінкою персонажів, що забезпечував би можливість маневрування ворожого війська місцевістю.

Основними задачами дослідження є:

- проаналізувати існуючі методи керування ворожими юнітами та алгоритми руху;
- розробити та описати матрицю небезпеки;

- розробити удосконалений метод керування поведінкою персонажів, з використанням матриць небезпеки;
- розробити стратегічну гру “DPS Log”, у якій реалізовано розроблений метод керування поведінкою персонажів.

Об’єкт дослідження – процес керування поведінкою персонажів у стратегічних іграх.

Предмет дослідження – методи та засоби керування поведінкою персонажів у стратегічних іграх.

Методи дослідження. У процесі дослідження застосовувались: теорія алгоритмів, комп’ютерне моделювання переміщення для відображення руху, теоретичні основи побудови архітектури програмного забезпечення. У процесі дослідження також були застосовані сучасні методи створення ігрового інтерфейсу та моделювання персонажів.

Наукова новизна отриманих результатів.

1. Вперше запропоновано поняття матриці небезпеки, яка дозволила підвищити рівень інтелектуальності та реалістичності поведінки ворожих персонажів у стратегічних іграх.

2. Удосконалено метод керування поведінкою персонажів, який, на відміну від відомих, використовує матрицю небезпек, що дозволило підвищити рівень інтелектуальності поведінки персонажів у стратегічних іграх.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритм та програмні засоби для підвищення інтелектуальності та реалістичності поведінки ворожих персонажів у стратегічних іграх.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати:

- проведення дослідження існуючих методів керування поведінкою ворогів [4];

- запропоновано вдосконалений метод керування поведінкою ворогів з використання матриці небезпеки [5].

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних конференціях:

- XII міжнародна науково-практична конференція «Інформаційні технології і автоматизація – 2019» (Одеса, 2019);
- Міжнародна науково-практична конференція «Електронні інформаційні ресурси в освіті і науці: створення, використання, доступ» (Вінниця, 2019).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях, у тому числі 2 – у матеріалах конференцій.

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 20 найменувань, 4 додатків. Робота містить 32 ілюстрації, 9 таблиць.

1 АНАЛІЗ ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА СУЧАСНИХ МЕТОДІВ КЕРУВАННЯ ПОВЕДІНКОЮ ПЕРСОНАЖІВ У СТРАТЕГІЧНИХ ІГРАХ

1.1 Аналіз сучасних методів керування поведінкою персонажів у сучасних стратегічних іграх

Методи керування поведінкою ворожих військ в стратегіях можна поділити на два основних види: механіки руху та взаємодії окремих юнітів з іншими об’єктами у грі, та штучний інтелект, що стоїть за цими діями ворогів.

Механіки руху включають в себе алгоритми знаходження шляху, організації руху юнітів в групах, динамічної зміни руху при появі нових перешкод, при цьому використовуючи мінімальну можливу кількість обчислювальних ресурсів комп’ютера.

Штучний інтелект натомість це нейронна мережа основана на нечіткій логіці, що обраховує параметри конкретної ситуації в конкретний момент часу для прийняття оптимального рішення. Нейронні мережі зазвичай мають декілька шарів обчислювальних блоків-нейроні, деякі з них приховані від користувача, щоб проводити потрібні обчислення, а деякі навпаки виводять свої результати прямо до користувача. Схема простої мережі зображеною на рисунку 1.1.

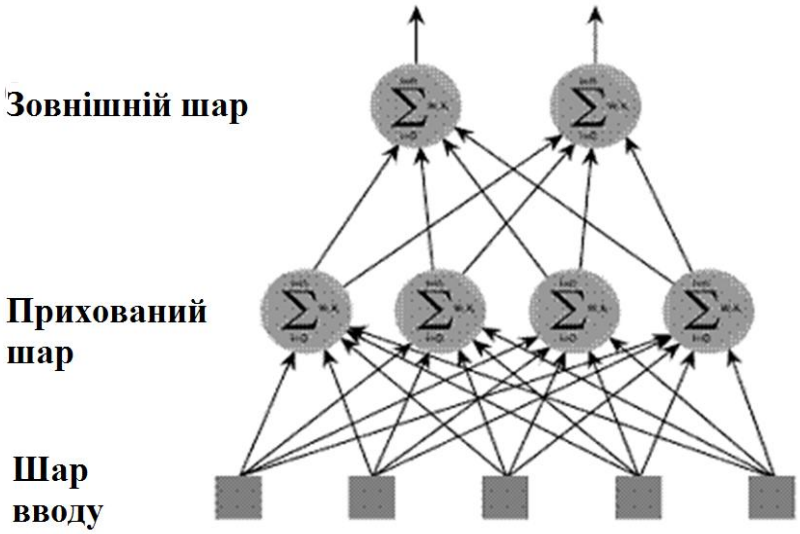


Рисунок 1.1 – Схема нейронної мережі

Такі мережі можуть приймати короткострокові рішення в бойових сутичках за допомогою відповідних формул. Наприклад, рішення нападати чи відступати. Для цього використовуються поняття відносної сили атаки та відносної швидкості. Опис атрибутів поданий у таблиці 1.1.

Таблиця 1.1

Опис комплексних атрибутів військ

Відносна сила атаки	$\frac{\frac{\sum_{i=1}^{i=k} OAS_i \times OAP_i}{EAr_1}}{\frac{\sum_{j=1}^{j=m} EAS_j \times EAP_j}{OAr_1}} \times 50$	OAS = Власна швидкість атаки EAS = Швидкість атаки ворога $\sum_{i=1}^{i=k} OAS_i \times OAP_i$= Сума повноважень військ, що атакують цільовий загін EAr_1= Броня цільового війська $\sum_{j=1}^{j=m} EAS_j \times EAP_j$= Сума повноважень військ противника, що атакують війська = Броня ворожого війська в цілі війська OAr_1= Броня війська. Ця змінна нормалізується шляхом множення на 50. Результат 50 означає рівність.

Відносна швидкість	$\frac{\frac{\sum_{i=1}^{i=k} OS_i}{k}}{\frac{\sum_{j=1}^{j=l} ES_j}{l}} \times 50$	OS = Власна швидкість руху ES = Швидкість руху ворога $\frac{\sum_{i=1}^{i=k} OS_i}{k}$ = Середнє значення швидкості атакуючого війська до цільової частини. $\frac{\sum_{j=1}^{j=l} ES_j}{l}$ = Середня швидкість наступу військ противника, що нападають на війська. Ця змінна нормалізується шляхом множення на 50. Результат 50 означає рівність.

Стани, які може мати армія гравця чи ворога в такому випадку, показані у таблиці 1.2.

Таблиця 1.2

Атрибути війська і різні їх стани

Власне здоров'я	**Здоров'я ворога**	**Відносна сила атаки**	**Відносна швидкість**
Майже мертві	Майже мертві	Менша	Повільніша
Травмовані	Травмовані	Рівна	Рівна
Нормальний стан	Нормальний стан	Більша	Швидша

Прикладами ситуацій і правил такого штучного інтелекту є таблиця 1.3.

.

Таблиця 1.3

Алгоритми дії при певних ситуаціях

ВХОДИ	**ВИХІД**

Власне здоров’я	**Здоров'я ворога**	**Відносна сила атаки**	**Відносна швидкість**	**Атака/Втеча**
Не «майже мертві»	Усі можливості	Усі можливості	Усі можливості	Атака
Майже мертві	Усі можливості	Усі можливості	Повільніша	Атака
Майже мертві	Травмовані	Рівна	Не Швидша	Атака

Продовження таблиці 1.3

1	2	3	4	5
Майже мертві	Майже мертві	Менша	Не Повільніша	Втеча
Майже мертві	Травмовані	Менша	Не Повільніша	Втеча
Майже мертві	Нормальний стан	Менша	Не Повільніша	Втеча
Майже мертві	Нормальний стан	Рівна	Швидша	Втеча
Майже мертві	Нормальний стан	Більша	Швидша	Втеча
Майже мертві	Травмовані	Рівна	Швидша	Втеча

Якщо військо не має важких поранень, воно повинно продовжувати атаку і не повинно тікати. Проблема з'являється в стані «Майже мертві», який необхідний як умова для всіх можливостей втечі. Навіть якщо військо має важкі поранення але повільніше ніж ворог, воно все рівно повинно атакувати, тому що можливості врятуватися немає. А решту правил легко зрозуміти за допомогою наведеної вище таблиці.

1.2 Порівняльний аналіз існуючих реалізацій методів керування поведінкою персонажів супротивника

На сьогоднішній день існує багато схожих по жанру ігор, що використовують алгоритми штучного інтелекту в певній мірі. Приладом такої гри є Rainbow Six 1998 року. Ця гра мала дуже сильну систему планування перед місією, під час якої в гравця був шанс обрати склад штурмової групи, їхнє спорядження, а також дослідити схеми будівлі, в якій буде проходити місія і прокласти маршрут для кожної із груп. Під час власне штурму будівлі, у гравця є можливість брати одного із членів групи під особистий контроль і, таким чином, вводити корективи в раніше заплановий план штурму [6]. На рисунку 1.2 наведено зображення зі стадії планування.

Рисунок 1.2 – Стадія планування гри Rainbow Six 1998

Ця гра має декілька важливих недоліків у керування ворогами гравця. Більшічть часу вони просто стоять у кімнатах, що були призначені їм, та очікують наближення юніта гравця, щоб почати діяти. Якщо гравець виказує свою позицію звуками пострілів, найближчі вороги отримають від гри місцезнаходження юніта, що стріляє, та наближаються до нього по найкоротшому шляху поки не зможуть відкрити вогонь у відповідь.

Це застарілий підхід до програмування штучного інтелекту, який об’єднує нереалістичність поведінки ворогів разом з шахрайством комп’ютера, так як він завжди знає точне місцезнаходження гравця.

Іншим прикладом є серія ігор X-COM, яка є чистим представником жанру тактичної покрокової стратегії. Там також є можливість збирати і споряджати загін, хоч і не до такої міри як у Rainbow Six, і геймплей, який заохочує швидке прийняття рішень. Момент із гри зображений на рисунку 1.3.

Хоча поведінка ворогів тут більш агресивна, коротко її можна описати алгоритмом з трьох пунктів. Спочатку, вороги що відносно рівномірно розставлені по бойовій мапі, стоять на місці, очікуючи поки гравець наткнеться на один з ворожих юнітів. Після того як сталась ця подія, усі вороги на карті починають зближуватись на місце першого контакту. Коли вони потрапляють в зону видимості гравця, вороги починають атакувати його відповідно до свого шаблону атаки. Недоліками такого підходу є те, що гравець може просто уникати контакту майже всю місію і лише через випадковість зтикнутись з ворогом. І хоча тихий підхід часто заохочується в тактичних іграх, X-COM робить ворогів абсолютно пасивними половину часу, що робить задачу гравця занадто простою.

Рисунок 1.3 – Приклад бойової сиуації в грі X-COM

Також до цього жанру належить така серія ігор як Disciples. Це покрокова стратегія як і X-COM, але має додаткові можливості і особливості, що відсутні у футуристичного аналогу. Disciples має у собі елементи рольової гри, у сеттінгу магічного середньовіччя, у вигаданому світі під назвою Невендаар.

Основна частина гри відбувається на глобальній карті, де гравець керує армією, яка схематично зображена персонажем гравця. Вигляд такого режиму зображено на рисунку 1.4.

Рисунок 1.4 – Карта світу і переміщення на ній в грі Disciples II

Коли стикаються дві армії, гра з глобальної карти перемикається на екран бою. Війська як гравця так і противника не можуть переміщуватися і розташовуються в двох рядах по 3 клітинки. За промовчуванням у перший ряд ставляться сильніші в рукопашному бою бійці, в другий — стрільці та маги, але гравець вільний змінювати порядок. Максимум на полі бою може бути 12 бійців. Бійці ближнього бою здатні атакувати лише тих ворогів, що стоять перед ними. На полі бою нападники завжди розташовуються вгорі, захисники — внизу.

Існує можливість чекати — хід персонажа переноситься в кінець списку черговості для даного раунду, і захиститися — пропустити хід в обмін на певну кількість одиниць броні. Здатися, втративши загін разом з героєм, але позбавивши противника досвіду за битву, в цій грі не можна. Приклад бою зображено на рисунку 1.5.

Рисунок 1.5 – Бій з бандитами в грі Disciples II

Основний недолік цієї гри полягає в тому наскільки абстрактно вона зображає бій. На основі проведеного аналізу аналогів, було виявлено, що основними їх недоліками є недостатньо точне відображення реальних обставин на полі бою через особливості геймплею. В режимі глобальної мапи, більшість армій противника теж пасивні та очікують наближення гравця, щоб напасти на нього.

Іншою, популярною та новою грою є Helldivers – тактичний кооператив на чотирьох гравців, де вони повинні командою виконувати поставлені задачі в місії та звільняти території. Приклад місії зображено на рисунку 1.6.

Рисунок 1.6 – Захист позиції у грі Helldivers

Існують три фракції ворогів, кожен з яких має свої унікальні юніти, свою власну модель штучного інтелекту та алгоритми переміщення по карті (спец можливості, як наприклад телепортація чи проривання тунелів під землею).

Штучний інтелект, що використовується в Helldivers має великий потенціал, але страждає від поганої реалізації. Цей недолік сильно погіршує загальну картину поведінки ворогів: вони часто застигають на місці, коли потрібно змінити маршрут, ідуть в стіну, фактично залишаючись на одному місці, тощо.

Таким чином, можна побачити, що у сучасних стратегічних іграх є значні проблеми з штучним інтелектом та його реалізацією, тому розробка нового методу підвищення реалістичності поведінки ворожих одиниць на полі бою є актуальною.

1.3 Постановка задачі

Метою магістерскої кваліфікаційна роботи є підвищення рівня інтелектуальності поведінки персонажів у сучасних стратегічних іграх, за

рахунок розробки нового методу керування поведінкою персонажів, що забезпечував би можливість маневрування ворожого війська місцевістю.

Для цього необхідно:

- розробити та описати поняття матриці небезпеки;
- розробити удосконалений метод керування поведінкою ворожими військами з використанням матриці небезпеки;
- розробити програмну систему для впровадження методу;
- розробити інтерфейс для програмної системи;
- провести тестування системи, виправити виявлені помилки.

Необхідно забезпечити стійну роботу алгоритмів методу, щоб уникнути вповільнення роботи системи.

1.4 Висновки

У результаті аналізу предметної області було досліджено принципи роботи штучного інтелекту та алгоритму прийняття рішень у галузі стратегічних ігор. Розглянуто існуючі варіанти реалізації систем керування юнітами.

Розглянуто варіанти методів керування поведінкою ворожих юнітів у існуючих стратегічних іграх. Було доведено важливість якісного штучного інтелекту у жанрі стратегічних ігор, та розглянуто можливості його удосконалення. Розглянуті аналоги не використовують усі можливості симуляції для створення реалістичного штучного інтелекту в грі, тому найкращим рішенням даної проблеми є розробка методу, на базі якого буде розроблена програмна система.

2 РОЗРОБКА ПОНЯТТЯ МАТРИЦІ НЕБЕЗПЕКИ ТА МЕТОДУ КЕРУВАННЯ ПОВЕДІНКОЮ ПЕРСОНАЖІВ З ЇЇ ВИКОРИСТАННЯМ

2.1 Аналіз моделі штучного інтелекту “Flocking AI”

Часто у відеоіграх ворожі юніти повинні рухатися в згуртованих групах, а не самостійно. Наприклад, рух ворожих до гравця тварин буде здаватись візуально реалістичнішим, якщо вони рухаються в зграї, а не безцільно ходять навколо. Хижаки, які полюють у зграї, а не самостійно, будуть більш реалістичними та поставлять перед гравцем бойову ситуацію з координованими групами хижаків. Таку ж методику можна застосувати до гігантських мурах, бджіл, щурів чи морських істот.

Ці приклади переміщення, випасу худоби або нападу на табуни чи зграї місцевої фауни можуть здатися очевидними способами, в яких можна використовувати поведінку зграйку в іграх. Незважаючи на це не потрібно обмежувати використання системи флокінгу лише фауною так як вона цілком здатна реалізовувати рух і інших юнітів у грі. Наприклад, при моделюванні стратегії в реальному часі можна використовувати поведінку групового руху для бойового загону. Ці підрозділи можуть бути керованими комп'ютером людьми, тролями, орками або механізованими транспортними засобами різного роду. У моделюванні бойового польоту можна застосувати такий груповий рух до керованих комп’ютерами ескадронів літальних апаратів. Навіть у шутері від першої особи комп'ютерні ворожі чи дружні загони можуть використовувати такий груповий рух.

У всіх цих прикладах ідея полягає в тому, щоб юніти на мапі рухалися згуртовано з ілюзією наявності мети. Це виглядає разюче відмінно від звичного переміщення юнітів, де кожен керується своїм особистим маршрутом і відбуваються часті колізії юнітів один з одним.

В основі такої групової поведінки лежать основні алгоритми зграйки, такі, як той, що представлений Крейгом Рейнольдсом у своїй роботі [7]. Можна застосувати алгоритм Рейнольдса, представлений у його первісному вигляді

для імітації зграй птахів, риб чи інших істот, або в модифікованих версіях для імітації групового руху підрозділів, загонів чи повітряних ескадронів

Крейг Рейнольдс запропонував термін *«flocking AI»* для опису поведінки ворожого війська як зграї, а не кожного окремого бійця, подібно до зграї риб чи рою бджіл [8]. Всі особини зграї можуть синхронно змінювати напрямок руху в одну мить, і коли наступного моменту особини, що знаходяться попереду зграї, повернуть, решта зграї піде за ними і хвиля повороту поширюється через усю зграю. Реалізація Рейнольдса не передбачає наявності лідера у зграї, натомість уся зграя сприймається як єдиний організм, що має власний інтелект. Така поведінка реалізовується на основі трьох правил:

- Згуртованість: кожна особина зграї повинна орієнтуватися на середнє положення своїх сусідів.
- Вирівнювання: кожна особина зграї повинна спрямовуватись відповідно до середнього напрямку руху сусідів.
- Розмежування: потрібно уникати скупчення сусідів, щоб не штовхати один одного.

З цих трьох правил зрозуміло, що кожен підрозділ повинен знати про своє місцеве оточення - він повинен знати, де розташовані його сусіди, куди вони очолюються та наскільки вони близькі до нього.

У фізично змодельованому безперервному середовищі можна керувати, застосовуючи зусилля рульового керування на імітовані одиниці.

У середовищі координатної сітки можна використовувати методи огляду, які використовуються в прикладах переслідування та ухилення на основі квадратів координат, щоб одиниці рухалися до певної точки. Наприклад, у випадку правила згуртованості, юніт «голова» повинен дивитись до середнього місця розташування, вираженого координатною сіткою, своїх сусідів.

В основному кожен підрозділ знає своє місцеве оточення - тобто знає середнє місце розташування, напрямок та розділення між ним та іншими підрозділами групи, що знаходиться в його безпосередній близькості. Підрозділ

не обов’язково знає, що робить вся група в даний момент часу. Рисунок 2.1 ілюструє локальну видимість юніта.

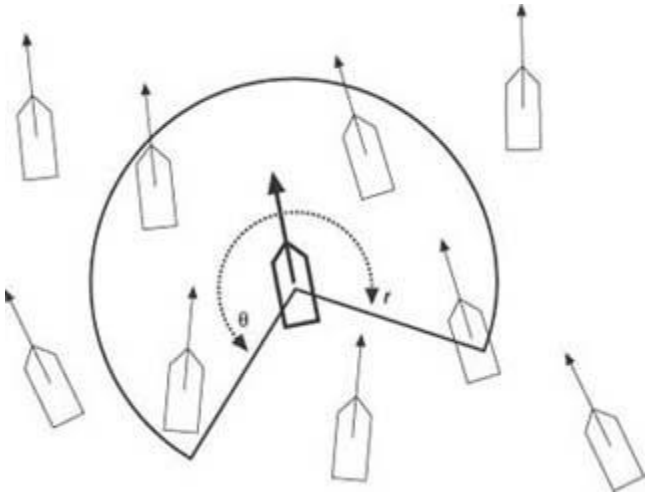


Рисунок 2.1 – Огляд одиниці

На рисунку 1 умовно зображено бійця (в середині рисунка) з дугою видимості радіусом r навколо нього та кутом огляду θ. Кожен боєць може бачити лише тих бійців, які потрапляють у сформований сектор. Видимі бійці використовуються при застосуванні правил стікання; решта бійців ігнорується.

Сектор видимості визначається двома параметрами – радіусом дуги r та кутом огляду θ. Обидва параметри впливають на результат руху стікання і можуть бути налаштовані в залежності від конкретних потреб.

Великий радіус дозволить одиниці бачити більше групи, що призводить до більш згуртованої зграї. Така зграя має тенденцію до того, що рідше розділяється на менші зграї, оскільки кожна одиниця може бачити, де знаходиться більшість чи всі одиниці, і спрямовуватись відповідно. З іншого боку, менший радіус, як правило, збільшує ймовірність розколу зграї, утворюючи менші групи. Зграя може розколотися, якщо деякі одиниці тимчасово втрачають з поля зору своїх сусідів, що може бути їх зв’язком з більшою зграєю. Коли це станеться, відокремлені одиниці сформують меншу зграю і, можливо, можуть знову приєднатися до інших, якщо вони знову

потраплять у поле зору. Навігація навколо перешкод також може спричинити розвал зграї.

Інший параметр θ вимірює поле зору кожної одиниці. Найширше поле зору - це 360 градусів. Деякі алгоритми флокінгу використовують поле зору 360 градусів, оскільки це легше здійснити; однак, що виникає поведінка зграї може бути дещо нереальною. Більш поширене поле зору схоже на те, що проілюстровано на рисунку 2.1 , де за кожною одиницею є чітка сліпа пляма. Загалом, широке поле зору, таке, як це показано на рисунку 2.1 в якому кут огляду приблизно 270 градусів, призводить до добре сформованих зграй. Вузьке поле зору, наприклад, проілюстроване на рисунку 2.2, в якому кут огляду вузький 45 градусів, призводить до появи отари, які мають тенденцію більше нагадувати лінію мурашок, що йдуть по доріжці.

Рисунок 2.2 – Зграї, створені з широким та вузьким полем огляду

Обидва результати мають своє використання. Наприклад, якщо при імітації ескадрильї винищувачів, можна використовувати широке поле зору. При цьому, при імітації загонів армійських підрозділів, які підкрадаються до когось, бажано використовувати вузьке поле зору, щоб вони слідували один за одним у рядку і, отже, не представляли широкої цілі, коли вони здійснюють свій напад. При поєднанні останній випадку із уникненням перешкод, візуально

схоже, що ваші підрозділи слідкують крок в крок за людиною, яка оминає перешкоди.

Хоча Flocking AI є надійною і продуманою системою, вона більше спрямована на симуляцію стай тварин, тому для використання її для керування поведінкою бойових одиниць на полі бою, потрібно розробити метод керування що дозволяв би більш ворожим персонажам реалістично реагувати на небезпечні ситуації та атаки супротивника.

2.2 Поняття матриці небезпеки

Більшість сучасних стратегічних ігор є мультиплеєрними, тобто в одній ігровій сессії грають одночасно 2 чи більше гравців. Вони самі керують рухами своїх військ та обирають рішення. Через це розвиток штучного інтелекту для стратегічних ігор це повільний процес, і інколи немає помітної різниці між штучним інтелектом гри 2019 чи 2001 року.

Часто режим на одного гравця проти комп’ютерного противника є лише тренуванням для гравців і йому не приділяють достатньо уваги. Як наслідок, часто штучний інтелект в стратегічних іграх показує прямоту і механічність усіх рухів, нереалістичну поведінку, так як підвищення складності в основному досягається шахрайством штучного інтелекту супротивника, коли він отримує точну інформацію про усі дії гравця чи отримує непомітні для гравця бонуси, як наприклад швидший темп побудови будівель чи більшу кількість стартових ресурсів.

Самі атаки військ штучного інтелекту зазвичай масовані та примітивні. Ціль атаки задається юнітам завчасно і вони керуються алгоритмами знаходження шляху, прямуючи по найкоротшому шляху до цілі, ігноруючи небезпечні ситуації, що відбувається навколо них. Як приклад такої поведінки, у іграх де є руйнівні спец-здібності з ефектом по області, як наприклад довгий артилерійській обстріл, війська будуть йти через вибухи, зазнаючи значних втрат, навіть не намагаючись обійти вибухи чи перечекати обстріл.

Хоча така поведінка інколи виправдана величезною кількістю окремих одиниць на одній мапі одночасно щоб зменшити навантаження на процесор комп’ютеру, в іграх з меншим масштабом бойових операцій така поведінка неприпустима.

Саме через це було запропоновано поняття матриці небезпеки - інформаційного шару, що може взаємодіяти з алгоритмами штучного інтелекту і алгоритмами знаходження шляху, який міг би підвищити реалістичність поведінки ворожих військ.

Матриця небезпеки – це інформаційний шар бойової мапи, що обчислює поточне місцезнаходження, поле огляду усіх персонажів гравця, а також активність, координати, зону дії та тривалість спец-здібностей що доступні цим персонажам. Ця інформація зберігається у цьому шарі, та за допомогою алгоритму обчислення, проводяться маніпуляції з шаром координатної сітки що збільшують чи зменшують вагу конкретних координат, в залежності від того чи належать вони до активної небезпечної ділянки. Схематично вигляд матриці небезпеки зображено на рисунку 2.3.

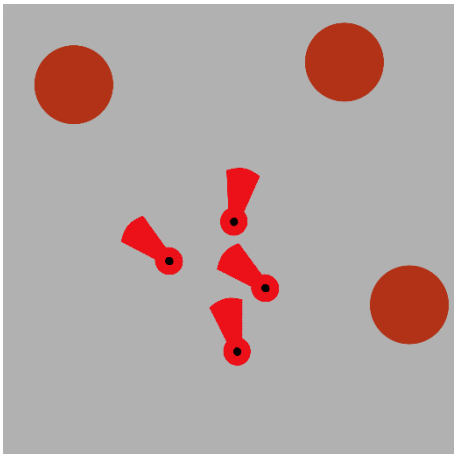


Рисунок 2.3 – Схематичне зображення матриці небезпеки з активними небезпечними ділянками

Матриця небезпеки впливає прямо на координатну сітку гри, змінюючи вагу координат місцевості для алгоритму знаходження шляху. Таким чином, області, що особливо небезпечні, будуть мати більшу вагу в алгоритмі, та будуть прирівняні до важко прохідної місцевості. Як наслідок навіть при

низькому рівні розвитку штучного інтелекту у грі, ворожі війська все рівно будуть намагатись знайти альтернативний підхід навколо небезпечних ділянок.

2.3 Розробка методу керування поведінкою персонажів з використанням матриці небезпеки

Звичайний алгоритм знаходження шляху буде знаходити такий маршрут військової групи, який має мінімальну відстань, або найкоротший час проходження. Приклад результату такого простого алгоритму показано на рисунку 2.4

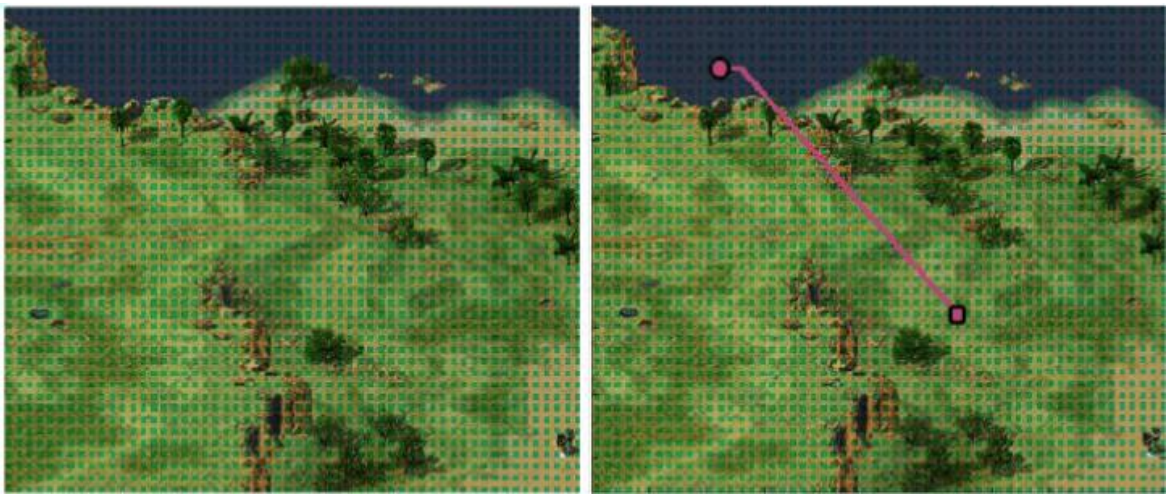


Рисунок 2.4 – Результат роботи алгоритму знаходження маршруту A*

Для відображення небезпеки в певних обставинах, було створено поле видимості/активності юніта. Всередині цього поля, ворожі юніти можуть помічати юнітів гравця і атакувати[9]. Також це поле відображає зону в якій юніт може використовувати свої спец уміння. Схематичне зображення цього поля видно на рисунку 2.5

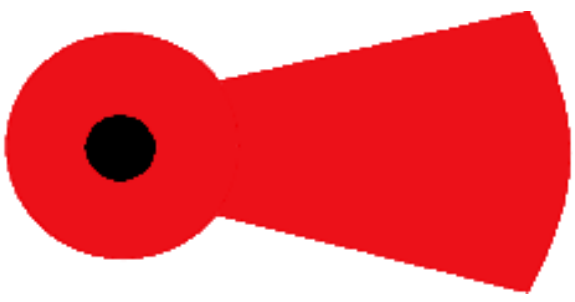


Рисунок 2.5 – Поле видимості юніту

Це поле є одним із елементів «Матриці Небезпеки» динамічного інформаційного шару шо відображає тактичну ситуацію на бойовій мапі. Схематично зв'язок між рівнями мапи зображено на рисунку 2.6.

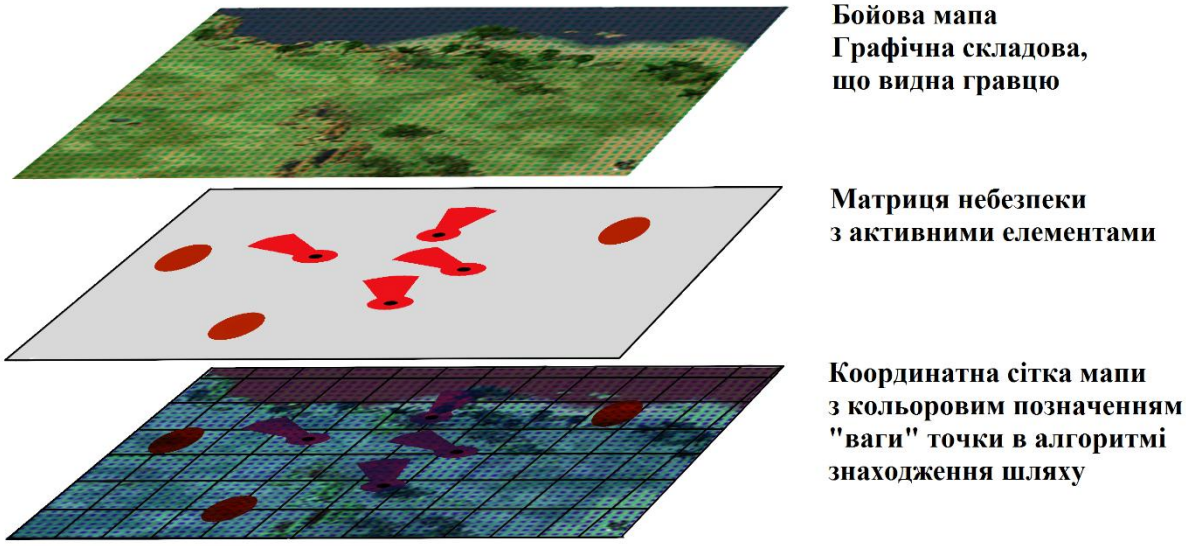


Рисунок 2.6 – Графічний, інформаційний та координатний шар бойової мапи

Верхній шар, який єдиний доступний гравцю для взаємодії, містить графічні моделі усіх юнітів та місцевості. Тут гравець взаємодіє з грою та бачить результати роботи алгоритмів що проходять на шарах знизу.

Середній шар – інформаційний, названий матрицею небезпеки, був розроблений в ході виконання магістерської роботи.

Принцип роботи матриці небезпеки полягає у фонових обчисленнях діяльності гравця та впливу його дій на тактичну обстановку на мапі. Окрім поля видимості/активності юніта, що відображає поле обзору та направленість зброї персонажа гравця, в матрицю було додано додаткові типи «зон небезпеки», які дозволяють у повній мірі оцінити тактичну ситуацію. Приклади зон небезпеки показані на рисунку 2.7.

Рисунок 2.7 – Типи зон небезпеки

Зони небезпеки залежно від свого типу мають різні властивості. Основні характеристики це максимальний ліміт ваги, яку може додавати ця зона в координатну сітку, та швидкість збільшення цієї ваги. Кожен внутрішньо ігровий «тік» - 0.2 секунди – всередині зони небезпеки підвищується вага згідно швидкості збільшення до максимуму в цій зоні. Цей модифікатор спадає у точках які, в момент тіку не знаходяться в активній зоні небезпеки.

Точні характеристики зон небезпеки подані у таблиці 2.1.

Таблиця 2.1

Зони небезпеки

Тип зони	**Максимум модифікатора ваги**	**Швидкість збільшення модифікатору**	**Опис зони**
Поле видимості/ активності юніту	+25	+10 од/сек	Поле видимості відображає пасивну здатність юніту бачити загрозу і бути готовим захищатись.
Зона дії здібності	+75	+25 од/сек	Зона дії активних здібностей як шквал кулеметного вогню чи закладена міна є надзвичайно небезпечними і ворожі юніти повинні намагатись уникати їх усіма способами
Відсутність активних зон небезпеки	0	-10 од/сек	Пасивна зона де залишились модифікатори від матриці небезпеки поступово втрачає свій вплив на прокладення шляху ворогом.

Далі, на рисунку 2.8 видно два простих сценарія. Перший полягає в тому, що ворог знаходиться по звичному шляху переміщення і тому юніт буде намагатись обійти його, без особливих втрат часу. Другий – де ворог і є кінцевим пунктом руху. В такому випадку юніт буде намагатись обійти його зліва чи справа і атакувати збоку.

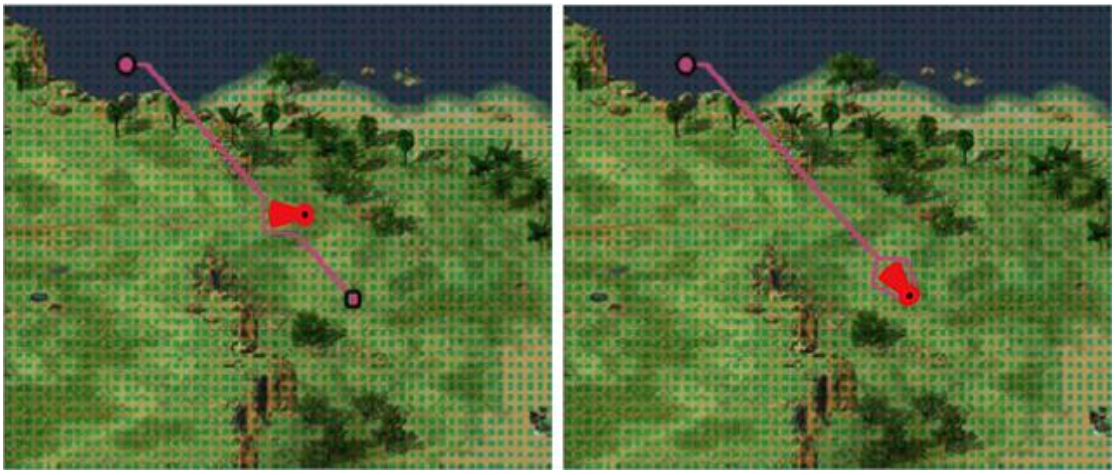


Рисунок 2.8 – Результат роботи алгоритму знаходження маршруту з полями небезпеки

Додатково, на рисунку 2.9 видно сценарій з великою кількістю і персонажів гравця і ворожих одиниць. По схематичним лініям видно, що зграя буде вимушена розділитись на дві, щоб уникнути гравця. Верхня група буде намагатись піти на прорив і маневрувати між юнітами гравця, тоді коли інша буде намагатись обійти їх стороною. В кінці маршруту, якщо перша група вціліє, вони знову об’єднаються разом у одну зграю і продовжать шлях до цілі.

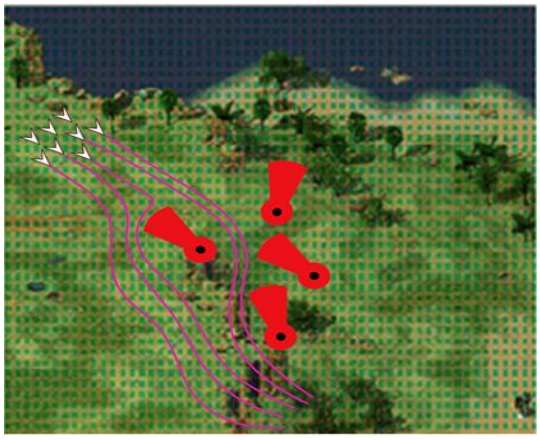


Рисунок 2.9 – Результат роботи алгоритму знаходження маршруту з матрицею небезпеки

Таким чином було описано принцип роботи методу, показано показники що використовуються в обчисленнях, та схематично показано сценарії з описом поведінки ворожих одиниць.

2.4 Розробка блок схеми алгоритму роботи матриці небезпеки

Наступним кроком є створення алгоритму роботи матриці небезпеки та обчислень, які відбуваються в інформаційному шарі. Алгоритм є ключовим при проектуванні додатку, оскільки він буде основою для подальшої розробки методу.

Блок–схема алгоритму роботи матриці небезпеки представлена на рисунку 2.10.

1 – Початок роботи модуля.

2 – Блок очікування введення команди від гравця. Перевірка відбувається кожний внутрігровий тік – 0.2 с.

3 – Якщо Введення = Null. Значит нової команди не наступило, модуль продовжує чекати.

5 – Якщо Введення = команда руху з легітимними координатами відбувається запуск гілки алгоритму для обрахунку полей видимості юнітів.

4 – Отримання позицій усіх юнітів гравця протягом даного тіку гри

6 – Обчислення зони видимості юнітів в даний момент. Збереження даних.

8 – Підвищення модифікатора ваги активних полей зі збереженої інформацією з відповідними до типу швидкістю збільшення і максимальним значенням.

12 – Зменшення модифікатора ваги пасивних зон згідно зі становленою швидкістю.

14 – Якщо позиція юнітів відповідає заданим в введеній команді руху координатам – повернутись до очікування введення. Якщо ні – повернення до блоку 4.

7 – Якщо Введення = команда використання здібностей відбувається запуск гілки алгоритму для обрахунки зон дії здібностей.

10 – Обчислення зони дії здібності. Збереження цих даних.

13 – Підвищення модифікатора ваги активних полей зі збереженої інформацією з відповідними до типу швидкістю збільшення і максимальним значенням.

15 – Запуск фонового таймеру по характеристиці тривалості здібності.

16 – При закінченні часу тривалості здібності, обнулення модифікатора ваги в зоні її дії.

8 – Якщо введення було одним із визначених типів команди, або командою «Вихід» алгоритм закінчує роботу.

11 – Кінець роботи модуля.

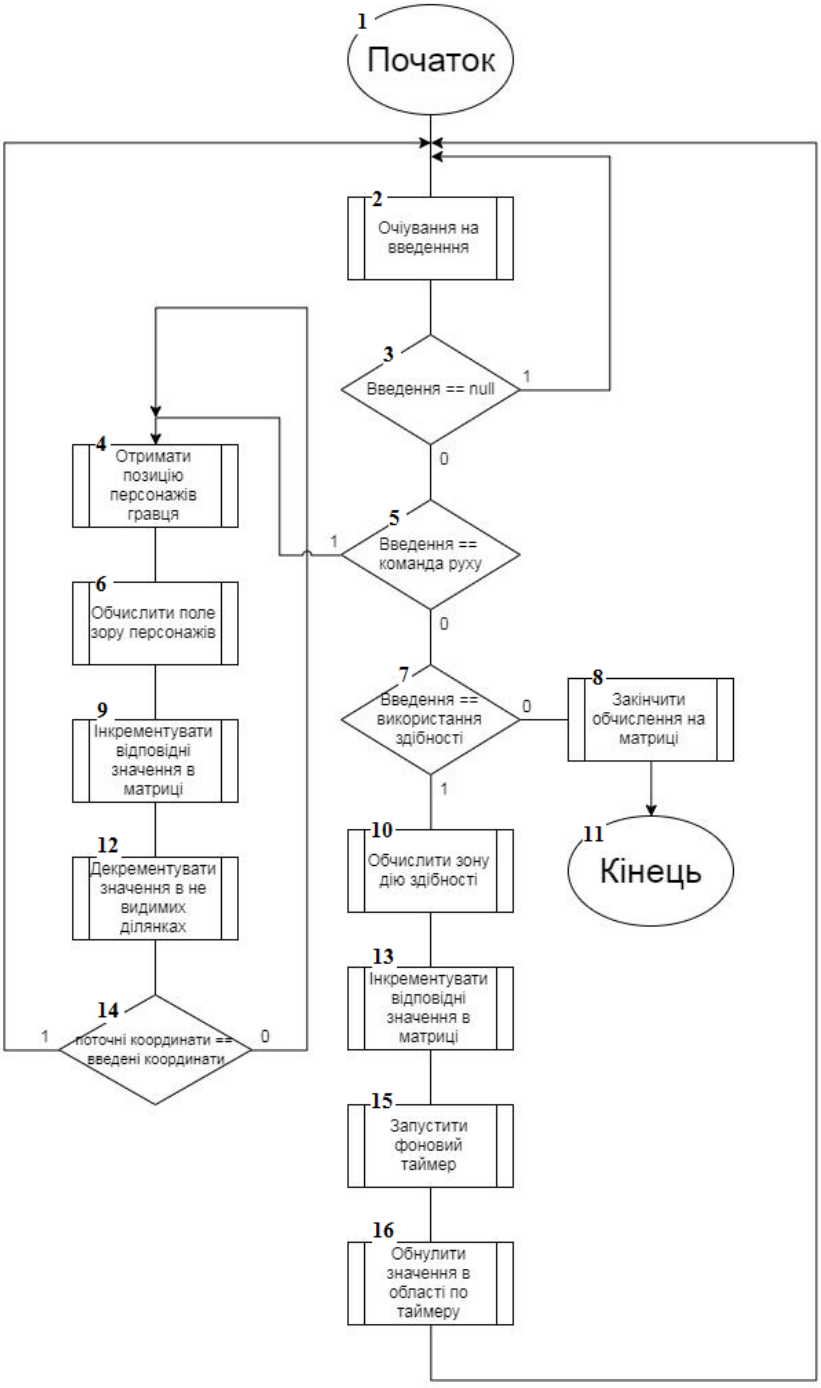


Рисунок 2.10 – Блок–схема алгориту роботи матриці небезпеки

В результаті, створена блок–схема алгоритму роботи матриці небезпеки буде використана у написанні програмної реалізації методу.

2.5 Висновки

У другому розділі було проведено аналіз методів керування поведінкою персонажів у стратегічній грі, виділені переваги та недоліки кожного з

розглянутих варіантів. Описано поняття матриці небезпеки – додаткового інформаційного шару на мапі, що містить інформацію про небезпечні події, що там відбуваються та який вплив на юнітів вони мають. Проаналізовано загальну класифікацію методів та розроблено новий метод керування поведінкою ворожих військ з використанням матриці небезпеки, в результаті чого було досягнуто більш реалістичної поведінки ворогів у бойових ситуаціях.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ УДОСКОНАЛЕНОГО МЕТОДУ КЕРУВАННЯ ПОВЕДІНКОЮ ПЕРСОНАЖІВ

3.1 Варіантний аналіз і обґрунтування вибору програмних засобів

На даний момент у кожного розробника, що починає розробку свого проекту, є вибір: робити весь додаток абсолютно з нуля, чи обрати вже існуючий двигун для нього. Обидва підходи мають свої переваги і недоліки.

Самостійна розробка надає можливість повністю контролювати цей процес, можливості програми обмежуються лише навичками розробниками і часом, який виділений на роботу. Також це набагато дешевше, так як багато двигунів платні.

В той же час розробка на базі існуючого двигуна значно пришвидшує роботу і дозволяє розробникам сильніше розкрити свій дизайнерський потенціал, так як, оскільки їм не потрібно перейматись реалізацією базових, примітивних функцій та механік. Обираючи готовий двигун потрібно звернути увагу на якість його написання, які функції він реалізовує, які технології підтримує. Також значним фактором є ціна.

Для розробки даного проекту було обрано двигун Galaxy Engine, створений компанією Blizzard Entertaiment для своєї гри Starcraft 2. Він написаний на C, C++ з домішками C#, Ассемблеру та з використанням Adobe Flash та системи руйнування середовища Havok. Цей двигун абсолютно безкоштовний та йде з власним, зручним редактором.

Використовуючи цей двигун, було вирішено декілька проблем розробки:

- Наявність редактора, який включає в себе редактор мап, тригерів, скріптів, та текстур, значно пришвидшує процес розробки, та робить його більш комфортним.
- Редактор мап компенсує скромні навички головного розробника у дизайні ландшафтів, надаючи йому прості, але потужні інструменти для терраформації середовища.

- Скріптова мова Galaxy, яка використовується у редакторі тригерів, основана на мові програмування C, з якою головний розробник добре знайомий, тому процес звикання буде швидкий.
- Редактор тригерів має функцію автогенерування скріптового коду на основі графічно створених інструкцій, що пришвидшує процес написання складних сцен та місій.
- Так як усе що написане за допомогою цього двигуна є певною мірою файлами розширення для гри Starcraft 2, їх можна підключити до гри і використовувати її як зручне середовище для відлагодження та тестування, як модульного так і остаточного.

Таким чином, вибором цього двигуна як основи для проекта, забезпечується зручне середовище для розробки з потужними можливостями, економиться час розробки, та компенсується певні слабкі сторони розробників.

3.2 Вибір середовища розробки

Для розробки програмного забезпечення потрібно обрати інтегроване середовище розробки (IDE − Integrated Development Environment). IDE − це комп'ютерна програма, що допомагає програмісту розробляти нове програмне забезпечення чи модифікувати (удосконалювати) вже існуюче.

В даному випадку, так як використовуться двигун Galaxy Engine, єдиним середовищем розробки, яке його підтримує, є Galaxy editor.

Galaxy Editor був опублікований для загального користування у квітні 2010 року і з тих пір постійно оновлюється і отримує нові можливості для розробки [10]. На даний момент список його основних можливостей містить в собі наступне:

- Робота з файлами.

Усі здібності юнітів в грі мають свої файли. Змінюючи їх, можна просто редагувати та створювати нові здібності.

Кожен файл бази даних стандартної гри відкритий для модифікування, редактор тригерів дозволяє створювати свої власні функції та бібліотеки.

Карти створенні в цьому редакторі мають можливість містити в собі тисячі різних об’єктів, тригерів та локацій.

Існує механіка збереження імені творця карти, для зберігання авторського права на неї за ним.

- Графіка

Редактор підтримує створення власного клімату, освітлення та кольору тексту для повідомлень.

- Ландшафт

Існують готові набори елементів ландшафту як наприклад Пустеля, Вулкан, Джунглі, Пустоші, Руїни, тощо.

Елементи ландшафту з цих наборів можуть бути вільно використані разом, та довільно змінені для відображення конкретних потреб. Маленькі об’єкти такі як каміння, кущі, та інші можуть бути додані на будь-який ландшафт.

Дозволена зміна рівня води, що дозволяє робити затоплені частини ландшафту, річки та озера саме такими як потрібно.

- Юніти

Усі юніти що наявні в стандартній грі, та багато додаткових можуть бути вільно використані у створених картах. Вони можуть мати довільну кількість спец-можливостей, та рости у рівнях отримуючи досвід у боях.

Також певні «геройські» юніти можуть носити на собі предмети, що підчищують їх атрибути, такі як запас здоров’я, рівень атаки, тощо.

- Інтерфейс

За допомогою редактора можна змінювати інтерфейс та «обгортку» гри, спливаючі вікна для завдань, місій та діалогів.

Основним завданням редактора все ж є створення ігор з видом зверху, але за його допомогою можна зробити аркадний боковий платформер чи шутер від третього лиця.

- Мова Galaxy

Це скріптова мова, основана на C, але користувачі, які мають малий досвід програмування можуть використовувати зручний і зрозумілий редактор тригерів.

Мова не є об’єктно орієнтованою і більшість її функціоналу основана на модифікування внутрішньо-ігрових об’єктів.

Тригери можуть «спілкуватись» один з одним, передаючи потрібну інформацію. Можна створювати власні функції для дій.

Мова Galaxy також використовує власну систему збору сміття яка не дає утворюватися витікам пам’яті.

3.3 Розробка загальної архітектури програми

Розробка системи починається із аналізу технічного завдання, тобто дослідження чіткого сформованих вимог до системи, що розробляється. Для цього слід визначити які характеристики повинні бути притаманними додатку, сформулювати функціональні можливості, архітектуру розробки та ін. Необхідно визначити, які модулі повинна система в себе включати та яким чином вона має працювати.

Тактично стратегічна гра в реальному часі “DPS Log” призначена для практичної демонстрації роботи розробленого методу керування поведінкою ворожих юнітів у грі з використанням матриць небезпеки. Одними з найважливіших модулів у розробці є модулі знаходження шляху та прийняття рішень, так як у них відображається реалізація розробленого методу.

Для розробки потрібно пройти такі етапи:

- розбити систему на менші відносно автономні частини;
- для кожної з них виділити основне призначення та скласти блок-схему;
- визначити зв’язки між модулями програми;
- розробити програмну реалізацію кожної частини;

- розробити загальну структурну схему;
- керуючись схемою зв’язності між модулями, створити цілісний програмний продукт із інтуїтивно зрозумілим інтерфейсом;
- перевірити розробку на відповідність поставлений перед розробкою вимогам;
- протестувати систему на різних пристроях.

Програмний продукт повинен бути реалізований у формі повноцінної стратегічної гри, для того, щоб продукт міг в повній мірі демонструвати роботу розробленого методу керування поведінкою у різних стратегічних ситуаціях.

Для подальшого проектування гри «DPS Log», є необхідність сформувати її загальну архітектуру за допомогою діаграм варіантів використання, взаємодії об’єктів гри за часом, діаграми класів.

Діаграма варіантів використання гри представлена на рисунку 3.1.

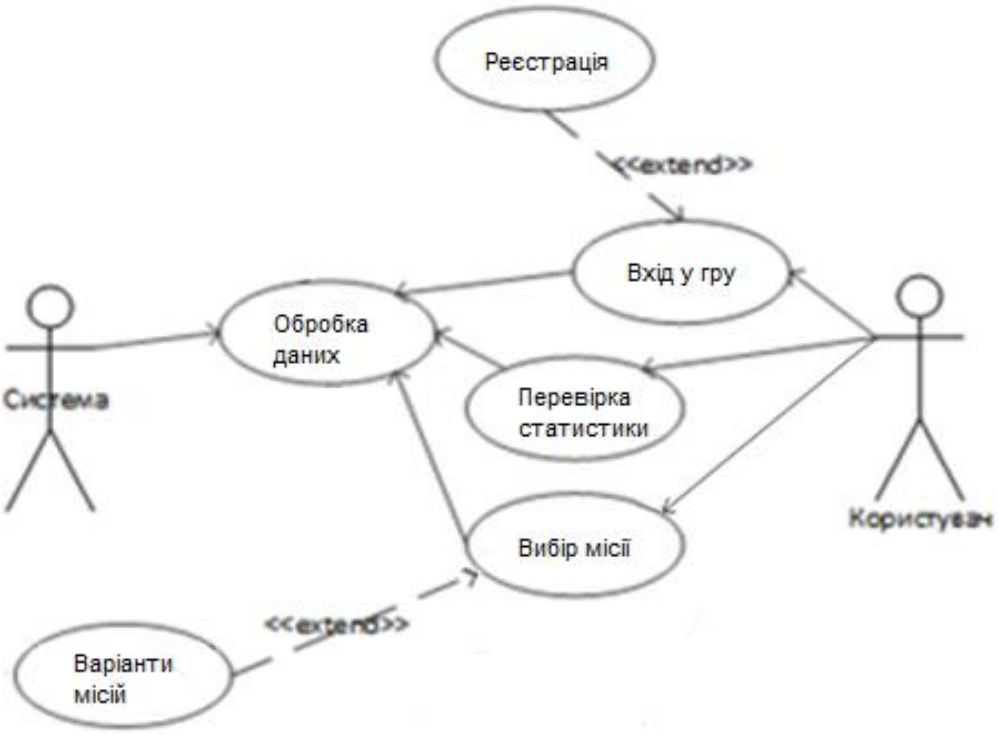


Рисунок 3.1 – Діаграма варіантів використання

Наявні 2 актори – система та користувач, які виконують властиві для них прецеденти. Особливістю є те, що система здійснює оброблення всіх даних, в залежності від вибору певних дій користувачем.

Взаємодія об'єктів гри за часом представлено на діаграмі послідовності на рисунку 3.2.

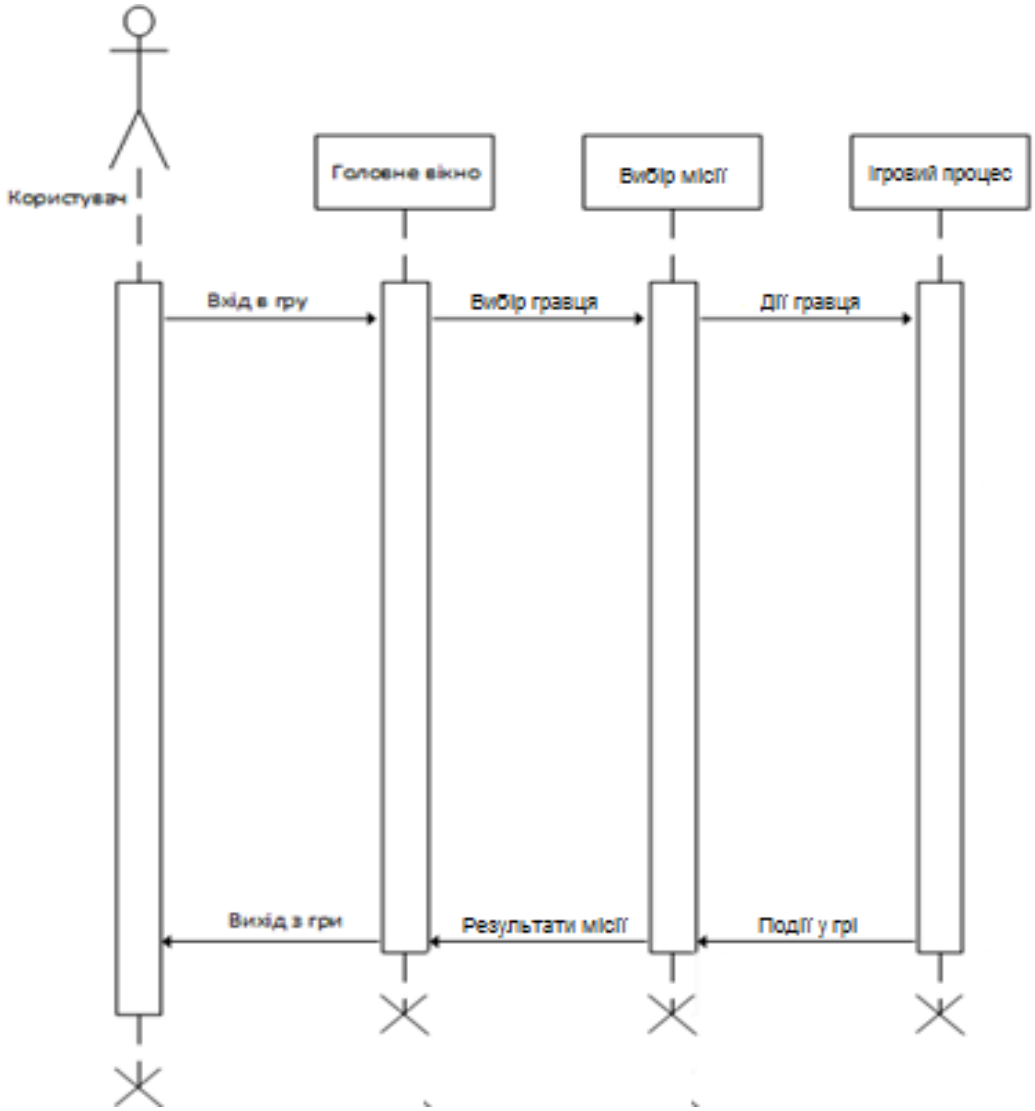


Рисунок 3.2 – Діаграма послідовності

Об'єктами гри є «Користувач», «Головне вікно» та місії, які може вибрати для проходження користувач. Продемонстровано послідовність взаємодії користувача з додатком у часі. Спочатку користувач логується, щоб перейти в головне вікно додатку, а лише потім обирає бажану місію.

Структурна карта Константайна є моделлю відносин ієрархії між програмними модулями [11]. Вона представляє собою взаємодію модулів додатку «DPS Log» і зображена на рисунку 3.3 у вигляді графу. Вершини – модулі, дуги – міжмодульні зв’язки [11]. Головним модулем є модуль інтерфейс. Він пов’язаний зв’язком по управлінню з модулями візуалізації та модулем керування програми. Модуль керування виконує підсистему – «Вибір місії» та функцію перевірки статистики користувача. Кожна з підсистем взаємодіє з модулем візуалізації зв’язком по даних, оскільки відповідний модуль використовує дані як для відображення відповідних компонентів підсистем, так і дані результату їх виконання.

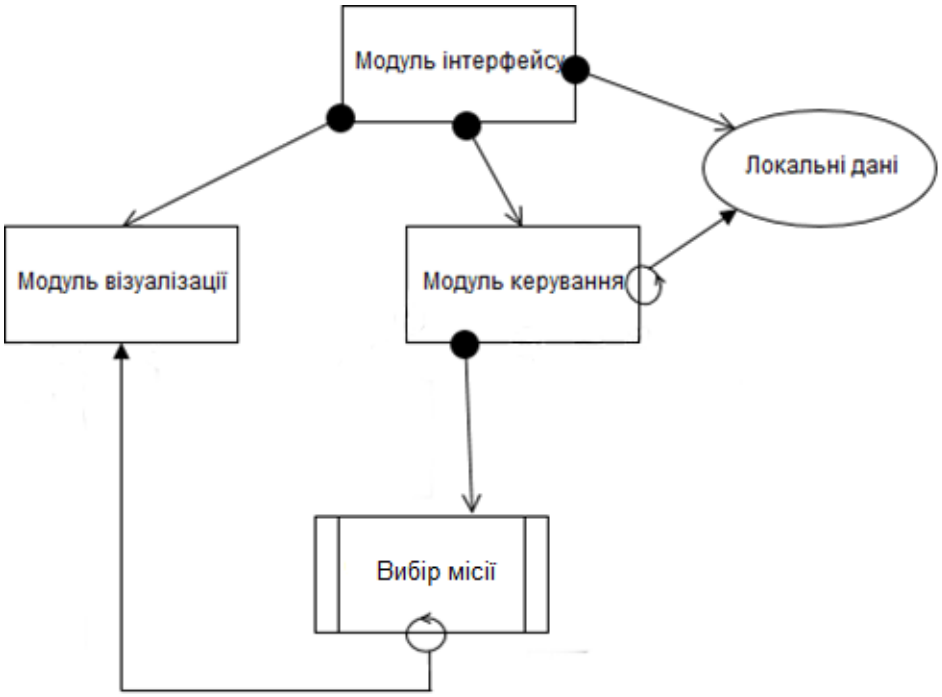


Рисунок 3.3 – Структурна карта Константайна додатку «DPS Log»

Діаграмі класів наявних у грі юнітів зображена на рисунку 3.4. Кожен з класів має ряд своїх полів та властивостей.

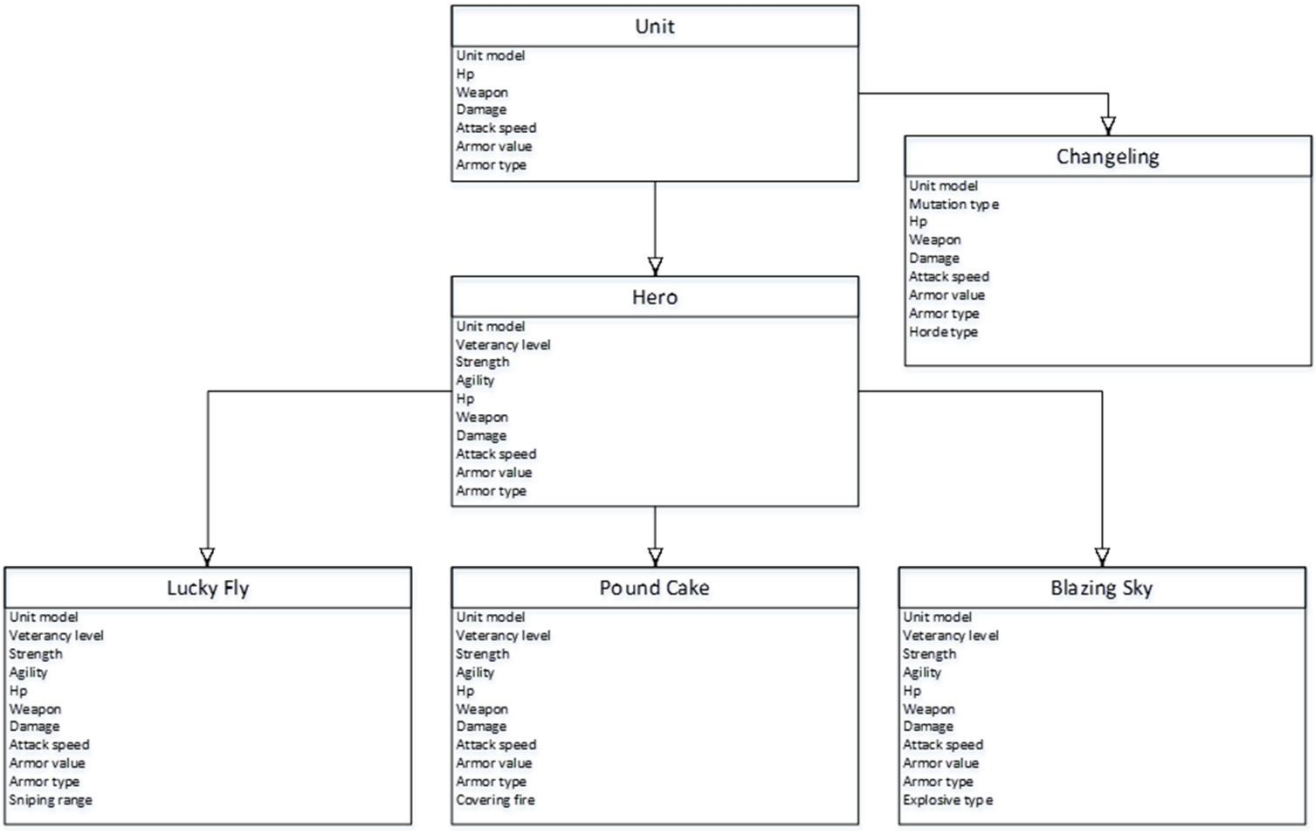


Рисунок 3.4 – Діаграма класів

Головним класом є «Unit», в якому задається загальна основа ігрових юнітів.

Клас «Hero» описує характеристику геройських юнітів гравця. До класу «Unit» знаходиться в відношенні наслідування.

Клас «Changeling» описує характеристику ворожих юнітів, якими керує штучний інтелект гри, і які будуть розставлені по всій карті місії. До класу «Unit» знаходиться в відношенні наслідування.

Клас «Lucky Fly» описує безпосередньо геройського юніту снайпера, роль якого прицільне знищення конкретних важливих юнітів противника. До класу «Hero» знаходиться в відношенні наслідування.

Клас «Pound Cake» описує безпосередньо геройського юніту солдата, універсального юніта який може виконувати роль основної наступальної сили так само як і обороняти стратегічно важливі позиції. До класу «Hero» знаходиться в відношенні наслідування.

Клас «Blazing Sky» описує безпосередньо геройського юніту підривника інженера, роль якого — знищення загород і стін супротивника, прорив оборони і нанесення шкоди по площі та мінування шляхів відступу. До класу «Hero» знаходиться в відношенні наслідування..

3.4 Аналіз інструментальних засобів для створення інтерфейсу та розробка інтерфейсу гри

Ігровий інтерфейс – це змога управління персонажем, взаємодія персонажів один з одним, спілкування гравців між собою і т.д. Практично всі ігри мають складний інтерфейс, що дозволяє управляти персонажами за допомогою різних способів – мишкою, клавіатурними кнопками, віртуальними кнопками на екрані. Ігри, де користувач за допомогою різних способів взаємодіє з об’єктами ігрового світу, набирає все більшої популярності. Більшість дій персонажів гри реалізуються стандартними способами, однаковими для всіх ігор. Часто гравець може змінити налаштування інтерфейсу так, щоб йому було більш звично і зручно. У той же час з використанням сенсорних екранів з’явилися і нові способи управління за допомогою рухів пальців [12].

Детальний опис інтерфейсу допомагає користувачу освоїтися в грі і максимально швидко отримати доступ до інформації, що його цікавить, та ігрових налаштувань. Для реалізації гри «DPS Log» прийнято рішення реалізовувати інтерфейс який може надати гравцю можливість керування юнітами в повній мірі, та контролювати ситуацію на віртуальному полі бою.

Для створення інтерфейсів використовуються багато різних мов розмітки наприклад HTML чи Javascript. Але враховуючи те, що додаток розроблюється на основі готового двигуна, яким є Galaxy Engine, для розробки користувацького інтерфейсу потрібно використовувати ті технології які використовуються в цьому двигуні, тобто мову розмітки XML.

HTML – стандартна мова розмітки веб-сторінок в Інтернеті. Документ HTML оброблюється браузером та відтворюється на екрані у звичному для

людини вигляді. HTML є похідною мовою від SGML, успадкувавши від неї визначення типу документа та ідеологію структурної розмітки тексту.

Для підвищення ефективності роботи з HTML використовуються препроцесорні мови які значно підвищують читабельність та структурованість коду, наприклад HAML чи SCSS.

Javascript [13] – динамічна, об'єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

Мова розмітки XML (Extensible Markup Language) це підмножина SGML, причому любий дійсний документ XML є дійсним документом SGML. І, як і SGML, XML - це метамова, що визначає інші мови розмітки для специфічних цілей [14]. Наприклад, мова синхронізованої інтеграції мультимедіа (Synchronized Multimedia Integration Language, SMIL) базується на XML.

XML дозволяє визначити формальний синтаксис мови, наприклад, правила вкладення елементів. Семантику можна описувати на звичайній англійській мові.

Ця мова використовується для розмітки стандартних документів багато в чому так само, як HTML. Як наслідок документ XML виглядає багато в чому схожим на HTML. Він також складається з текстових фрагментів, анотованих вкладеними в кутові дужки тегами. Проте, на відміну від HTML, зміст тега залежить від регістра, а кожний відкриваючий тег повинний в усіх випадках мати парний закриваючий тег.

XML це мова розмітки, що описує цілий клас об'єктів даних, називаних XML- документами. Ця мова використовується в якості засобу для опису грамматики інших мов і контролю за правильністю впорядкування документів. XML не містить ніяких тегів, призначених для розмітки, а просто визначає порядок їх створення.

Таким чином, у розробників з'являється унікальна можливість визначати власні команди, що дозволяють їм найбільш ефективно визначати дані, що зберігаються в документі. Автор документа створює його структуру, будує необхідні зв'язки між елементами, використовуючи ті команди, що задовольняють його вимогам і домагається такого типу розмітки, що необхідно йому для виконання операцій перегляду, пошуку, аналізу документа.

Не обмежуючи автора яким-небудь фіксованим набором тегів, XML дозволяє йому вводити будь-які імена. Ця можливість є ключовою для активного маніпулювання даними.

На основі XML уже сьогодні створені такі відомі спеціалізовані мови розмітки, як SMIL, CDF, MathML, XSL, і список робочих проектів нових мов, що знаходяться на розгляді W3C, постійно поповнюється.

Для проектування гри необхідним є створення графічних схем інтерфейсу. Розглядаємо проектування інтерфейсу головного меню гри, та ігрового процесу.

Структурна схема інтерфейсу гри «DPS Log» зображена на рисунку 3.5. Інтерфейс гри складається з «MainMenu» – головне вікно програми, що містить меню з кнопками початку гри та опціями, та «Gameplay» що містить інтерфейс самого ігрового процесу.

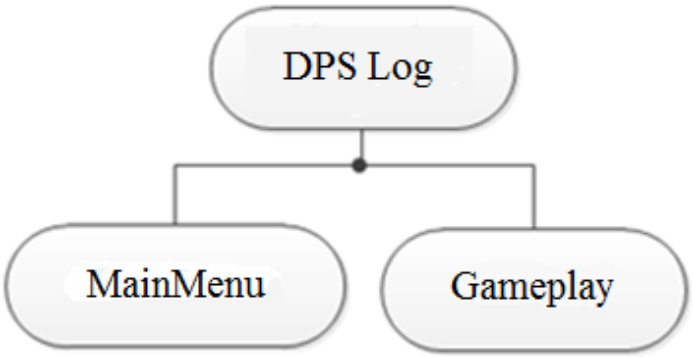


Рисунок 3.5 – Структурна схема інтерфейсу гри «DPS Log»

Графічна структурну схема головної сторінки гри «DPS Log» зображена на рисунку 3.6.

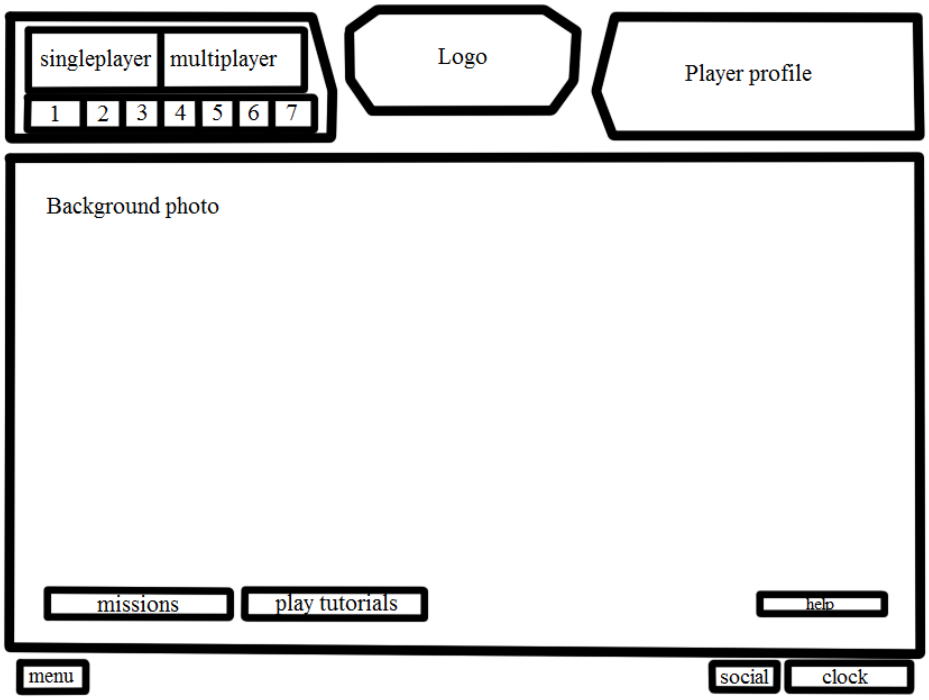


Рисунок 3.6 – Графічна структурна схема інтерфейсу головного сторінки

Головне вікно програми складається з таких елементів:

1. Кнопка Singleplayer – кнопка для відкриття режиму на одного гравця.
2. Кнопка Multiplayer – кнопка для відкриття режиму на одного гравця.
3. Logo – Логотип гри.
4. Player profile – Основна інформація з профіля гравця.
5. Background photo - Фонове зображення.
6. Кнопка Missions – кнопка для переходу у меню вибору місій.
7. Кнопка Play tutorials – кнопка для відкриття навчальної місії для нових гравців.
8. Кнопка Help – кнопка для вікна допомоги.
9. Кнопка Menu – кнопка для відкриття головного меню лаунчера.
10. Кнопка Social – кнопка для відкриття списку друзів у лаунчері.
11. Clock - Годинник.
12. Кнопка Home (на рисунку зображено як «1») – кнопка для відкриття головного меню.

13. Кнопка Profile (на рисунку зображено як «2») – кнопка для відкриття профіля гравця, де можна перевірити його статистику
14. Кнопка Achievements (на рисунку зображено як «3») – кнопка для відкриття списку ігрових досягнень гравця.
15. Кнопка Ladder top (на рисунку зображено як «4») – кнопка для відкриття поточного стану рейтингу гравців за тиждень.
16. Кнопка Replays (на рисунку зображено як «5») – кнопка для відкриття меню перегляду записів проходження попередніх місій.
17. Кнопка Global rank (на рисунку зображено як «6») – кнопка для відкриття глобального рейтингу гравців.
18. Кнопка Info (на рисунку зображено як «7») – кнопка для відкриття вікна з додатковою інформацією.

Якщо користувач увійшов у додаток під своїми даними він може редагувати свої дані, замінити фотографію профілю, змінити пароль та переглядати свою статистику.

Графічна схема інтерфейсу ігрового процесу зображено на рисунку 3.7.

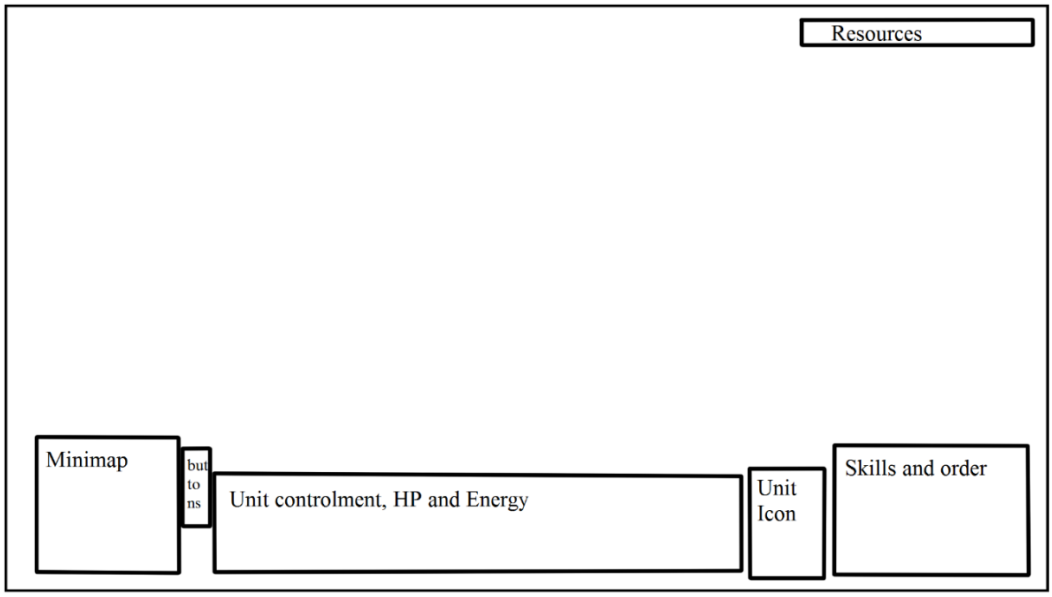


Рисунок 3.7 – Графічна структурна схема інтерфейсу ігрового процесу

Вікно ігрового процесу складається з таких елементів:

1. Resources - Показник ресурсів – інформаційний рядок з кількістю обмежених ресурсів гравця.
2. Minimap – Мінікарта, де відображається позиція гравця, виявленні ним ворожі одиниці та схематичний план місцевості.
3. Buttons - Кнопки управління картою, зменшення, збільшення, відключення маркерів позиції гравця.
4. Unit controlment, HP and Energy - Панель для відображення здоров'я, енергії, характеристик юніта.
5. Unit Icon - Іконка юніта.
6. Skills and order – Список спец-умінь та наказів для юніта.
7. Решта екрану відведена для відображення поля бою.

Таким чином була створена структура графічного інтерфейсу програми.

3.5 Програмна реалізація

Для реалізації методу керування поведінкою ворожих персонажів з використанням матриці небезпеки, спочатку потрібно підготовити стандартну мапу для змін в редакторі Galaxy Editor. Для цього потрібно створити клас DPSLMap та внести в нього потрібні зміни. Створення об'єкту мапи показано на рисунку 3.8..

```
DPSLMap::DPSLMap( string toolPathIn,
                  string outputPathIn,
                  string argPathIn,
                  string archiveNameIn,
                  string archiveWithExtIn )
{
  toolPath       = toolPathIn;
  outputPath     = outputPathIn;
  argPath        = argPathIn;
  archiveName    = archiveNameIn;
  archiveWithExt = archiveWithExtIn;

  mapHeight       = NULL;
  mapCliffChanges = NULL;
  mapPathing      = NULL;
  mapOpenness     = NULL;
  mapOpennessPrev = NULL;
  mapPathNodes    = NULL;

  totalMinerals     = 0.0f;
  totalVespeneGas   = 0.0f;
  totalHYMinerals   = 0.0f;
  totalHYVespeneGas = 0.0f;
}
```

Рисунок 3.8 – Програмний код класу DPSLMap

Після визначення карти, потрібно виконати функцію DPSLMapChanger для виставлення потрібних додаткових атрибутів у мапу, які дозволять використання розширених скриптів, на яких базується принцип роботи матриці небезпеки. Код функції показаний на рисунку 3.9.

```
void DPSLMapChanger::buildOutputColumns()
{
  addColumn( "Archive",                "%s",     offsetof( DPSLMapSummary, archiveName         ), COLTYPE_STR );
  addColumn( "Name In Output Files",   "%s",     offsetof( DPSLMapSummary, fileName            ), COLTYPE_STR );
  addColumn( "Map",                    "%s",     offsetof( DPSLMapSummary, mapName             ), COLTYPE_STR );
  addColumn( "Playable Size",          "%s",     offsetof( DPSLMapSummary, playableSize        ), COLTYPE_STR );
  addColumn( "% Pathable",             "%.0f%%", offsetof( DPSLMapSummary, percentPlayableCellsPathable ), COLTYPE_FLOAT );
  addColumn( "Num Start Locs",         "%d",     offsetof( DPSLMapSummary, numStartLocs        ), COLTYPE_INT );
  addColumn( "Num Bases (inc. StrtLc)", "%d",    offsetof( DPSLMapSummary, numBasesIncludingSL ), COLTYPE_INT );
  addColumn( "Avg Resources/Base",     "%.1fK",  offsetof( DPSLMapSummary, avgResourcesPerBase ), COLTYPE_FLOAT );
  addColumn( "Total Minerals",         "%.1fK",  offsetof( DPSLMapSummary, totalMinerals       ), COLTYPE_FLOAT );
  addColumn( "% Min HY",               "%.0f%%", offsetof( DPSLMapSummary, percentMineralsHY   ), COLTYPE_FLOAT );
  addColumn( "Total Vespene Gas",      "%.1fK",  offsetof( DPSLMapSummary, totalVespeneGas     ), COLTYPE_FLOAT );
  addColumn( "% Gas HY",               "%.0f%%", offsetof( DPSLMapSummary, percentVespeneGasHY ), COLTYPE_FLOAT );

  addColumn( "Min Ground Distance Main-to-Main",     "%.1f", offsetof( DPSLMapSummary, minGroundDistanceMain2Main ), COLTYPE_FLOAT );
  addColumn( "Avg Ground Distance Main-to-Main",     "%.1f", offsetof( DPSLMapSummary, avgGroundDistanceMain2Main ), COLTYPE_FLOAT );
  addColumn( "Min Cliff-Walk Distance Main-to-Main", "%.1f", offsetof( DPSLMapSummary, minCWalkDistanceMain2Main  ), COLTYPE_FLOAT );
  addColumn( "Avg Cliff-Walk Distance Main-to-Main", "%.1f", offsetof( DPSLMapSummary, avgCWalkDistanceMain2Main  ), COLTYPE_FLOAT );
  addColumn( "Min Air Distance Main-to-Main",        "%.1f", offsetof( DPSLMapSummary, minAirDistanceMain2Main    ), COLTYPE_FLOAT );
  addColumn( "Avg Air Distance Main-to-Main",        "%.1f", offsetof( DPSLMapSummary, avgAirDistanceMain2Main    ), COLTYPE_FLOAT );

  addColumn( "Min Ground Distance Nat-to-Nat",     "%.1f", offsetof( DPSLMapSummary, minGroundDistanceNat2Nat ), COLTYPE_FLOAT );
  addColumn( "Avg Ground Distance Nat-to-Nat",     "%.1f", offsetof( DPSLMapSummary, avgGroundDistanceNat2Nat ), COLTYPE_FLOAT );
  addColumn( "Min Cliff-Walk Distance Nat-to-Nat", "%.1f", offsetof( DPSLMapSummary, minCWalkDistanceNat2Nat  ), COLTYPE_FLOAT );
  addColumn( "Avg Cliff-Walk Distance Nat-to-Nat", "%.1f", offsetof( DPSLMapSummary, avgCWalkDistanceNat2Nat  ), COLTYPE_FLOAT );
  addColumn( "Min Air Distance Nat-to-Nat",        "%.1f", offsetof( DPSLMapSummary, minAirDistanceNat2Nat    ), COLTYPE_FLOAT );
  addColumn( "Avg Air Distance Nat-to-Nat",        "%.1f", offsetof( DPSLMapSummary, avgAirDistanceNat2Nat    ), COLTYPE_FLOAT );

  addColumn( "Max Openness", "%.2f", offsetof( DPSLMapSummary, maxOpenness ), COLTYPE_FLOAT );
  addColumn( "Avg Openness", "%.2f", offsetof( DPSLMapSummary, averageOpennessPathableCells ), COLTYPE_FLOAT );

  addColumn( "Watchtower Coverage of Pathable Cells", "%.1f%%", offsetof( DPSLMapSummary, watchtowerCoverage ), COLTYPE_FLOAT );

  addColumn( "% Positional Balance", "%.1f%%", offsetof( DPSLMapSummary, positionalBalancePercentage ), COLTYPE_FLOAT );

  //addColumn( "", "", offsetof( DPSLMapSummary,  ), COLTYPE_ );
}
```

Рисунок 3.9 – Програмний код функції DPSLMapChanger

Після того як було виконано роботу по підготовці мапи для інтеграції матриці небезпеки, потрібно реалізувати метод DPSLMapAnalyzer. Він виконує дві функції: перевіряє чи правильно виконані попередні кроки роботи програми, та чи здатна відредагована мапа прийняти додатковий шар у вигляді матриці небезпеки та ініціює перші статичні зміні що будуть використовуватись матрицею небезпеки в подальшому. Цей крок особливо важливий для усього процесу роботи програми. Уривок програмного коду методу показано на рисунку 3.10.

```
if( !DPSLmap->configUserLocal.outputPath.empty() )
{
  struct stat dirStatus;
  if( stat( DPSLmap->configUserLocal.outputPath.data(), &dirStatus ) >= 0 )
  {
    if( S_ISDIR( dirStatus.st_mode ) )
    {
      DPSLmap->outputPath = DPSLmap->configUserLocal.outputPath;
    }
  }
}

DangerMatrixInitiateTerrain          = DPSLmap->getOutputOption( "DangerMatrixInitiateTerrain"          );
DangerMatrixInitiatePathing          = DPSLmap->getOutputOption( "DangerMatrixInitiatePathing"          );
DangerMatrixInitiateBases            = DPSLmap->getOutputOption( "DangerMatrixInitiateBases"            );
DangerMatrixInitiateOpenness         = DPSLmap->getOutputOption( "DangerMatrixInitiateOpenness"         );
DangerMatrixInitiateShortest         = DPSLmap->getOutputOption( "DangerMatrixInitiateShortest"         );
DangerMatrixInitiateInfluence        = DPSLmap->getOutputOption( "DangerMatrixInitiateInfluence"        );
DangerMatrixInitiateInfluenceHeatMap = DPSLmap->getOutputOption( "DangerMatrixInitiateInfluenceHeatMap" );
DangerMatrixInitiateSummary          = DPSLmap->getOutputOption( "DangerMatrixInitiateSummary"          );
writeCSVpermap         = DPSLmap->getOutputOption( "writeCSVpermap"         );

if( DPSLmap->readMap() < 0 )
{
  printWarning( "Could not read required map files for %s, skipping\n\n", file.data() );
  return;
}

printMessage( "Prepping analysis,\n" );

DPSLmap->countPathableCells();

printMessage( "." );

DPSLmap->prepShortestPaths();

printMessage( "." );

DPSLmap->identifyBases();

printMessage( "." );

DPSLmap->computeOpenness();

printMessage( "." );

DPSLmap->locateChokes();

printMessage( "." );
```

Рисунок 3.10 – Програмний код функції DPSLMapAnalyzer

Останнім виконується функція створення матриці небезпеки, вона обраховує первинні атрибути, установлює зв’язки з координатною сіткою, запускає фонові процеси очікування вводу гравця для обчислень змін. Фрагмент програмного коду функції DPSLDangerMatrix показано на рисунку 3.11

```
const float point::MatrixNaN  = -987.654321f;
const int   point::MatrixtNaN = -12345;
const int   point::MatrixcNaN = -23456;
const int   point::ptNaN = -34567;
const int   point::pcNaN = -45678;


void point::set( point* p )
{
  Matrixx  = p->Matrixx;  Matrixy  = p->Matrixy;
  Matrixtx = p->Matrixtx; Matrixty = p->Matrixty;
  Matrixcx = p->Matrixcx; Matrixcy = p->Matrixcy;
  ptx = p->ptx; pty = p->pty;
  pcx = p->pcx; pcy = p->pcy;
  ix  = p->ix;  iy  = p->iy;
}

void point::MatrixSet( float MatrixxIn, float MatrixyIn )
{
  Matrixx  = MatrixxIn;           Matrixy  = MatrixyIn;
  Matrixtx = Matrix2Matrixt  ( MatrixxIn ); Matrixty = Matrix2Matrixt  ( MatrixyIn );
  Matrixcx = Matrix2Matrixc  ( MatrixxIn ); Matrixcy = Matrix2Matrixc  ( MatrixyIn );
  ptx = Matrixx2ptx( MatrixxIn ); pty = Matrixy2pty( MatrixyIn );
  pcx = Matrixx2pcx( MatrixxIn ); pcy = Matrixy2pcy( MatrixyIn );
  ix  = Matrixx2ix ( MatrixxIn ); iy  = Matrixy2iy ( MatrixyIn );
}

void point::MatrixtSet( int MatrixtxIn, int MatrixtyIn )
{
  Matrixx  = Matrixt2Matrix   ( MatrixtxIn ); Matrixy  = Matrixt2Matrix   ( MatrixtyIn );
  Matrixtx = MatrixtxIn;            Matrixty = MatrixtyIn;
  Matrixcx = MatrixcNaN;            Matrixcy = MatrixcNaN;
  ptx = Matrixtx2ptx( MatrixtxIn ); pty = Matrixty2pty( MatrixtyIn );
  pcx = pcNaN;            pcy = pcNaN;
  ix  = Matrixtx2ix ( MatrixtxIn ); iy  = Matrixty2iy ( MatrixtyIn );
}
```

Рисунок 3.11 – Фрагмент програмний коду функції DPSLDangerMatrix

3.6 Висновки

У третьому розділі було проведено варіантний аналіз та вибір засобів розробки та інтегрованого середовища розробки, а також розроблено архітектуру програмної системи. Було розроблено діаграми варіантів, послідовності і класів. Створенно структуру графічного інтерфейсу. Було розроблено програмні модулі, описано структуру компонентів, що забезпечують створення та налаштування мапи, ініціацію та процес роботи матриці небезпеки.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вибір методики тестування програмної системи

Тестування програмного забезпечення – це:

1. Процес дослідження ПЗ з метою отримання інформації про якість продукту;

2. Процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснений шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином;

3. Оцінка системи для того, щоб знайти відмінності між тим, якою система повинна бути і якою вона є.

У широкому сенсі, тестування – це одна з технік контролю якості (Quality Control), яка включає планування, складання тестів, безпосередньо виконання тестування і аналіз отриманих результатів [15].

Важливо розуміти, що тестування ПЗ включає в себе не тільки проведення тестів, але і багато інших дій, пов’язаних з процесом забезпечення якості:

1. Аналіз і планування.
2. Розробку тестових сценаріїв.
3. Оцінку критеріїв закінчення тестування.
4. Написання звітів.
5. Рецензування документації (в тому числі і вихідного коду).
6. Проведення статичного аналізу.

Розглянемо наступні методи тестування: «біла скринька» і «чорна скринька».

Метод тестування «біла скринька» виконується розробником (так як необхідно знати внутрішні принципи роботи програми) для перевірки внутрішньої структури програмного засобу. Об’єктом тестування є дані, отримані шляхом аналізу логіки програми. Перевіряється коректність побудови

всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше – інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми. Зазвичай тестування «білої скриньки» засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління [15].

Принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

1. Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

2. Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

3. Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

4. При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

Метод тестування «чорна скринька» використовується, коли тестувальнику не потрібно знати внутрішні властивості програми. При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок [16].

Тести демонструють:

1. Як виконуються функції програми.
2. Як приймаються вихідні дані.
3. Як виробляються результати.
4. Як зберігається цілісність зовнішньої інформації.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

1. Некоректних чи відсутніх функцій.
2. Помилок інтерфейсу.
3. Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.
4. Помилок характеристик (необхідна ємність пам'яті і т. д.).
5. Помилок ініціалізації та завершення.

На основі розглянутих методів існує також тестування «сіра скринька». При роботі з даним методом тестувальник має доступ до коду програми, проте тестування проводить з точки зору кінцевого користувача. Суть даних методів не є складною, проте ефективність тестування за допомогою кожного з них вимагає хороших знань та навичок [16]. В результаті, обрано метод тестування «чорна скринька», оскільки за даним методом тестуються лише вхідні/вихідні дані.

Оскільки розроблено комп’ютерну гру, доцільним є провести тестування графічного інтерфейсу користувача для забезпечення його відповідності до даної специфікації. Завданням тестування графічного інтерфейсу користувача є виявлення помилок наступного характеру:

1. Помилки у функціональності за допомогою інтерфейсу.
2. Необроблені виключення при взаємодії з інтерфейсом.
3. Втрата або перекручування даних, переданих через елементи інтерфейсу.

4. Помилки в інтерфейсі (невідповідність проектної документації, відсутність елементів інтерфейсу).

Важливим для тестування інтерфейсу користувача є оцінка його зручності. Зручність використання графічного інтерфейсу виявляє міру простоти доступу користувача до функцій системи, наданих через інтерфейс програми. Тестування зручності використання інтерфейсу не належить до класичного методу тестування програмних систем. Фахівець по тестуванню користувальницького інтерфейсу має поєднувати у собі знання в галузі як програмної інженерії, так і фізіології, психології та ергономіки [16].

На зручність використання користувальницького інтерфейсу впливають такі чинники:

1. Легкість навчання – чи швидко людина навчається використовувати систему.
2. Ефективність навчання – чи швидко людина працює після навчання.
3. Помилки – чи часто людина припускається помилок у роботі.
4. Загальна задоволеність – чи є загальне враження від співпраці з системою позитивним.

Усі ці фактори, незважаючи на неформальність, можуть бути виміряні. Для таких вимірів вибирається група типових користувачів. Під час їх роботи вимірюються показники роботи системи, і їм пропонується висловити враження від інтерфейсу за допомогою заповнення опитувальних аркушівТестування програмного забезпечення –процес дослідження програмного продукту, що має на меті виявлення інформації про якість продукту стосовно контексту його використання. Процес тестування означає як пошук помилок або інших дефектів, так і випробування програмного забезпечення для його оцінки. Критерії оцінювання:

1. Відповідність програмного продукту технічному завданню.
2. Коректність отриманих результатів для будь-яких дозволених вхідних даних.

3. Відповідність вимогам, що стосуються споживання системних ресурсів.

4. Відповідність вимогам, що стосуються сумісності з програмним забезпеченням та операційними системами.

Зважаючи на факт того, що кількість можливих тестів навіть для примітивних програмних компонент є дуже великою, стратегія тестування полягає в тому, щоб максимізувати покриття програмного продукту тестами, витративши на тестування прийнятну кількість часу.

Існує багато видів тестування: одні зазвичай виконують самі розробники, а інші – спеціалізовані групи. Деякі види тестування перераховані нижче [17]:

1. Модульне тестування (unit testing).
2. Інтеграційне тестування.
3. Регресивне тестування.
4. Системне тестування.

Існує два основних методи тестування: тестування «білої скриньки», тестування «чорної скриньки» [18].

Перший вид тестування має на меті з'ясування обставин, в яких поведінка програми не відповідає її специфікації. Тестові ж дані використовуються тільки у відповідності зі специфікацією програми (без урахування знань про її внутрішню структуру).

Стратегія «білої скриньки», або стратегія тестування, керованого логікою програми, дозволяє досліджувати внутрішню структуру програми. В цьому випадку QA-інженер отримує тестові дані шляхом аналізу логіки програми. При використанні даної стратегії часто не використовується специфікація програми.

Стратегія «білої скриньки» надлишкова, оскільки існує специфікація вимог до системи. Необхідність отримання інформації шляхом аналізу програмного коду виникає лише у разі відсутності специфікації та особи, відповідальної за опис вимог до програмного продукту – що трапляється у

випадку необхідності підтримувати так званий «legacy code» – програмний код, що був розроблений іншим розробником або групою розробників.

Для тестування програмного продукту було обраного стратегію «чорної скриньки».

4.2 Хід тестування

Тестування програмної системи проводиться за методикою «чорної скриньки». Вона базується на використанні шаблонів тестування (тест-кейсів). Це означає, що буде створено декілька тест-кейсів для перевірки правильності роботи основних функцій програмного додатку [19]. Розроблені для перевірки правильності роботи сайту тест-кейси описано в таблиці 4.1.

Таблиця 4.1

Тест-кейси

<table>
<tr><td rowspan="3">Код тест-кейса</td><td colspan="3">Опис тест-кейса</td></tr>
<tr><td colspan="2">Хід тестування</td><td>Очікуваний результат</td></tr>
<tr><td>Дата тестування</td><td>Результат</td><td>Примітка</td></tr>
<tr><td rowspan="3">001</td><td colspan="3">Тест відкриття гри та завантаження</td></tr>
<tr><td colspan="2">1. Відкрити гру Starcraft 2.
2. Зайти у меню «Custom Games»
3. Вибрати DPS Log.
4. Натиснути «Старт»</td><td>Починається завантаження модуля гри «DPS Log»
Перед гравцем екран завантаження</td></tr>
<tr><td>11.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td colspan="4"> 
</td></tr>
</table>

Продовження таблиці 4.1

<table>
<tr><td rowspan="3">1</td><td colspan="3">2</td></tr>
<tr><td colspan="2">3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td rowspan="3">002</td><td colspan="3">Тест вибору місії та меню спорядження</td></tr>
<tr><td colspan="2">1. Війти у меню вибора місій
2. Обрати першу місію та запустити її.
3. Відкрити меню спорядження та обрати комплект.</td><td>Користувач залишається авторизованим у грі, відкривається редактор спорядження перед місією</td></tr>
<tr><td>11.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td colspan="4"> 
</td></tr>
<tr><td rowspan="3">003</td><td colspan="3">Тест завантаження місії</td></tr>
<tr><td colspan="2">1. Повторити кроки, описані у тесті 002.
2. Натиснути на кнопку «Старт».
3. Зачекати протягом процесу завантаження.
4. Обрати стартового персонажа</td><td>Місія і бойова мапа завантажилась.
Гравець має контроль на ігровими персонажами.</td></tr>
<tr><td>11.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td colspan="4">  </td></tr>
</table>

Продовження таблиці 4.1

<table>
<tr><td rowspan="3">1</td><td colspan="3">2</td></tr>
<tr><td colspan="2">3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td rowspan="3">004</td><td colspan="3">Тест роботи полів видимості</td></tr>
<tr><td colspan="2">1. Повторити кроки, описані у тесті 001.
2. Зайти у гру в режимі розробника.
3. Включити графічне відображення зон небезпеки</td><td>Поля видимості присутні та рухаються з персонажами.</td></tr>
<tr><td>12.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td colspan="4">  </td></tr>
<tr><td rowspan="3">005</td><td colspan="3">Тест роботи зон дії здібностей</td></tr>
<tr><td colspan="2">1. Повторити кроки, описані у тесті 001.
2. Зайти у гру в режимі розробника.
3. Включити графічне відображення зон небезпеки
4. Використати здібність «Ливень свинцю»</td><td>Зона дії здібності присутня, статична та існує протягом усієї тривалості використання.</td></tr>
<tr><td>12.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td colspan="4">  </td></tr>
</table>

Продовження таблиці 4.1

<table>
<tr><td rowspan="3">1</td><td colspan="3">2</td></tr>
<tr><td colspan="2">3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td rowspan="3">006</td><td colspan="3">Тест знаходження шляху ворожих віськ через велику кількість зон небезпеки</td></tr>
<tr><td colspan="2">1. Повторити кроки, описані у тесті 001.
2. Зайти у гру в режимі розробника.
3. Включити графічне відображення зон небезпеки
4. Створити велику кількість зон дії здібності «Міна» на шляху руху групи вороги</td><td>Після виявлення мін, юніти ворога змінять свій маршрут і будуть намагатись уникати заміновані ділянки мапи.</td></tr>
<tr><td>13.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td colspan="4">  </td></tr>
</table>

При тестуванні програми за методикою тестування «чорної скриньки» помилок не виявлено.

4.3 Розробка інструкції користувача

Оскільки гра «DPS Log» була розроблена за допомогою Galaxy Editor, для того щоб запустити гру, потрібно спочатку запустити лаунчер Battle.net, що пов’язаний з Galaxy Editor та авторизуватись у сервісі Battle.net, ввівши email та пароль. В разі введення некоретних даних з’явиться повідомлення про помилку. Меню авторизації зображене на рисунку 4.1.

Рисунок 4.1 – Авторизація у Battle.net

Після цього натискаємо кнопку «Custom Games» у головному меню та потрапляємо на список ігор і режимів. Обираємо DPS Log і натискаємо Join Game. (рис. 4.2).

Рисунок 4.2 – Вибір DPS Log зі списку режимів та ігор

У головному меню гри DPS Log, перед гравцем буде вибір між набором місій. Кожна з них має короткий опис та при запуску дозволяє вибрати обладнання для персонажів гравця (рис. 4.3).

Рисунок 4.3 – Вибір місії у меню гри

Після вибору місії, гравець завантажується на бойову мапу де йому показують список бойових задач які він повинен виконати, щоб місія вважалася успішно виконаною.

Протягом місії, гравець керує загоном із 3 персонажів, кожен з яких має унікальні характеристики та здібності. Втрата 1 чи 2 персонажів значно знижує кінцевий рейтинг виконання місії, втрата всіх трьох – закінчує місію і гравець переноситься в головне меню гри.

Отож, була написана інструкція користувача. В ній описана правильна послідовність дій для користувача, щоб досягнути необхідного результату.

4.4 Висновки

У четвертому розділі магістерської кваліфікаційної роботи було розглянуто види та методики тестування, обрано методику тестування «чорної скриньки», складено набір тест-кейсів для тестування програмного продукту, проведено тестування, а також розроблено інструкцію користувача, у якій наведено покроковий опис дій, необхідних для користування системи.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Романюк О.В. та Ракитянська Г.Б.

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою [20].

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1

Результати оцінювання комерційного потенціалу розробки

Критерій	Романюк О. В.	Ракитянська Г.Б.
	Бали, виставлені експертами	
1	4	4
2	4	3
3	3	4
4	4	3
5	3	3
6	4	4
7	4	3
8	3	4
9	4	3
10	4	4
11	3	3
12	4	4
Сума балів	СБ1 = 44	СБ2 = 42
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^2 СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідні роботи та впровадження результатів

Для розробки нового програмного продукту необхідні такі витрати. Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} * t, \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника;

$Т_р$ – кількість робочих днів у місяці, $Т_р$ – 21 день;

t – кількість днів роботи розробника, t = 45 днів.

Результати розрахунку заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2

Розрахунки основної заробітної плати

Працівник	Оклад М, грн	Оплата за робочий день, грн	Кількість днів роботи, t	Витрати на оплату праці, грн
Науковий керівник	5000	238,1	8	1904,8
Інженер-програміст	3500	166,66	45	7499,9
Всього				9404,7

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0,1 \cdot 9404,7 = 940,47 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп}= (З_о + З_р)\cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп}= (785,7+ 7857)\cdot \frac{36,3}{100} = 3755,3 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А=\frac{Ц\cdot Н_а}{100}\cdot\frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3

Розрахунок амортизаційних відрахувань

Найменування	Балансова вартість, грн	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
Персональний комп’ютер	9000	25	3	562,5
Всього				562,5

Розрахуємо витрати на комплектуючі. Для розрахунку витрат на комплектуючі використовується формула (5.4).

$$К = \sum_1^n Н_i * Ц_i * К_i, \quad (5.4)$$

де n – кількість комплектуючих;

$Н_і$ - кількість комплектуючих і-го виду;

$Ц_і$ – покупна ціна комплектуючих і-го виду, грн;

$К_і$ – коефіцієнт транспортних витрат (приймемо $К_і$ = 1,1)

Проведемо оцінку витрат на комплектуючі, що було використано для розробки програмного забезпечення. Результати занесено до таблиці 5.4.

Таблиця 5.4

Витрати на комплектуючі, що було використано для розробки програмного забезпечення

Найменування матеріалу	Одиниці виміру	Ціна, грн	Витрачено	Вартість витрачених матеріалів, грн
Флешка	шт.	150	1	150
Пачка паперу	уп.	100	1	100
Ручка	шт.	5	1	5
Всього з урахуванням транспортних витрат				280,5

Витрати на електроенергію розраховуються за формулою (5.5):

$$В_е = В * П * Ф * К_п, \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,66 грн/кВт);

П – установлена потужність комп'ютера (П = 0,6 кВт);

Ф – фактична кількість годин роботи комп'ютера (Ф = 200 год);

$К_п$ – коефіцієнт використання потужності ($К_п$ < 1, $К_п$ = 0,8).

Підставивши відомі дані в формулу (5.5), отримаємо величину витрат на електроенергію:

$$Ве = 1{,}66 \cdot 0{,}6 \cdot 200 \cdot 0{,}8 = 163{,}2 \text{ (грн)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $В_{ін}$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які виконували дану роботу. Розрахунок виконується за формулою (5.6):

$$В_{ін} = (1 \dots 3) * (З_о + З_{дод}) \quad (5.6)$$

Отже, розраховуємо інші витрати:

$$В_{ін} = 1 * (9404,7 + 940,47) = 10345,17 \text{ (грн)}$$

Сума всіх попередніх статей витрат дає витрати на виконання розробки програмного забезпечення, що обчислюється за формулою (5.7):

$$В = З_о + З_{дод} + Н_{зп} + А + К + В_е + В_{ін} \quad (5.7)$$

Підставивши обчислені раніше величини у формулу (5.7), отримаємо значення вартості розробки:

$$В = 9404,7 + 940,47 + 3755,3 + 562,5 + 280,5 + 163,2 + 10345,17 = 25481,54 \text{ (грн)}$$

Розрахуємо загальну вартість наукової роботи $В_{заг}$ за формулою (5.8):

$$В_{заг} = \frac{В}{\alpha}, \quad (5.8)$$

де α – частка витрат, які безпосередньо здійснює виконавець поточного етапу роботи, у відносних одиницях – приймаємо рівним 1.

$$В_{заг} = \frac{25481,54}{1} = 25481,54 \text{ (грн)}$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою (5.9):

$$ЗВ = \frac{В_{заг}}{\beta}, \quad (5.9)$$

де β – коефіцієнт, що характеризує етап (стадію) виконання даної роботи.

$$В_{заг} = \frac{25481,54}{0,9} = 28279,82 \text{ (грн)}$$

Таким чином, загальні витрати становлять 28279,82 грн.

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою (5.10):

$$\Delta П_i = \sum_1^n (\Delta П_я * N + П_я * \Delta N)i, \quad (5.10)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

N – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 30 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 30 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 200 користувачів, протягом другого року – на 150 користувачів, протягом третього року – 100 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 1000 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 400 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $ΔП_1$ протягом першого року складатиме:

$$ΔП\_1 = 30 \cdot 1000 + (400 + 30) \cdot 200 = 116000 \text{ грн.}$$

Протягом другого року:

$$ΔП\_2 = 30 \cdot 1000 + (400 + 30) \cdot (200 + 150) = 180500 \text{ грн}$$

Протягом третього року:

$$ΔП\_3 = 30 \cdot 1000 + (400 + 30) \cdot (200 + 150 + 100) = 223500 \text{ грн}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $E_{абс}$ вкладених інвестицій розраховується за формулою (5.11):

$$E_{абс} = (ПП - PV), \quad (5.11)$$

де ПП – вартість чистих прибутків,

PV – теперішня вартість інвестицій.

Схематична характеристика руху платежів, таких як інвестиції та додаткові прибутки, наведено на рисунку 5.1.

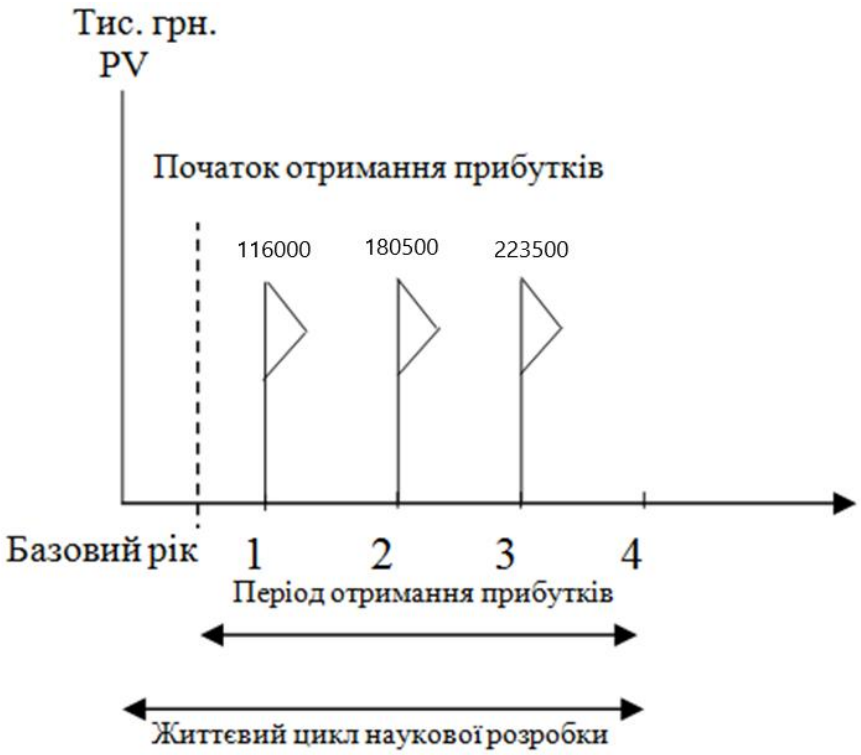


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів магістерської кваліфікаційної роботи

Вартість чистого прибутку розраховується за формулою (5.12):

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої магістерської кваліфікаційної роботи, грн;

t – період часу, протягом якого виявляються результати впровадженої магістерської кваліфікаційної роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні. Для України цей показник знаходиться на рівні 0,1.

Розрахуємо вартість чистого прибутку:

$$ПП = 28279{,}82/((1 + 0{,}1)\text{^}0\ ) + 116000/((1 + 0{,}1)\text{^}2\ ) + 180500/((1 + 0{,}1)\text{^}3\ ) + 223500/((1 + 0{,}1)\text{^}4\ ) = 412413{,}4 \text{ (грн).}$$

На основі вартості чистого прибутку розраховуємо $Е_{абс}$:

$$Е_{абс} = 412413{,}4 - 28279{,}82 = 384133{,}58 \text{ (грн).}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів магістерської кваліфікаційної роботи буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою (5.13):

$$Е_в = \sqrt[T]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (5.13)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій, грн;

Т – життєвий цикл наукової розробки, роки.

Таким чином,

$$E_в = \sqrt[3]{1 + \frac{384133,58}{28279,82}} - 1 = 1,44 \text{ або } 144\ \%$$

Показник відносної ефективності вкладених в наукову розробку інвестицій становить 137,94%.

Розрахована величина порівнюється з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає мінімальну дохідність, за якої можливе інвестування в розробку. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою (5.14):

$$\tau = d + f, \qquad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках. В 2019 році в Україні цей показник становить 0,2;

f – показник, що характеризує ризикованість вкладень. Приймемо його рівним 0,1.

Підставивши ці показники у формулу (5.14), отримаємо значення мінімальної ставки:

$$\tau = 0,2 + 0,1 = 0,3.$$

Оскільки показник відносної ефективності (137,94%) вкладених в наукову розробку інвестицій перевищує значення мінімальної ставки (30%), то розробка є такою, що має інвестиційну привабливість.

Термін окупності вкладених у реалізацію наукового проекту визначається за формулою (5.15):

$$T_{ок} = \frac{1}{E_в}, \quad (5.15)$$

Термін окупності становить 0,69 року.

5.5 Висновки

Під час виконання економічної частини магістерської кваліфікаційної роботи на основі розрахунків було доведено, що розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань та навичок є доцільною та має економічне обґрунтування.

На основі виконаних підрахунків одержано наступні результати:

- витрати на розробку та її впровадження складають 28279,82 грн., що не перевищує задане у технічному завданні значення;
- абсолютний ефект розробки складає 384133,58 грн. за три роки, що також задовольняє значення задане у технічному завданні;
- термін окупності системи, що розробляється, складає менше одного року, що вписується в задані в технічному завданні часові рамки та є показником доцільності розробки.

Таким чином, отримані вище результати доводять корисність, доцільність та необхідність даної розробки.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було розроблено метод та програмне забезпечення для керування поведінкою персонажами супротивника у стратегічних іграх.

1. Виконано аналіз предметної області, у результаті якого було зроблено висновок про актуальність розробки удосконаленого методу керування поведінкою персонажів супротивника у стратегічних іграх. Проведено дослідження існуючих методів керування поведінкою ворожих персонажів, а також існуючих стратегічних ігор, виявлено їх недоліки. Було доведено важливість якісного штучного інтелекту у жанрі стратегічних ігор, та розглянуто можливості його удосконалення

2. Проаналізовано модель штучного інтелекту Flocking AI, виявлено недоліки у її роботі. Запропоновано поняття матриці небезпеки – інформаційного шару бойової мапи, що зберігає інформацію про небезпечні події, що там відбуваються та який вплив на юнітів вони мають, та сприяє більш реалістичній поведінці персонажів супротивника. Проаналізовано загальну класифікацію методів та розроблено новий метод керування поведінкою ворожих військ з використанням матриці небезпеки, в результаті чого було досягнуто більш реалістичної поведінки ворогів у бойових ситуаціях.

3. Розроблено архітектуру програмної системи для впровадження розроблених методів, діаграми варіантів, послідовності і класів. Було розроблено програмні модулі, описано структуру компонентів, що забезпечують створення та налаштування мапи, ініціацію та процес роботи матриці небезпеки Для розробки використано середовище розробки Galaxy Editor, технологія XML та мова програмування скриптів Galaxy.

4. Виконано тестування розробленої системи, у результаті якого було зроблено висновок про відповідність програмного продукту критеріям якості. Розроблено інструкцію користувача, яка містить всю необхідну для користування програмною системою інформацію.

Отримані в магістерській кваліфікаційній роботі наукові та практичні результати можна використати для підвищення реалістичності поведінки персонажів противника у стратегічних комп’ютерних іграх.

Виконано розрахунок економічного ефекту від можливого впровадження розроблених методів та програмного забезпечення, в результаті якого підтверджено доцільність фінансування цієї наукової розробки.

Підсумовуючи усе вищесказане, можна зробити висновок, що задачі магістерської кваліфікаційної роботи було виконано у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тактична рольова гра – Вікіпедія. URL: https://goo.gl/98NDWZ (дата звернення: 13.10.2019)
2. Тактическая ролевая игра – Википедия. URL: https://goo.gl/M4JHE1 (дата звернення: 14.10.2019)
3. Short Term Decision Making with Fuzzy Logic And Long Term Decision Making with Neural Networks In Real-Time Strategy Games – Hevi. URL: http://hevi.info/tag/artificial-intelligence-in-real-time-strategy-games/ (дата звернення: 17.10.2019)
4. Романюк О. В., Любивий Б. О. Аналіз методів керування поведінкою ворогів у сучасних стрaгегічних іграх [Текст] / Б. О. Любивий, О. В. Романюк // XII Міжнародна науково-практична конференція "Інформаційні технології і автоматизація – 2019", Одеса, 17-18 жовтня 2019 : збірник доповідей. Одеса, 2019. – Ч. 2. – С. 60-62.
5. Романюк О. В., Любивий Б. О. Удосконалення методу керування поведінкою ворогів «flocking ai» в стратегічних іграх з використанням карти небезпек. Збірник матеріалів Міжнародної науково-практичної конференції «Електронні інформаційні ресурси в освіті і науці: створення, використання, доступ», Вінниця, 2019.
6. Tom Clancy’s Rainbow Six – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Tom_Clancy%E2%80%99s_Rainbow_Six_(%D0%B8%D0%B3%D1%80%D0%B0) (дата звернення: 21.10.2019)
7. Craig Reynolds Flocks, Herds, and Schools: A Distributed Behavioral Model. USA: SIGGRAPH , 1987 . 10 с.
8. Flocking - AI for Game Developers – Oreilly. URL:https://www.oreilly.com/library/view/ai-for-game/0596005555/ch04.html (дата звернення: 22.10.2019)
9. Mechanics of Starcraft 2 – TL.Net. URL: https://tl.net/forum/starcraft-2/132171-the-mechanics-of-sc2-part-1 (дата звернення: 23.10.2019)

10. Galaxy Map Editor – Starcraft Wiki. URL: http://starcraft.wikia.com/wiki/Galaxy_Map_Editor (дата звернення: 26.10.2019)
11. Структурные карты Константайна – Interface. URL: http://www.interface.ru/fset.asp?Url=/case/defs71.htm (дата звернення: 28.10.2019)
12. Терещенко П. В. Інтерфейси інформаційних систем.. Київ, 2012. – 650
13. Sells, Chris. Windows Forms Programming in C# (1st ed.), Addison-Wesley Professional, ст. 39.
14. Що таке XML – Taina. URL: https://www.taina.com.ua/shho-take-xml (дата звернення: 30.10.2019)
15. Степанченко И.В. Методы тестирования программного обеспечения: Учеб. Пособие. Волгоград: ВолгГТУ, 2006. 74 с.
16. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. Питер: СПб, 2004. 320 с
17. Поняття про тестування програмного забезпечення – Helpiks. URL: http://helpiks.org/5-101392.html (дата звернення: 11.11.2019) (дата звернення: 01.11.2019)
18. Степанченко И. В. Методы тестирования программного обеспечения. Волгоград: ВолгГТУ, 2006. 74 с
19. Тест-кейс.URL: https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F (дата звернення: 15.11.19).
20. Козловський В.О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТКИ

ДОДАТОК А Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри ПЗ, д.т.н., проф.

Романюк О. Н.

«___» _____ 2019 р.

Технічне завдання

на магістерську кваліфікаційну роботу

на тему: «Підвищення рівня інтелектуальності поведінки персонажів у стратегічних іграх з урахуванням особливостей місцевості»

за спеціальністю: 121 – «Інженерія програмного забезпечення»

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доц. Романюк О. В.

«___» _____ 2019 р.

Виконав:

_____ ст. гр. 1ПІ-18м Любивий Б. О.

«___» _____ 2019 р.

Вінниця – 2019 р.

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Підвищення рівня інтелектуальності поведінки персонажів у стратегічних іграх з урахуванням особливостей місцевості».

Галузь застосування – комп’ютерні стратегічні ігри.

2. Підстава для розробки

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №____ ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки

Метою роботи є підвищення рівня інтелектуальності поведінки персонажів у сучасних стратегічних іграх, за рахунок розробки нового методу керування поведінкою персонажів, що забезпечував би можливість маневрування ворожого війська місцевістю.

Призначення роботи – розробка нових методів керування, що характеризуватимуться більшим рівнем реалістичності поведінки.

4. Вихідні дані для проведення науково-дослідної роботи

Перелік основних літературних джерел, на основі яких буде виконуватись магістерська кваліфікаційна робота:

1. Flocking - AI for Game Developers – Oreilly. URL:https://www.oreilly.com/library/view/ai-for-game/0596005555/ch04.html (дата звернення: 22.10.2019
2. Galaxy Map Editor – Starcraft Wiki. URL: http://starcraft.wikia.com/wiki/Galaxy_Map_Editor (дата звернення: 26.10.2019)

5. Технічні вимоги

Вихідні дані до роботи: Жанр гри – тактично-стратегічна в реальному часі; базовий алгоритм керування поведінкою персонажів – flocking AI; стандартна кількість одиниць в загоні гравця – 3, максимальна кількість ворожих одиниць - 150.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт

- пояснювальна записка до магістерської кваліфікаційної роботи;
- технічне завдання;
- лістинг програмної системи.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання та існуючих аналогів, постановка задач дослідження	04.09.2019 – 30.09.2019	Вик.
2	Розробка методу керування поведінкою персонажів з використанням «Матриці Небезпеки»	30.09.2019 – 27.10.2019	Вик.
3	Розробка структур і алгоритмів програмного продукту	28.10.2019 – 10.11.2019	Вик.
4	Розробка програмної реалізації модифікованої моделі штучного інтелекту "Flocking AI"	11.11.2019 – 15.11.2019	Вик.
6	Тестування розробленого програмного продукту	16.11.2019 – 21.11.2019	Вик.

1	2	3	4
7	Економічна частина	17.11.2019 – 20.11.2019	Вик.
8	Оформлення матеріалів до захисту МКР	04.09.2019 – 21.11.2019	Вик.

10. Порядок контролю на прийняття

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б Загальний алгоритм роботи матриці небезпеки

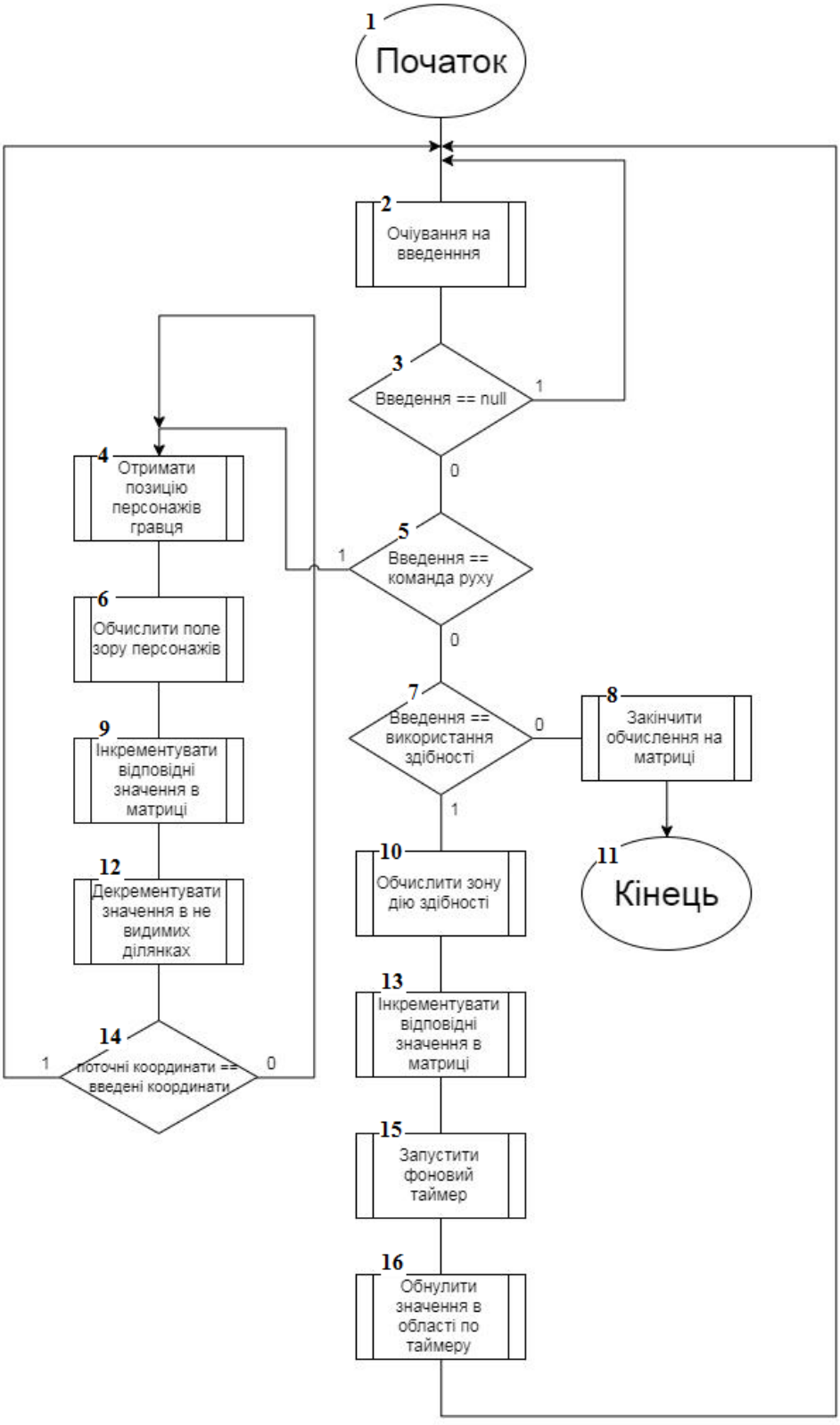


Рисунок В.1 – Загальний алгоритм роботи матриці небезпеки

ДОДАТОК В Графічні матеріали

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д.т.н., професор _____ О. Н. Романюк

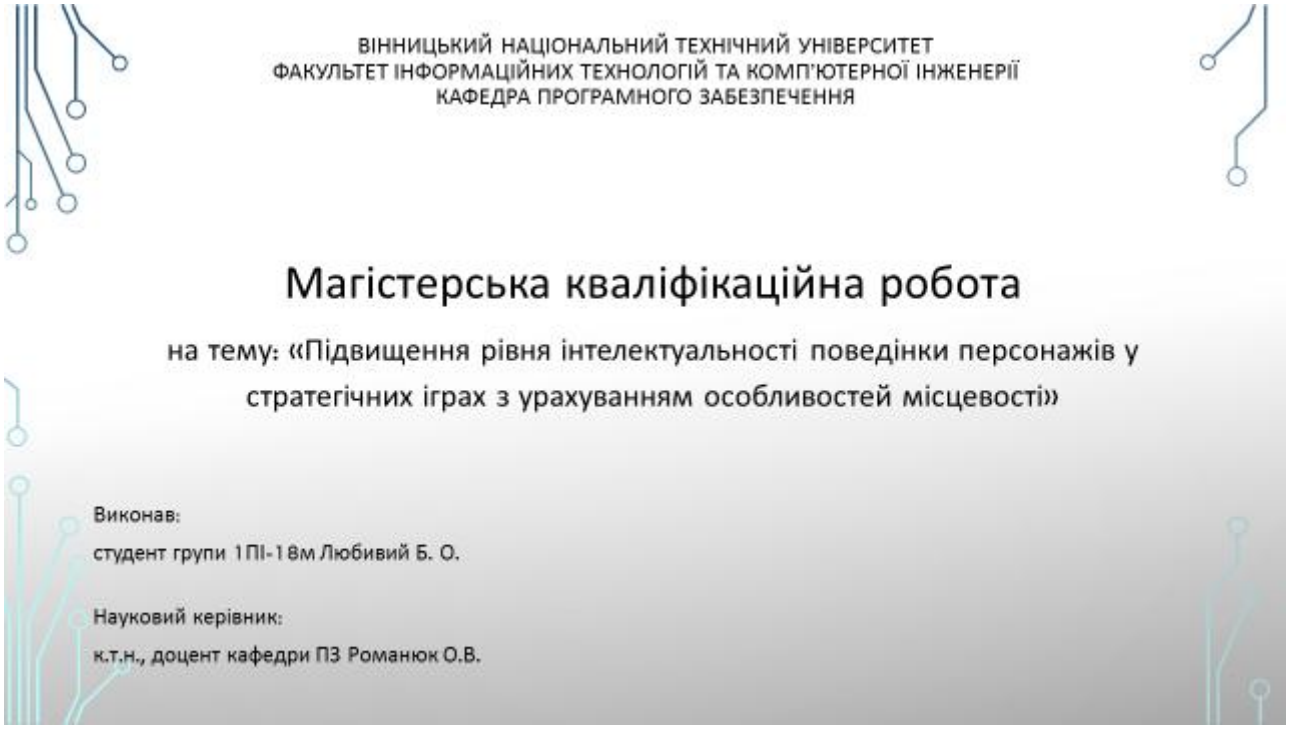
Науковий керівник, к.т.н., доцент _____ О. В. Романюк

Рецензент, к.т.н., доцент _____ Я. В. Іванчук

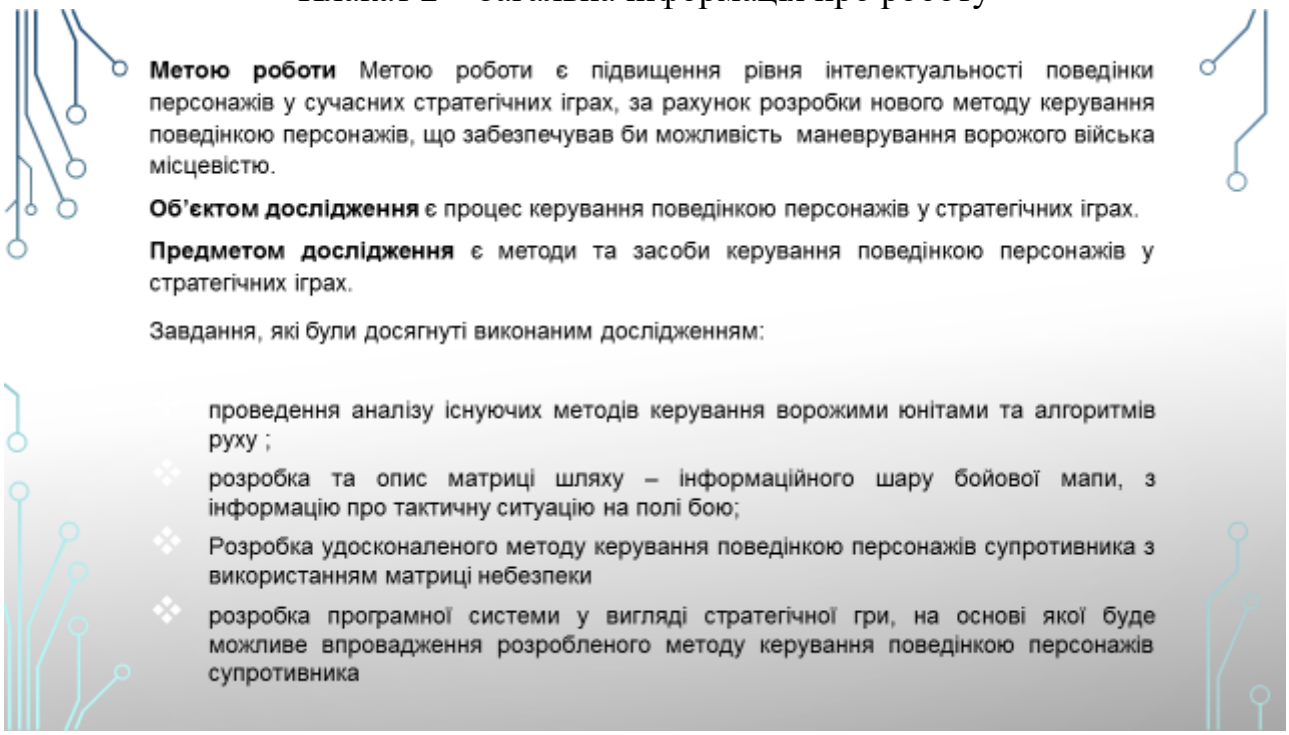
Нормоконтроль, к.т.н., доцент _____ О. В. Романюк

Виконавець, студент групи 1ПІ-18м _____ Б. О. Любивий

Плакат 1 – Тема роботи, автор та керівник



Плакат 2 – Загальна інформація про роботу



Плакат 3 – Наукова новизна отриманих результатів

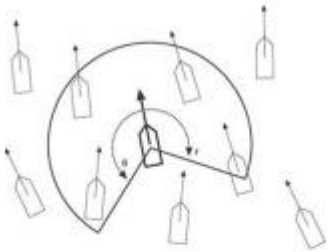
Наукова новизна отриманих результатів

Вперше запропоновано поняття матриці небезпеки, яка дозволила підвищити рівень інтелектуальності та реалістичності поведінки ворожих персонажів у стратегічних іграх.

Удосконалено метод керування поведінкою персонажів, який, на відміну від відомих, використовує матрицю небезпек, що дозволило підвищити рівень інтелектуальності поведінки персонажів у стратегічних іграх.

Плакат 4 – Flocking AI

Flocking AI



Крейг Рейнольдс запропонував термін «flocking AI» для опису поведінки ворожого війська як зграї, а не кожного окремого бійця.

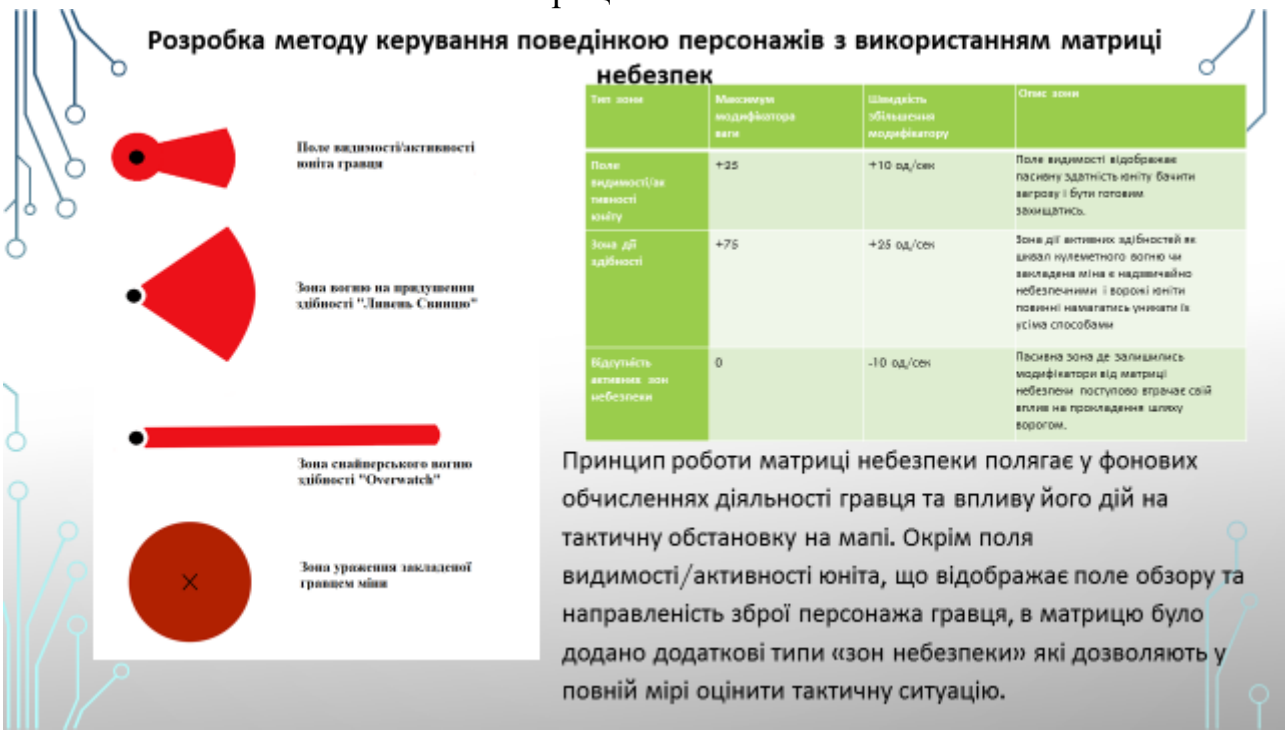
Така реалізація не передбачає наявності лідера у зграї, натомість уся зграя сприймається як єдиний організм, що має власний інтелект. Подібна поведінка реалізовується на основі трьох правил:

- Згуртованість: кожна особина зграї повинна орієнтуватися на середнє положення своїх сусідів.
- Вирівнювання: кожна особина зграї повинна спрямовуватись відповідно до середнього напрямку руху сусідів.
- Розмежування: потрібно уникати скупчення сусідів, щоб не штовхати один одного.

З цих трьох правил зрозуміло, що кожен підрозділ повинен знати про своє місцеве оточення - він повинен знати, де розташовані його сусіди, куди вони очолюються та наскільки вони близькі до нього.

Плакат 5 – Поняття матриці небезпеки

Плакат 6 – Розробка методу керування поведінкою персонажів з використанням матриці небезпеки



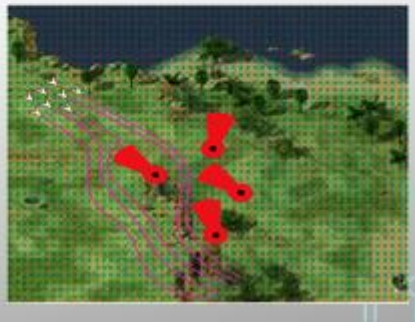
Тип зони	Максимум модифікатора ваги	Швидкість збільшення модифікатору	Опис зони
Поле видимості/активності юніту	+25	+10 од/сек	Поле видимості відображає пасивну здатність юніту бачити загрозу і бути готовим захищатись.
Зона дії здібності	+75	+25 од/сек	Зона дії активних здібностей як шквал кулеметного вогню чи закладена міна є надзвичайно небезпечними і ворожі юніти повинні намагатись уникати їх усіма способами
Відсутність активних зон небезпеки	0	-10 од/сек	Пасивна зона де залишились модифікатори від матриці небезпеки поступово втрачає свій вплив на прокладання шляху ворогом.

Плакат 7 – Сценарії поведінки ворожих персонажів

Сценарії поведінки ворожих персонажів

Базові сценарії: Перший полягає в тому що ворог знаходиться по звичному шляху переміщення і тому юніт буде намагатись обійти його, без особливих втрат часу. Другий – де ворог і є кінцевим пунктом руху. В такому випадку юніт буде намагатись обійти його зліва чи справа і атакувати збоку

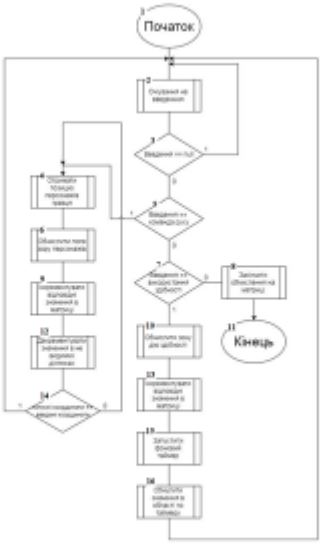
Складніший сценарій - з великою кількістю і персонажів гравця і ворожих одиниць. По схематичним лініям видно що зграя буде вимушена розділитись на дві, щоб уникнути гравця. Верхня група буде намагатись піти на прорив і маневрувати між юнітами гравця, тоді коли інша буде намагатись обійти їх стороною. В кінці маршруту, якщо перша група вціліє, вони знову об'єднаються разом у одну зграю і продовжать шлях до цілі.



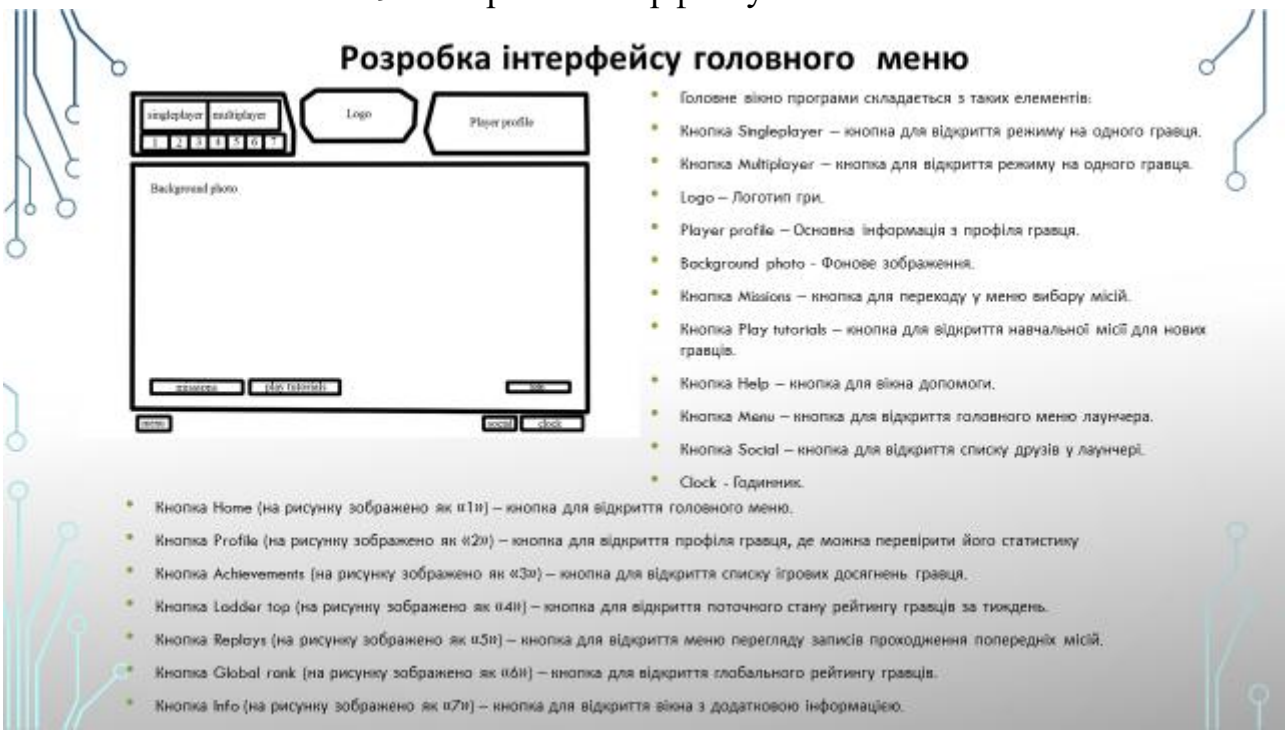
Плакат 8 – Алгоритм роботи матриці небезпеки

Алгоритм роботи матриці небезпеки

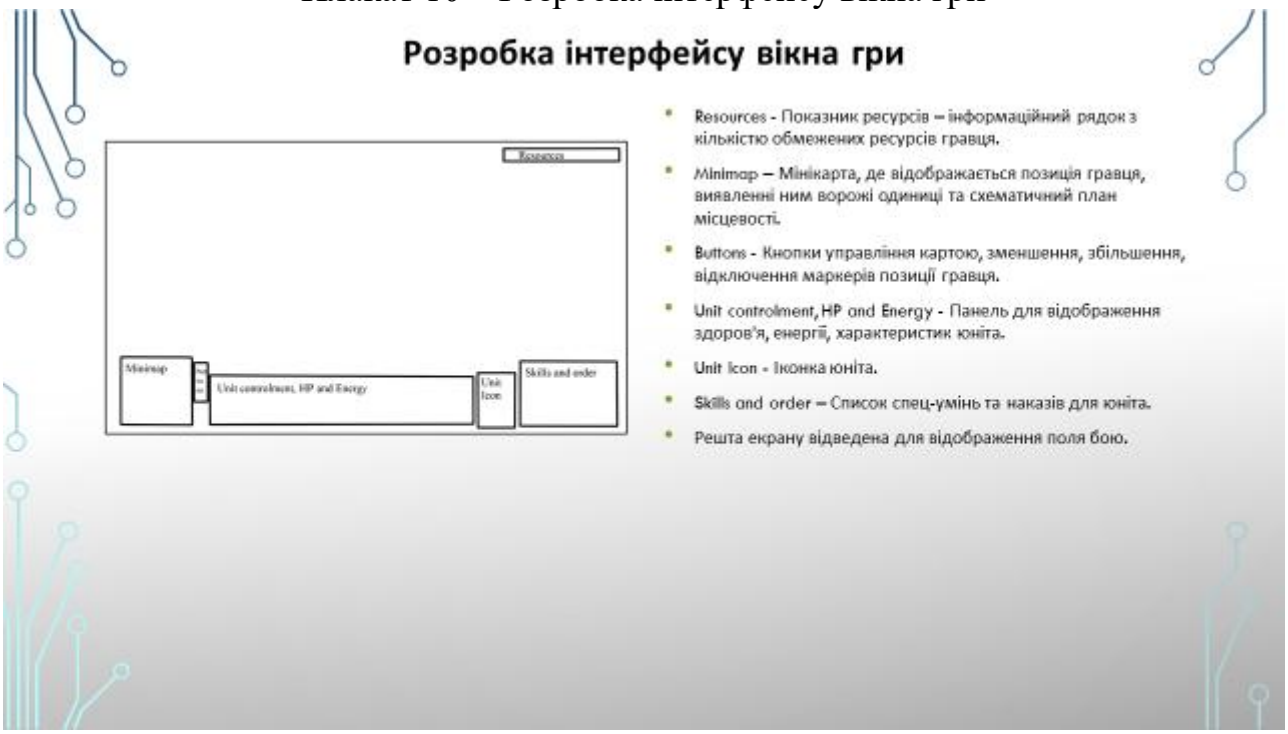
- 1 – Початок роботи модуля.
- 2 – Блок очікування введення команди від гравця. Перевірка відбувається кожний внутрігровий тік – 0.2 с.
- 3 – Якщо Введення = Null. Значит нової команди не наступило, модуль продовжує чекати.
- 5 – Якщо Введення = команда руху з легітимними координатами відбувається запуск гілки алгоритму для обрахунку полей видимості юнітів.
- 4 – Отримання позицій усіх юнітів гравця протягом даного тіку гри
- 6 – Обчислення зони видимості юнітів в даний момент. Збереження цих даних.
- 8 – Підвищення модифікатора ваги активних полей зі збереженої інформацією з відповідними до типу швидкістю збільшення і максимальним значенням.
- 12 – Зменшення модифікатора ваги пасивних зон згідно зі сталою швидкістю.
- 14 – Якщо позиція юнітів відповідає заданим в введеній команді руху координатам – повернутись до очікування введення. Якщо ні – повернення до блоку 4.
- 7 – Якщо Введення = команда використання здібностей відбувається запуск гілки алгоритму для обрахунки зон дії здібностей.
- 10 – Обчислення зони дії здібності. Збереження цих даних.
- 13 – Підвищення модифікатора ваги активних полей зі збереженої інформацією з відповідними до типу швидкістю збільшення і максимальним значенням.
- 15 – Запуск фонового таймеру по характеристиці тривалості здібності.
- 16 – При закінченні часу тривалості здібності, обнулення модифікатора ваги в зоні її дії.
- 8 – Якщо введення було одним із визначених типів команди, або командою «Вихід» алгоритм закінчує роботу.
- 11 – Кінець роботи модуля.



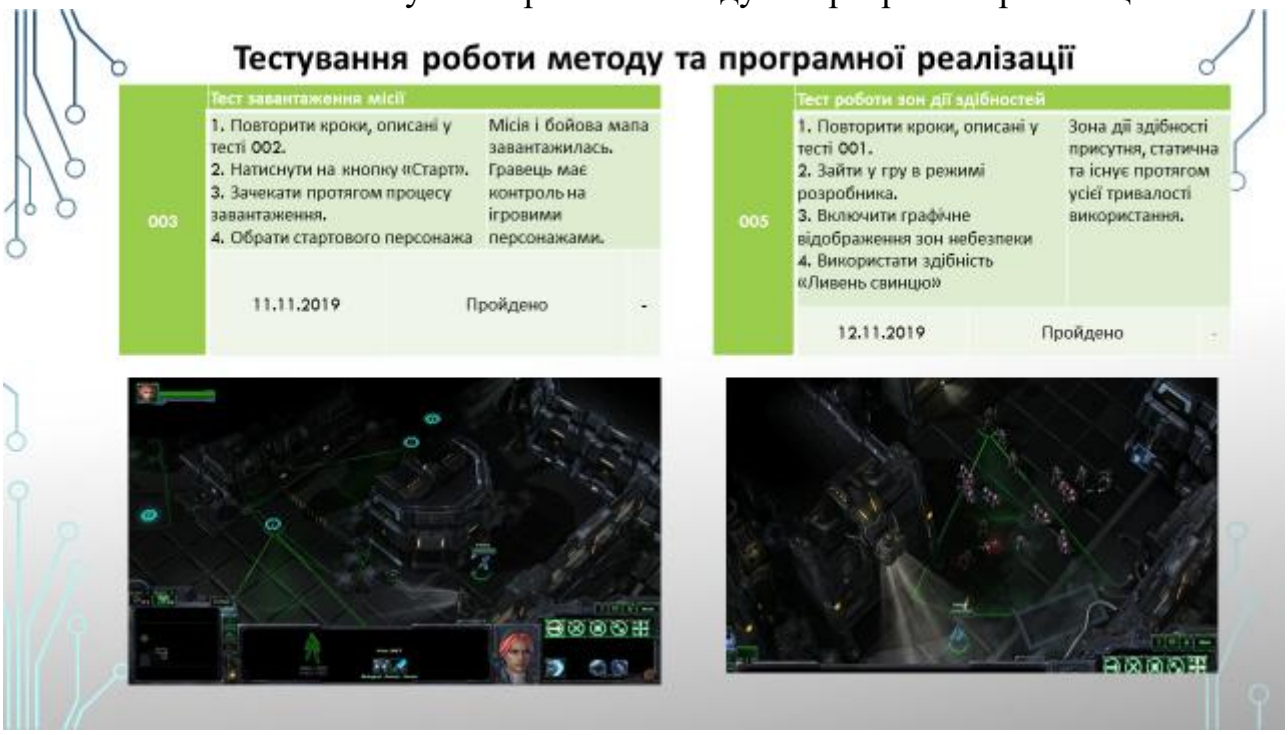
Плакат 9 – Розробка інтерфейсу головного меню



Плакат 10 – Розробка інтерфейсу вікна гри

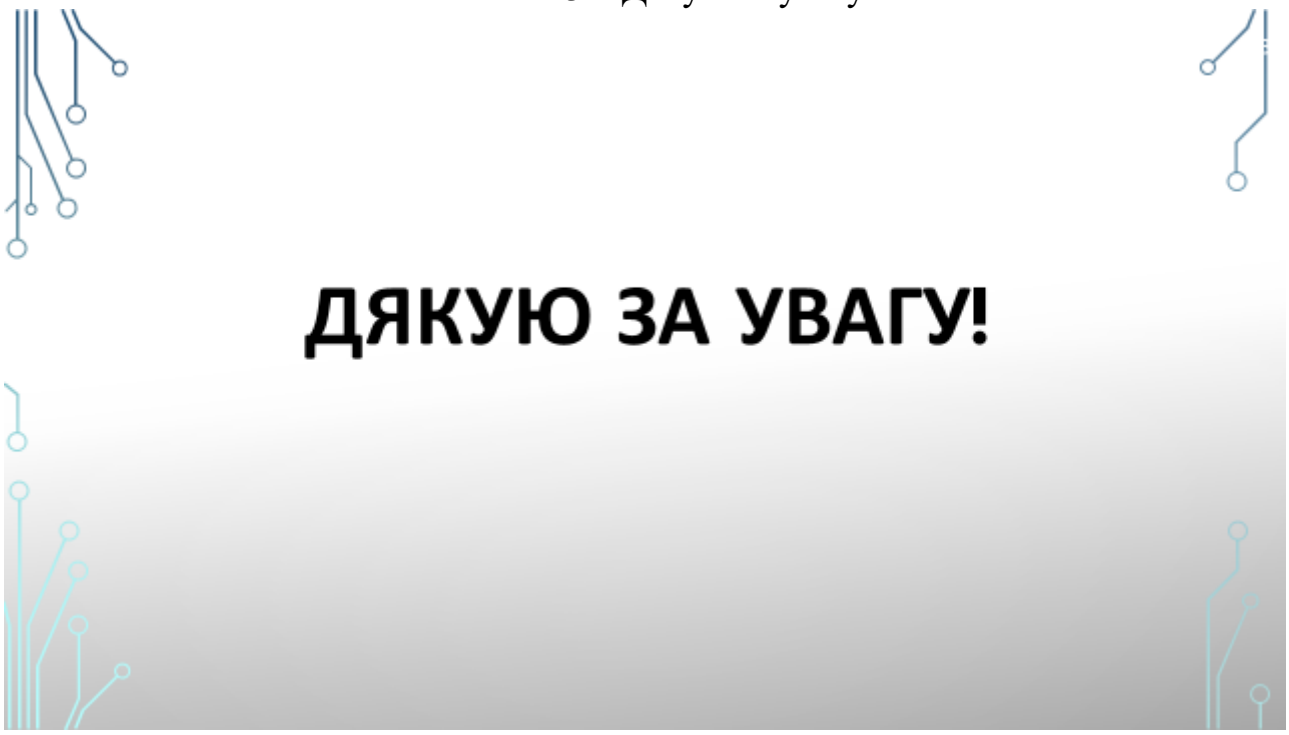


Плакат 11 – Тестування роботи методу та програмної реалізації



Плакат 12 – Економічне обґрунтування

Плакат 13 – Дякую за увагу



ДОДАТОК Г Лістинг коду

Лістинг «DPSLMapChanger.cpp»

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include "pngwriter.h"

#include "outstreams.hpp"
#include "coordinates.hpp"
#include "DPSLMap.hpp"


// Note that this set is static, there
// is only one for the execution
set<string> DPSLMap::mapFilenamesUsed;



DPSLMap::DPSLMap( string toolPathIn,
                  string outputPathIn,
                  string argPathIn,
                  string archiveNameIn,
                  string archiveWithExtIn )
{
  toolPath       = toolPathIn;
  outputPath     = outputPathIn;
  argPath        = argPathIn;
  archiveName    = archiveNameIn;
  archiveWithExt = archiveWithExtIn;

  mapHeight       = NULL;
  mapCliffChanges = NULL;
  mapPathing      = NULL;
  mapOpenness     = NULL;
  mapOpennessPrev = NULL;
  mapPathNodes    = NULL;

  totalMinerals     = 0.0f;
  totalVespeneGas   = 0.0f;
  totalHYMinerals   = 0.0f;
  totalHYVespeneGas = 0.0f;
```

```
}


DPSLMap::~DPSLMap()
{
 delete img;

 for( list<Resource*>::const_iterator itr = resources.begin();
      itr != resources.end();
      ++itr )
 {
  delete *itr;
 }

 for( list<Base*>::const_iterator itr = bases.begin();
      itr != bases.end();
      ++itr )
 {
  Base* b = *itr;

  for( list<StartLoc*>::const_iterator slItr = startLocs.begin();
       slItr != startLocs.end();
       ++slItr )
  {
   StartLoc* sl = *slItr;
   delete b->sl1vsl2_influence[sl];
  }

  delete b;
 }

 for( list<StartLoc*>::const_iterator itr = startLocs.begin();
      itr != startLocs.end();
      ++itr )
 {
  delete *itr;
 }

 for( list<Watchtower*>::const_iterator itr = watchtowers.begin();
      itr != watchtowers.end();
      ++itr )
 {
  delete *itr;
 }
```

```
for( list<Destruct*>::const_iterator itr = destructs.begin();
    itr != destructs.end();
    ++itr )
{
 delete *itr;
}

for( list<LoSB*>::const_iterator itr = losbs.begin();
    itr != losbs.end();
    ++itr )
{
 delete *itr;
}

if( mapHeight )
{
 delete mapHeight;
}

if( mapCliffChanges )
{
 delete mapCliffChanges;
}

if( mapPathing )
{
 delete mapPathing;
}

if( mapOpenness )
{
 delete mapOpenness;
}

if( mapOpennessPrev )
{
 delete mapOpennessPrev;
}

if( mapPathNodes )
{
 delete mapPathNodes;
}
```

```
  for( int t = 0; t < NUM_PATH_TYPES; ++t )
  {
    for( int i = 0; i < nodes[t].size(); ++i )
    {
      delete nodes[t][i];
    }

    for( map< int, vector<float>* >::iterator itr = d[t].begin();
         itr != d[t].end();
         ++itr )
    {
      delete (*itr).second;
    }

    for( map< int, vector<Node*>* >::iterator itr = pi[t].begin();
         itr != pi[t].end();
         ++itr )
    {
      delete (*itr).second;
    }
  }
}
```

Лістинг «DPSLMapAnalyzer.cpp»

```
#include <string>
using namespace std;

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>

#include "debug.hpp"
#include "utility.hpp"
#include "config.hpp"
```

```
#include "outstreams.hpp"
#include "DPSLMap.hpp"
#include "DPSLMapChanger.hpp"

static bool pauseTerminal         = true;
static bool dirRecurse            = false;
static bool renderTerrain         = false;
static bool renderPathing         = false;
static bool renderBases           = false;
static bool renderOpenness        = false;
static bool renderShortest        = false;
static bool renderInfluence       = false;
static bool renderInfluenceHeatMap = false;
static bool renderSummary         = false;
static bool writeCSVpermap        = false;
static bool writeCSVaggr          = false;


void statFail( string* fileFullPath );

void tryFileOrDir( string* fileFullPath, bool first, bool recurse );
void tryFile     ( string* fileFullPath );


void AtExit()
{
  // flush output
  fflush( stdout );
  fflush( stderr );

  printOutstreamStatus();

  // make sure to hold terminal open so
  // user can read output
  if( pauseTerminal )
  {
    system( "pause" );
  }
}


static string          toolPath;
static DPSLMapChanger MapChanger;
```

```
int main( int argc, char** argv )
{
  atexit( &AtExit );


  if( DPSLMap::pngwriterTest() != 0 )
  {
    printError( "Cannot load resources for PNGwriter: you may need to run this
program as Administrator.\n" );
    exit( -1 );
  }

  string toolFullName( argv[0] );

  string::size_type locBSlash = toolFullName.rfind( "\\" );
  toolPath.assign( toolFullName, 0, locBSlash + 1 );
  formatPath( &toolPath );

  printMessage( "DPSL Map Analyzer v%s, Algorithms v%s\n  Tool path: %s\n
Message %s for support.\n\n",
                QUOTEMACRO( VEXE ),
                QUOTEMACRO( VALG ),
                toolPath.data(),
                adminEmail
              );

  initConfigReading();
  readConfigFiles( false, NULL, &configInternal );

  readConfigFiles( true, &toolPath, &configUserGlobal );


  writeCSVaggr = getGlobalOutputOption( "writeCSVaggr" );


  if( configUserGlobal.toAnalyze.empty() &&
      configUserGlobal.toAnalyzeRecurse.empty() )
  {
    printMessage( "No file arguments, searching current directory for maps...\n\n" );
    string currentDir( "." );
    tryFileOrDir( &currentDir, true, true );
  }
```

```
  for( list<string>::iterator argItr = configUserGlobal.toAnalyze.begin();
      argItr != configUserGlobal.toAnalyze.end();
      ++argItr )
  {
   tryFileOrDir( &(*argItr), true, false );
  }

  for( list<string>::iterator argItr = configUserGlobal.toAnalyzeRecurse.begin();
      argItr != configUserGlobal.toAnalyzeRecurse.end();
      ++argItr )
  {
   tryFileOrDir( &(*argItr), true, true );
  }


  if( writeCSVaggr )
  {
   printMessage( "Calculating aggregate statistics...\n" );
   MapChanger.buildOutputColumns();

   if( configUserGlobal.outputPath.empty() )
   {
    MapChanger.writeToCSV( &toolPath );
   } else {
    MapChanger.writeToCSV( &(configUserGlobal.outputPath) );
   }
  }

  return 0;
}



void statFail( string* fileFullPath )
{
  const char* filename = fileFullPath->data();

  const char msgToAnalyze[] = "\nHave you edited to-analyze.txt with paths to maps
you want to analyze?\n\n";

  switch( errno )
  {
   case EACCES:
    printWarning( "Permission denied for %s, skipping...\n",
```

```
                filename );
      break;

    case EBADF:
    case EFAULT:
      printWarning( "%s is an invalid file, skipping...\n",
                filename );
      break;

    case ENAMETOOLONG:
      printWarning( "Path and filename %s is too long, skipping...\n",
                filename );
      break;

    case ENOENT:
    case ENOTDIR:
      printWarning( "A component of the path %s does not exist or is not a directory,
skipping...\n%s",
                filename,
                msgToAnalyze );
      break;

    case ENOMEM:
      printError( "Not enough system memory to continue.\n" );
      exit( -1 );
      break;

    default:
      printWarning( "Unknown problem trying to access %s, skipping...\n",
                filename );
  }
}


// find out if the non-option argument is a file or
// a directory, and for directories decide if we should
// recursively explore them if the flag is set
void tryFileOrDir( string* fileFullPathIn, bool first, bool recurse )
{

  formatPath( &fileFullPath );
  struct stat fileStatus;
  if( stat( fileFullPath.data(), &fileStatus ) < 0 )
  {
```

```
  statFail( &fileFullPath );
  return;
 }
 if( S_ISREG( fileStatus.st_mode ) )
 {
  tryFile( &fileFullPath );


 } else if( S_ISDIR( fileStatus.st_mode ) ) {

  if( !first && !recurse )
  {
   return;
  }

  DIR* dir = opendir( fileFullPath.data() );
  if( dir != NULL )
  {
   struct dirent* de;
   while( de = readdir( dir ) )
   {
    // ignore the current and up-one directories, . and ..
    if( strcmp( de->d_name, "."  ) == 0 ||
        strcmp( de->d_name, ".." ) == 0 )
    {
     continue;
    }

    // make a path out of this directory name
    string deeperFile( fileFullPath );
    deeperFile.append( "\\" );
    deeperFile.append( de->d_name );

    // only continue to recurse beyond the first level
    // if the user passed recursive option
    tryFileOrDir( &deeperFile, false, recurse );
   }
   closedir( dir );

  } else {
   printWarning( "Could not open directory %s.\n", fileFullPath.data() );
  }

 } else {
```

```
    printWarning( "%s is not a regular file or directory, skipping...", fileFullPath.data()
);
  }
}



// found a regular file, see if it has a map
// extension and then try to open the archive
void tryFile( string* fileFullPath )
{
  string ext1( ".s2ma" );
  string ext2( ".DPSLMap" );

  string::size_type loc1 = fileFullPath->rfind( ext1 );
  string::size_type loc2 = fileFullPath->rfind( ext2 );

  // if either search for an extension doesn't come
  // up with the location where the extension should be...
  if( loc1 != fileFullPath->size() - ext1.size() &&
      loc2 != fileFullPath->size() - ext2.size() )
  {
    // not a Starcraft 2 Map Archive...
    return;
  }

  // find last dot, last back slash
  string::size_type locDot    = fileFullPath->rfind( "." );
  string::size_type locBSlash = fileFullPath->rfind( "\\" );

  string path( *fileFullPath, 0,            locBSlash + 1               );
  string file( *fileFullPath, locBSlash + 1, locDot - locBSlash - 1       );
  string ext ( *fileFullPath, locDot,       fileFullPath->size() - locDot );

  string localOutputPath( path );

  // consider switching to the globally specified
  // output path, if it exists AND is valid
  if( !configUserGlobal.outputPath.empty() )
  {
    struct stat dirStatus;
    if( stat( configUserGlobal.outputPath.data(), &dirStatus ) >= 0 )
    {
      if( S_ISDIR( dirStatus.st_mode ) )
```

```
    {
     localOutputPath.assign( configUserGlobal.outputPath );
    }
   }
  }


  string fileWithExt( file );
  fileWithExt.append( ext );

  DPSLMap* DPSLmap = new DPSLMap( toolPath,
                        localOutputPath,
                        path,
                        file,
                        fileWithExt );

  printMessage( "\nOpening %s...\n", fileFullPath->data() );

  readConfigFiles( false, &path, &(DPSLmap->configUserLocal) );

  // consider switching to the locally specified
  // output path, if it exists AND is valid
  if( !DPSLmap->configUserLocal.outputPath.empty() )
  {
   struct stat dirStatus;
   if( stat( DPSLmap->configUserLocal.outputPath.data(), &dirStatus ) >= 0 )
   {
    if( S_ISDIR( dirStatus.st_mode ) )
    {
     DPSLmap->outputPath = DPSLmap->configUserLocal.outputPath;
    }
   }
  }

  DangerMatrixInitiateTerrain         = DPSLmap->getOutputOption(
"DangerMatrixInitiateTerrain"         );
  DangerMatrixInitiatePathing         = DPSLmap->getOutputOption(
"DangerMatrixInitiatePathing"         );
  DangerMatrixInitiateBases           = DPSLmap->getOutputOption(
"DangerMatrixInitiateBases"           );
  DangerMatrixInitiateOpenness        = DPSLmap->getOutputOption(
"DangerMatrixInitiateOpenness"        );
  DangerMatrixInitiateShortest        = DPSLmap->getOutputOption(
"DangerMatrixInitiateShortest"        );
```

```
  DangerMatrixInitiateInfluence        = DPSLmap->getOutputOption(
"DangerMatrixInitiateInfluence"        );
  DangerMatrixInitiateInfluenceHeatMap = DPSLmap->getOutputOption(
"DangerMatrixInitiateInfluenceHeatMap" );
  DangerMatrixInitiateSummary          = DPSLmap->getOutputOption(
"DangerMatrixInitiateSummary"          );
  writeCSVpermap         = DPSLmap->getOutputOption( "writeCSVpermap"         );

  if( DPSLmap->readMap() < 0 )
  {
    printWarning( "Could not read required map files for %s, skipping\n\n", file.data()
);
    return;
  }

  printMessage( "Prepping analysis,\n" );

  DPSLmap->countPathableCells();

  printMessage( "." );

  DPSLmap->prepShortestPaths();

  printMessage( "." );

  DPSLmap->identifyBases();

  printMessage( "." );

  DPSLmap->computeOpenness();

  printMessage( "." );

  DPSLmap->locateChokes();

  printMessage( "." );

  DPSLmap->analyzeBases();

  printMessage( "\nAnalyzing and generating output,\n" );
  printMessage( "  Map-specific output in [%s]\n", DPSLmap->outputPath.data() );

  if( renderTerrain )
  {
```

```
  DPSLmap->renderImageTerrainOnly();
}

printMessage( "." );

if( renderPathing )
{
  DPSLmap->renderImagesPathing();
}

printMessage( "." );

if( renderBases )
{
  DPSLmap->renderImageBases();
}

printMessage( "." );

if( renderOpenness )
{
  DPSLmap->renderImageOpenness();
}

printMessage( "." );

if( renderShortest )
{
  DPSLmap->renderImagesShortestPaths();
}

printMessage( "." );

if( renderInfluence )
{
  DPSLmap->renderImagesInfluence();
}

printMessage( "." );

if( renderInfluenceHeatMap )
{
  DPSLmap->renderImagesInfluenceHeatMap();
}
```

```
  printMessage( "." );

  if( renderSummary )
  {
    DPSLmap->renderImageSummary();
  }

  printMessage( "." );

  if( writeCSVpermap )
  {
    DPSLmap->writeToCSV();
  }

  printMessage( "." );

  if( writeCSVaggr )
  {
    MapChanger.aggregate( DPSLmap );
  }

  printMessage( "\n\n" );


  // normally the guts of this function
  // are commented out, uncomment for testing
  DPSLmap->renderTestImages();


  delete DPSLmap;
}
```

Лістинг «DPSLDangerMatrix.cpp»

```
#include <stdlib.h>
#include <stdio.h>

#include " DPSLDangerMatrix.hpp"
```

```
const float point::MatrixNaN  = -987.654321f;
const int   point::MatrixtNaN = -12345;
const int   point::MatrixcNaN = -23456;
const int   point::ptNaN = -34567;
const int   point::pcNaN = -45678;


void point::set( point* p )
{
  Matrixx  = p->Matrixx;  Matrixy  = p->Matrixy;
  Matrixtx = p->Matrixtx; Matrixty = p->Matrixty;
  Matrixcx = p->Matrixcx; Matrixcy = p->Matrixcy;
  ptx = p->ptx; pty = p->pty;
  pcx = p->pcx; pcy = p->pcy;
  ix  = p->ix;  iy  = p->iy;
}

void point::MatrixSet( float MatrixxIn, float MatrixyIn )
{
  Matrixx  = MatrixxIn;          Matrixy  = MatrixyIn;
  Matrixtx = Matrix2Matrixt  ( MatrixxIn ); Matrixty = Matrix2Matrixt  ( MatrixyIn );
  Matrixcx = Matrix2Matrixc  ( MatrixxIn ); Matrixcy = Matrix2Matrixc  ( MatrixyIn
);
  ptx = Matrixx2ptx( MatrixxIn ); pty = Matrixy2pty( MatrixyIn );
  pcx = Matrixx2pcx( MatrixxIn ); pcy = Matrixy2pcy( MatrixyIn );
  ix  = Matrixx2ix ( MatrixxIn ); iy  = Matrixy2iy ( MatrixyIn );
}

void point::MatrixtSet( int MatrixtxIn, int MatrixtyIn )
{
  Matrixx  = Matrixt2Matrix   ( MatrixtxIn ); Matrixy  = Matrixt2Matrix   (
MatrixtyIn );
  Matrixtx = MatrixtxIn;          Matrixty = MatrixtyIn;
  Matrixcx = MatrixcNaN;          Matrixcy = MatrixcNaN;
  ptx = Matrixtx2ptx( MatrixtxIn ); pty = Matrixty2pty( MatrixtyIn );
  pcx = pcNaN;          pcy = pcNaN;
  ix  = Matrixtx2ix ( MatrixtxIn ); iy  = Matrixty2iy ( MatrixtyIn );
}

void point::MatrixcSet( int MatrixcxIn, int MatrixcyIn )
{
  Matrixx  = Matrixc2Matrix   ( MatrixcxIn ); Matrixy  = Matrixc2Matrix   (
MatrixcyIn );
  Matrixtx = MatrixtNaN;          Matrixty = MatrixtNaN;
```

```
  Matrixcx = MatrixcxIn;          Matrixcy = MatrixcyIn;
  ptx = ptNaN;           pty = ptNaN;
  pcx = Matrixcx2pcx( MatrixcxIn ); pcy = Matrixcy2pcy( MatrixcyIn );
  ix  = Matrixcx2ix ( MatrixcxIn ); iy  = Matrixcy2iy ( MatrixcyIn );
}

void point::ptSet( int ptxIn, int ptyIn )
{
  Matrixx  = ptx2Matrixx ( ptxIn ); Matrixy  = pty2Matrixy ( ptyIn );
  Matrixtx = ptx2Matrixtx( ptxIn ); Matrixty = pty2Matrixty( ptyIn );
  Matrixcx = MatrixcNaN;          Matrixcy = MatrixcNaN;
  ptx = ptxIn;           pty = ptyIn;
  pcx = pcNaN;           pcy = pcNaN;
  ix  = ptx2ix ( ptxIn ); iy  = pty2iy ( ptyIn );
}

void point::pcSet( int pcxIn, int pcyIn )
{
  Matrixx  = pcx2Matrixx ( pcxIn ); Matrixy  = pty2Matrixy ( pcyIn );
  Matrixtx = MatrixtNaN;          Matrixty = MatrixtNaN;
  Matrixcx = pcx2Matrixcx( pcxIn ); Matrixcy = pcy2Matrixcy( pcyIn );
  ptx = ptNaN;           pty = ptNaN;
  pcx = pcxIn;           pcy = pcyIn;
  ix  = pcx2ix ( pcxIn ); iy  = pcy2iy ( pcyIn );
}

void point::iSet( int ixIn, int iyIn )
{
  Matrixx  = MatrixNaN;  Matrixy  = MatrixNaN;
  Matrixtx = MatrixtNaN; Matrixty = MatrixtNaN;
  Matrixcx = MatrixcNaN; Matrixcy = MatrixcNaN;
  ptx = ptNaN; pty = ptNaN;
  pcx = pcNaN; pcy = pcNaN;
  ix  = ixIn;  iy  = iyIn;
}


// translation constants
int point::MatrixcLeft;
int point::MatrixcBottoMatrix;
int point::iDiMatrixT;
int point::iDiMatrixC;

float point::MatrixcLeftf;
```

```
float point::MatrixcBottoMatrixf;
float point::iDiMatrixTf;

void point::setFraMatrixeTranslationConstants( int MatrixcLeftIn,
                                               int MatrixcBottoMatrixIn,
                                               int iDiMatrixTIn,
                                               int iDiMatrixCIn )
{
  MatrixcLeft   = MatrixcLeftIn;
  MatrixcBottoMatrix = MatrixcBottoMatrixIn;
  iDiMatrixT    = iDiMatrixTIn;
  iDiMatrixC    = iDiMatrixCIn;

  MatrixcLeftf   = (float)MatrixcLeft;
  MatrixcBottoMatrixf = (float)MatrixcBottoMatrix;
  iDiMatrixTf    = (float)iDiMatrixT;
}


int point::Matrixtx2ptx( int Matrixtx )
{
  return Matrixtx - MatrixcLeft;
}

int point::Matrixty2pty( int Matrixty )
{
  return Matrixty - MatrixcBottoMatrix;
}

int point::ptx2Matrixtx( int ptx )
{
  return ptx + MatrixcLeft;
}

int point::pty2Matrixty( int pty )
{
  return pty + MatrixcBottoMatrix;
}

int point::Matrixcx2pcx( int Matrixcx )
{
  return Matrixcx - MatrixcLeft;
}
```

```
int point::Matrixcy2pcy( int Matrixcy )
{
 return Matrixcy - MatrixcBottoMatrix;
}

int point::pcx2Matrixcx( int pcx )
{
 return pcx + MatrixcLeft;
}

int point::pcy2Matrixcy( int pcy )
{
 return pcy + MatrixcBottoMatrix;
}

int point::Matrixx2ptx( float Matrixx )
{
 return Matrixtx2ptx( Matrix2Matrixt( Matrixx ) );
}

int point::Matrixy2pty( float Matrixy )
{
 return Matrixty2pty( Matrix2Matrixt( Matrixy ) );
}

float point::ptx2Matrixx( int ptx )
{
 return Matrixt2Matrix( ptx2Matrixtx( ptx ) );
}

float point::pty2Matrixy( int pty )
{
 return Matrixt2Matrix( pty2Matrixty( pty ) );
}

int point::Matrixx2pcx( float Matrixx )
{
 return Matrixcx2pcx( Matrix2Matrixc( Matrixx ) );
}

int point::Matrixy2pcy( float Matrixy )
{
 return Matrixcy2pcy( Matrix2Matrixc( Matrixy ) );
}
```

```
float point::pcx2Matrixx( int pcx )
{
  return Matrixc2Matrix( pcx2Matrixcx( pcx ) );
}

float point::pcy2Matrixy( int pcy )
{
  return Matrixc2Matrix( pcy2Matrixcy( pcy ) );
}

int point::Matrixx2ix( float Matrixx )
{
  return (int)(iDiMatrixTf*(Matrixx - MatrixcLeftf));
}

int point::Matrixy2iy( float Matrixy )
{
  return (int)(iDiMatrixTf*(Matrixy - MatrixcBottoMatrixf));
}

int point::Matrixtx2ix( int Matrixtx )
{
  return iDiMatrixT*Matrixtx2ptx( Matrixtx );
}

int point::Matrixty2iy( int Matrixty )
{
  return iDiMatrixT*Matrixty2pty( Matrixty );
}

int point::Matrixcx2ix( int Matrixcx )
{
  return iDiMatrixT*Matrixcx2pcx( Matrixcx ) + iDiMatrixT/2;
}

int point::Matrixcy2iy( int Matrixcy )
{
  return iDiMatrixT*Matrixcy2pcy( Matrixcy ) + iDiMatrixT/2;
}

int point::ptx2ix( int ptx )
{
  return iDiMatrixT*ptx;
```

```
}

int point::pty2iy( int pty )
{
  return iDiMatrixT*pty;
}

int point::pcx2ix( int pcx )
{
  return iDiMatrixT*pcx + iDiMatrixT/2;
}

int point::pcy2iy( int pcy )
{
  return iDiMatrixT*pcy + iDiMatrixT/2;
}
```