Центральноукраїнський національний технічний університет
Центр заочної та дистанційної освіти
Кафедра кібербезпеки та програмного забезпечення

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“ Дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор”

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

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

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Заочної та дистанційної освіти*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *магістр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *122 "Комп'ютерні науки"*

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« 21 » грудня 2021 року

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

Пономаренку Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор*

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

затверджені наказом вищого навчального закладу №40-13 від 02.08.2021 року

3. Строк подання роботи до захисту *04.12.2021 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне забезпечення системи генерації та проходження лабіринтів для розробки відеоігор*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*4 аркуша*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

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

КАЛЕНДАРНИЙ ПЛАН

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

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Пономаренко А.С. Дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній магістерській роботі розроблено програмне забезпечення, яке призначено для системи генерації та проходження лабіринтів та приклади пошуку шляху для розробки відеоігор.

Метою розробки є програмне забезпечення системи генерації та проходження лабіринтів для розробки відеоігор.

Результат роботи – програмна реалізація системи генерації та проходження лабіринтів.

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

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

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

Програму розроблено на мові програмування C#.

Ключові слова: комп'ютерна інженерія, генерація, лабіринт, відеогра.

ABSTRACT

Ponomarenko AS Research and software implementation of the system of generations and the passage of mazes for the development of video games. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this qualifying master's thesis developed software that is designed for the system of generation and passage of mazes and examples of finding a way to develop video games.

The purpose of the development is the software of the system of generation and passage of mazes for the development of video games.

The result is a software implementation of the system of generation and passage of mazes.

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

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

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

The program was developed in the programming language C#.

Keywords: computer engineering, generation, maze, video game.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ	–	Програмне забезпечення
Unity	–	Ігровий рушій
АЗ	–	Апаратний засіб
Blender	–	Редактор моделей
IK	–	Інверсна кінематика
MG	–	Генерація лабіринтів (Maze Generator)
ТЗ	–	Технічне завдання
ПЗ	–	Постановка задач
RIC	–	Resetting of initial conditions
RPG	–	Role playing game
ШІ	–	Штучний інтелект
OS	–	Операційна система

ВСТУП

Актуальність теми. У даній роботі ставиться завдання розглянути сучасні інструментальні засоби для розробки ігор. Проведено аналіз найпопулярніших ігрових рішень сучасності, виконання завдань з вивчення, будови, проходження лабіринтам в ШІ, фреймворків та конструкторів ігор.

Репутація комп‘ютерних ігор зростає кожного року. Ігрова індустрія постійно розробляє нові додатки для пришвидшення процесу розробки ігор. Велика різноманітність, конкурентоздатність, специфічність розробки, зміна актуальності технологій, які вже існують, дозволяють виконання задач вибору для розробки відеоігор неоднозначною.

У більшості випадків рішення алгоритмів невідомо на перед. Немає і точного критерію визначення комп’ютером «розумності». Штучному інтелекту було поставлена маса гіпотез, такі як: тест Тюрінга чи гіпотеза Ньюела С. Нині існує велика кількість підходів як до розуміння задач штучного інтелекту, так і до створення інтелектуальних систем. Штучний інтелект вивчає методи рішення завдань, які потребують розуміння людини. Ціль роботи – навчання, розв’язання тестів для інтелекту. Це передбачає розвиток способів рішення задач за аналогією, методів дедукції та індукції, занесення та збереження базових знань і вміння їх використовувати.

Мета й завдання дослідження. Метою даної роботи є дослідження та розробка методів, алгоритмів та програмного забезпечення системи генерації та проходження лабіринтів для розробки відеоігор. А також збір теоретичних відомостей про сучасні інструментальні засоби для побудови ігор, вирішення задач для створення тесових проектів ігор-прототипів, генерації різних лабіринтів та можливість знаходження кінцевої точки у лабіринті.

Дослідження програмного забезпечення системи генерації та проходження лабіринтів для розробки відеоігор, є актуальною задачею, яка потребує

вирішення.

Об'єктом дослідження є процеси генерації лабіринтів та пошуку руху у них.

Предметом дослідження є методи генерації лабіринтів та пошуку шляху у них за допомогою компонентів та інтерфейсів системи ігрового рушія Unity та їх оптимізація для роботи на ПК.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії ймовірності та теорії штучного інтелекту.

Наукова новизна отриманих результатів. Згідно обумовлених цілей дослідження у процесі рішення завдань, отримані такі результати:

- Удосконалено методи генерації та проходження лабіринтів на основі штучного інтелекту для застосування у комп'ютерних іграх;

- Розроблено вітчизняний продукт для генерації та проходження лабіринтів у комп'ютерних іграх на основі штучного інтелекту, який є більш легким у використанні, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає у тому, що розроблене програмне забезпечення дозволяє генерувати зовнішнє середовище у комп'ютерних іграх та керувати рухом ігрових об'єктів для пересування по ньому. Розроблені алгоритми можна використовувати в ігровому 3D та 2D рушії для створення основи різних ігор.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, відповідністю отриманих результатів окремими результатами, наведеними у науковій літературі.

З огляду на усе вищезазначене, дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор є актуальною задачею, яка потребує вирішення у магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Завданням дослідження є формування архітектури системи, що дає можливість отримати оптимально вірний шлях крізь лабіринт. Базуючись на популярних рішеннях (бібліотек, алгоритмів) доводячи їх до єдиної системи, яка полягає у виконані набору задач та під задач (від отриманих та перетворених даних у формальний вигляд рішення) з мінімальним втручанням користувача.

Вхідними даними є системи зображення лабіринту та точок початку руху та закінчення, пошуку виходу. Розв'язками задачі є зображенням позначень на лабіринті, оптимально знаходження шляху. В основу розрахунку траєкторії шляху є розробка максимально короткого руху між двома точками координат. Алгоритм пошуку будується на дослідженні графа, починаючі з стартової координат та суміжних вузлів до поки не буду досягнуто кінченого результату (вузла). Багато алгоритмів пошуку шляху пов'язані метою знаходження найоптимальнішого шляху. Наприклад: існує метод пошуку, який потребує достатню кількість часу, для дослідження траєкторії(пошук в ширину). Інші в свою чергу витрачають менше часу. За основу взято дві найбільш головні теми пошуку шляху:

- Дослідження та знаходження шляху між двома вузлами на графі;
- Знаходження за максимально короткий проміжок часу шлях до виходу.

Розроблена систем має ціль створити багатогранну структуру та охоплюючи якомога більше алгоритмів знаходження і розпізнавання рух крізь лабіринт для досягнення кінцевої точки виходу. Вони складаються з 4-х основних компонентів:

- Отримання зображення лабіринту (модуль);

- Перетворення та дослідження зображення лабіринту для досягнення зручної структури, легкої маніпуляції (модуль);

- Для розробки 2D та 3D додатків багатоплатформений модуль, що має змогу працювати на ОС Windows і OS X. Створені за допомогою ігрового рушія Unity (Android, IOS, Linux, Wii, PS3, Xbox 360 та інші.)

- Формування результату дослідження (модуль).

Багато галузей беруть за основу лабіринт і роблять його визначним атрибутом - це комп'ютерні ігри. Значна кількість ігор, пригодницьких жанрів і аркади використовують лабіринти. Також дуже часто спостерігаються в інших жанрах: екшен, логічні ігри, іноді в симуляторах. Додатково перед користувачем ПК постає дві задачі: Дослідження, знаходження шляху крізь лабіринт в ігрових додатках і розробка самого лабіринту для особистих додатків.

Однозв'язна схема займає значну необхідність в побудові лабіринту, в основу якої входить – прокласти шлях до кінцевої точки локації і слідувати до неї, перешкодами для гравця постають загалом стіни.

За звичай лабіринт базується на будь-якої структури (двомірний чи тривимірний локація/просторі), із шляху до виходу з обходженням заплутаних перешкод, або ті шляхи, що ведуть до глухого кут). У старовині часи в народі древніх греків під лабіринтом малось на увазі великий за площею широкий простір, що складається з багато кількості просторів, залів, камер та переходів, у основі яких покладено різноманітні схеми з завданням не дати виходу людині котра потрапила до лабіринту та заплутати його.

Лабіринт поділяються за походженням на дві категорії: природні та штучні.

- До створення природного лабіринту відношення людині не якого значення не має, все було створено вже давно природою, наприклад: природні печери.

- Розробка штучного лабіринту людиною поділяться на с відому чи несвідому.

Ігрові лабіринти в основу ігор часто входить сам лабіринт, виконаний в 2D і 3D просторі. Приклади таких ігор зображені на малюнках 1.3 і 1.4.

База алгоритму складається з формування нових рядків. У кожному рядку є однакова кількість клітинок, яка формуються на початку. Множену клітинок використовують для контролю переходу між клітинками. Клітини однієї множини з'єднанні між собою на момент генерації, вони ізольовані між собою в різних частинах лабіринту. Клітина може бути присутня або відсутня певна кількість сторін, стін правної та ніжної. Взагалі формування стінок відбувається випадково, певні правила гарантують відсутність циклів у лабіринтів.

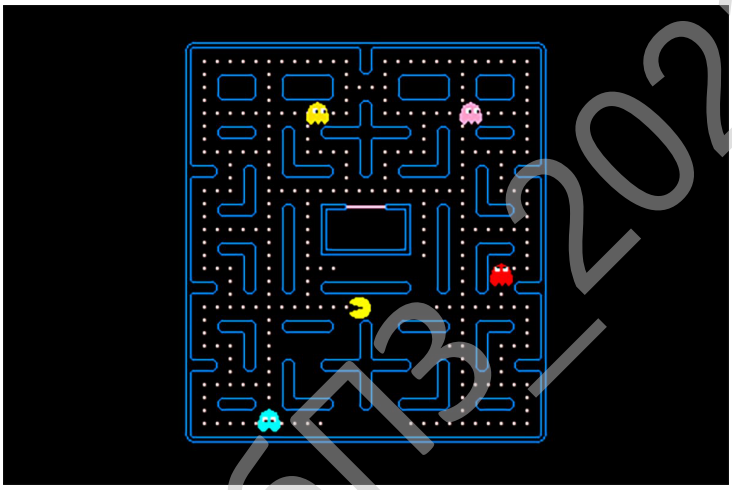


Рисунок 1.1 – Приклад лабіринту в 2D – гра

Рисунок 1.2 – Приклад лабіринту в 3D – гра

Комп’ютерну графіку можна поділити на наступні види: Комп’ютерна двовимірна 2D гра складається з ряду об’єктів, які моделюються лише в двухвимірному зображені. Згідно представленим типом інформації притаманні графічні та алгоритмічне обробки зображення. Дуже часто використовують програми друку та малювання, наприклад: картографія, маркетинг, креслення, реклама та інше. У цих додатках двовимірне зображення — не просто представлення реального об'єкта, але і незалежний культурний експонат з власним семантичним значенням; тому двовимірні моделі є кращими, так як вони дають більш безпосередній контроль над зображенням, ніж комп'ютерна 3D-графіка (підхід якої швидше схожий на фотографію, ніж типографію).

Тривимірна комп’ютерна гра в основу якої входить (3D) графіка обмежується об'єктами в тривимірному просторі. Загалом в результаті зображення являють проекцію, плоску картинку.

Терміни “тривимірний” або 3D використовують для стереоскопічних фільмів і позначає створення ілюзій об’ємного зображення. Розширюються використання засобів комп’ютерної тривимірної графіки, при створені фільмів з відчуттям віртуальної реальності.

За допомогою 3D-графіки імітують фото або відео контент тривимірних образів, які попередньо створюються в пам’яті комп’ютера. Спочатку попередня підготовка створення геометричної моделі сцени, потім налаштування освітлення і знімальних камер. Таким чином можна створити власний віртуальний світ.

В сьогоденні фільми у 3D форматі дуже популярні, їх можна дивитися в звичайних кінотеатрах за допомогою 3D окулярів.

Тривимірна графіка не обов’язково використовує проектування на площину при створенні 3D – принтерів та 3D дисплеїв.

Візуалізація сцени або рендерінг (rendering) полягає в проведенні програмою розрахунків і нанесення на зображення всіх тіней, бліків, взаємних відблисків об’єктів і т. п. і може тривати досить довго, що залежить від складності сцени і швидкодії комп’ютера.

Типовими галузями застосування тривимірної графіки і анімації є:

- комп’ютерне проектування – архітектурне;
- конструювання інтер’єрів;
- створення віртуальних виставок;
- створення тривимірних образів деталей;
- створення конструкцій і комп’ютерні ігри.

1.2 Область застосування

Загалом штучний інтелект являється одним з найрозвинутіших напрямків комп'ютерних наук, який вивчає методи розв'язання задач, для яких не існує способів вирішення. Самонавчання ШІ виконується за рахунок надходження даних та можливістю їх модифікації. Застосування систем ШІ необмежена, користується великою популярністю та є необхідною - від розробки інтелекту для машин/роботів, які мають самостійно приймати рішення: автопілоти чи онлайн-перекладачі.

За допомого систем ШІ при створені комп’ютерних ігор, або написання ботів, ігор зі стратегію, комп’ютер має прорахувати велику кількість варіантів гри і обрати найкращий. Розповсюдженим способом контролю є розробка скриптів, застосування ігрового ШІ проявляється в контролі не ігрових персонажів. Не ігровий персонаж переходить з однієї точки на мапі до іншої, за допомогою пошук шляху в стратегії реального часу. Враховуються перешкоди, ландшафт та інші затемнення. Ігровий ШІ також пов'язаний із динамічним ігровим балансуванням.

На основі аналогічних ознак створюють працюючи систем ідентифікації графічних об’єктів за допомогою ШІ. Можуть розглядатися різні характеристики об’єктів, які підлягають розпізнаванню. Ознаки повинні бути індивідуальними і мати оригінальний розмір та форму об’єкту. Система ШІ виконує сегментацію об’єктів і визначає людей з потоку. Наприклад ШІ є звуковим набором та рукописним текстом в мобільних телефонах, визначення розташування будинку,

який був знятий на камеру мобільного телефону, та проектування внутрішнього плану в системі 3D.

Велику популярність отримали нейроні мережі, завдяки тому що вони звільняють дослідника від зовнішнього втручання в тонкості фізичного процесу і створення складної фізичної моделі. Необхідно розуміти, які фактори впливають на поведінку системи при різних ситуаціях і мати певну кількість фактичних даних, щоб натренувати мережу і дати змогу їй прогнозувати поведінку системи в нових умовах. Можна прогнозувати попит на електроенергію за для того щоб знати як зміниться дані на основі зміни погодних умов і зниження витрат на виробництво.

Тест Тюрінга відрізняє роботу ШІ від робіти інтелекту людини. Він дуже актуальній в умовах, коли ШІ зробив великий крок у своєму розвитку.

Першопроходець комп'ютерної технік і шифрування, Алан Тьюрінг у 1950р. зробив одну з спроб вирішити проблему різниці між людиною та ШІ. За допомогою його тесту зараз оцінюється здатність комп'ютера наслідувати дії людини. В останні роки пройшла інформація про те що програми ШІ пройшли Тест Тюрінга. Це змусило звернути увагу на те що тест не працює в розрізі нашого часу, він застарів. Це дає привід для за непокоєння за наш унікальний статус. Правильно створений Тест Тюрінга виявляє найбільші відмінності для людини та ШІ:

1. Різний інтелект.
2. Відмінності у вірі.
3. Поведінка.

Які не видно серед тварин чи техніки. За допомогою тесту можна визначити здатність комп'ютера розбиратися в людській культурі. Наприклад: Тьюринг зробив дослід в тесті, на " імітаційній грі" в якій чоловік прикидається жінкою, а суддя намагається відгадати, хто є хто, задаючи питання учасникам. Так у тесті Тюрінга суддя намагається відгадати, хто комп'ютер а хто людина.

Більше ніж 50 років і наука зробила великий крок вперед. Було розроблено

програму ELIZA. Ця програма використовувалася для імітації роботи психотерапевта, пацієнти не змогли відрізнити ШІ від справжньої людини. Але винахіднику Джозефу Вайзенбауму було зрозуміло, що ELIZA це насправді жарт. В цьому експерименті він не слідував протоколу тесту Тюрінга. Пацієнти не були попереджені і не було одночасних відповідей реального психотерапевта. Тому будь-який тест Тюрінга повинен мати ще одного учасника – людину, окрім судді які діють як людина.

Існує декілька способів створення ідеальних лабіринтів. Більшість з них створюють за допомогою вирізанням проходів, або шляхом додавання стін. Різниця між “фундаментальними алгоритмами” та “похідними алгоритмами” є різниця в будові поясного дерева, структурою даних, або основним методом використання для вибору наступної комірки. Похідні алгоритми включають в себе конкретні реалізації, які поєднають фундаментальні алгоритми будь-якими способами, наприклад: навмисне зміщення одного типу переходу над іншими. Зазвичай існує незлічена кількість похідних алгоритмів, які можна реалізувати.

Алгоритм Kruskala використовують мінімального поясного дерева. Лабіринт “не виростає”, як дерево, а вирізає сегменти проходу по всьому лабіринту навмання. В підсумку все одно виходить ідеальний лабіринт. Потрібно зберігати пропорційність розміру лабіринту і можливість перераховувати кожне ребро окремо і перемішування списку всіх країв випадковим чином. Якщо клітинки по обидва боки мають однаковий ідентифікатор, тоді існує шлях між ними. В цьому випадку стіну залишають у спокої щоб не створювати петлю. У випадку правильно зробленого алгоритму, він працює дуже швидко. Цей алгоритм складає лабіринти з низьким фактором “річки”, але не таким низьким, як алгоритм Prima. Він використовується для створення мінімального поясного дерева і працює над унікальними випадковими вагами ребр. Алгоритм Prima вимагає зберігання пропорційного розміру лабіринту. Враховуючі однакове початкове число випадкових числи, можна створити ідентичні лабіринти Prima та Kruskala.

При правильному протоколі тест дуже вимогливий і складний для ШІ. На даний час не має машин, які мали здібності розбиратися в людській культурі.

Розробка програмного забезпечення системи генерації та проходження лабіринтів для розробки відеоігор, використання сучасних рішень в багатьох сферах є актуальною задачею, яка потребує вирішення у даній кваліфікаційній магістерській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Дослідження алгоритмів генерації лабіринтів

Окремим класом ігор є пересування по різноманітним локацій с вибором руху напрямку в певне місце знаходження, наприклад: ряд печер з великою кількістю варіантів шляхів, які беруться за основу широкому класі RPG-ігор і т.д. Однакові локації можуть набриднути користувачеві-гравцеві, тому для розробників ігор виникає ряд нових задач, процедурну генерацію лабіринтів, тобто щоб кожне чергове проходження гри проходило на новій згенерований локації. Велика кількість алгоритмів створюються і модифікуються до сьогодні.

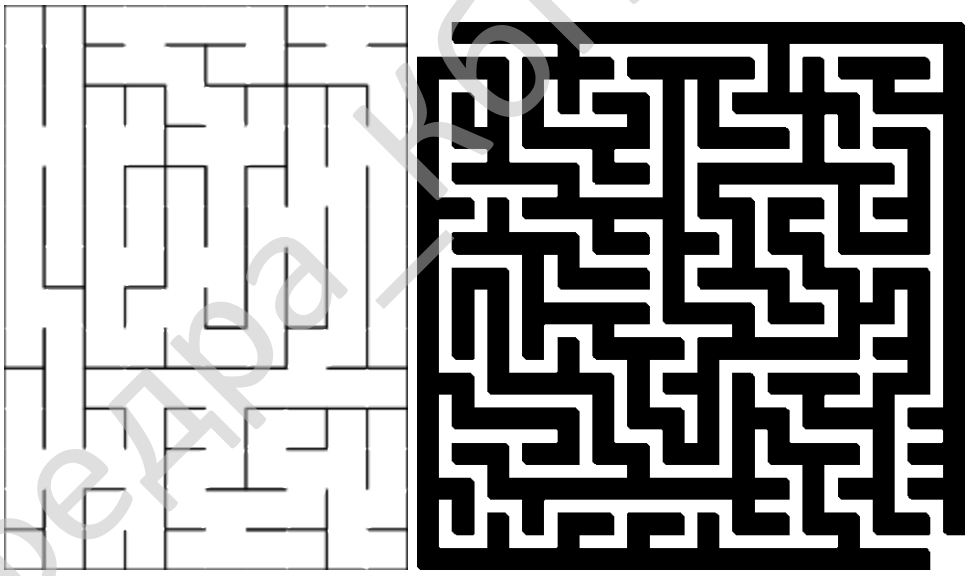


Рисунок 2.1- Приклади лабіринтів

Основою лабіринту являється побудова на клітинному прямокутному полі, їх можна поділити на дві групи: лабіринти з “тонкими” та з “товстими” стінками.

До перших можна віднести стінки розташовані на кордонах клітин, для другого варіанту деякі клітини є непрохідними, тобто перетворюються в стінки. Вони відрізняються за способом зберігання даних про карту.

На даний момент вже існує багато готових рішень для генерації лабіринтів: генератор Oblige приклад серія ігор Doom.

Дослідження зайняло значну ланку у сфері відеоігор та мультимедійних проектів.

В основи досліджені процедури входить автоматична і напівавтоматична розробка і динамічна зміна різних конфігурацій ігор, в тому числі двовимірних та тривимірних зображень (ефектів, звуків, музики, персонажів та інші.)

У процедурній генерації є багато переваг в порівнянні з самостійним втручанням у налаштування рівнів:

- різноманітність: випадкова створення світів без повторів розплодження перешкод, або об’єктів;

- швидкість розробки.

Розробнику не треба втручатись у розстановку блоків на рівнях і придумувати різноманітні ходи, досить обрати алгоритм описаний для генерації об'єктів, і гра виставить самостійно всі об’єкти на локації.

Також використовується для генерації Шум Перліна, який становить середнє значення кольору пікселів, на прикладі локацій (ландшафту). Існує елементи випадковості та велика кількість малих частин пікселів. За допомогою Шум Перліна можна створити в будь-якій кількості різноманітних вимірів. Зображення може бути різне від простого до складного, легко розібрати на графіку.

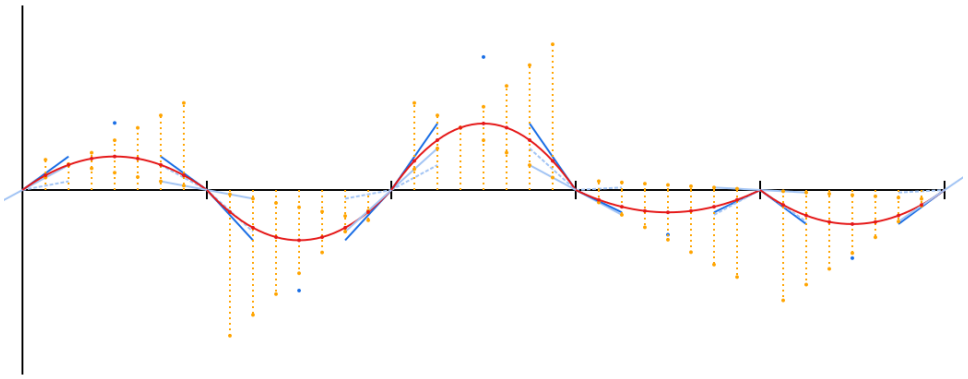


Рисунок 2.2 – Графік Шуму Перліна

Алгоритм Ейлера. Дозволяє розробляти лабіринти які мають лише один шлях між двома точками. Він дуже швидкий і використовує пам'ять ефективніше ніж інші алгоритми, такі як Kruskal та Prim. Він потребує пам’яті пропорційна числу рядків. Це дозволяє створювати лабіринти великих розмірів, при мінімальних розмірах пам’яті. Лабіринти класифікуються за семи різними класифікаціями: текстура, гіпервимірність, розмірність, топологія, фокус, теселяція. маршрутизація. При створені лабіринту можна взяти по одному предмету з кожного класу в різних комбінаціях. Наприклад: Гі перлабіринт включає в себе об’єкт що розв’язується, він складається з ліній, де підчас згинання та переміщення шлях утворює поверхню. Гіперлабіринт використовують лише в середовищі іперлабіринт використовують лише в середовищі 3D або більшого розміру. Де умовою входу до гіперлабіринту є лінія. Він може бути великих розмірів, збільшує розмірність розв’язаного об’єкту рішенням, якого є площина, де підчас його переміщення шлях за ним утворюється тверде тіло в такому випадку Гіперлабіринт другого порядку може існувати в середовищі 4D або вище.

В класі Теселяції використовують геометрію окремих комірок з яких складають лабіринти. Теселяція буває таких типів:

1. Ортогональна. Виглядає як прямокутна сітка, де клітина має проходи і перетинаються під прямим кутом;
2. Дельта. Будується на основі трикутників, що з’єднуються між собою і

можуть мати до трьох з’єднаних проходів;

3. Сигма. Лабіринт складається із шестикутників, з’єднаних між собою, де кожна клітина може мати до 6 каналів підключення до неї.

4. Зета. Розміщення на прямокутній сітці, в якій діагональні проходи під кутом 45 градусів між клітинками дозволенні на додаток до горизонтальних і вертикальних.

5. Іпсілон. Це лабіринти які складаються з взаємозв’язаних квадратів і восьмикутників, де кожна клітинка може мати 4 або 8 можливих проходів пов’язаних з нею.

Клас текстури є тонким і описує стилі проходження в будь-якій маршрутизації, в будь-якій геометрії. Текстура включає в себе: зміщення, біг, Elite, симетричний лабіринт, однорідність. Коефіцієнт “елітароності” вказує на розмір лабіринту і на довжину рішення. Зазвичай Елітарний лабіринт має коротке рішення він може бути набагато складнішим, ніж неелітарний.

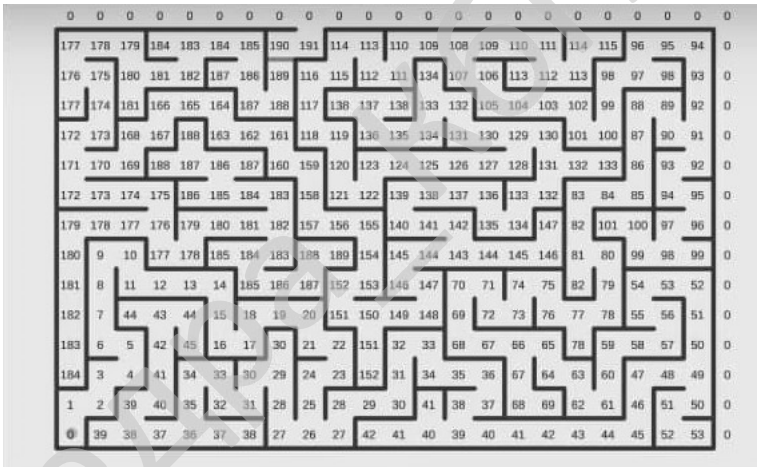


Рисунок 2.3 – Приклад структури розробки лабіринту Ейлера

Включає **наївний алгоритм** в себе генерацію, коли розташування відбувається невидимо для очей користувача. Досить поставити граничні точки розміру клітинки. Їх межі та кількість області, випадковим способом розміщує по координатам в просторі та його розміри. Потім достатньо координати з’єднати з

ними. Недолік: для перевірки перетну треба окрема функція.

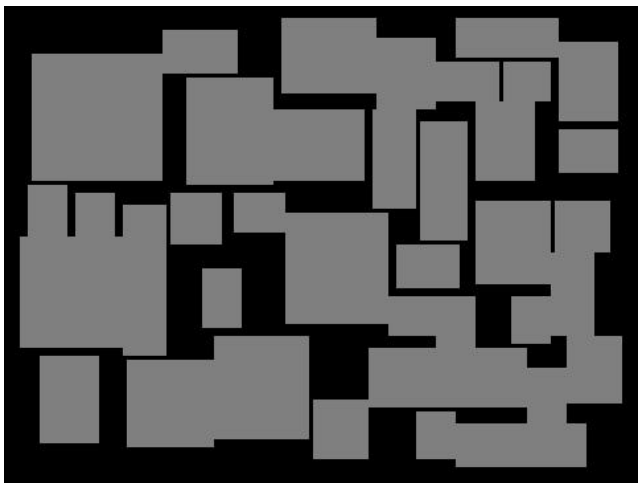


Рисунок 2.4 – Лабіринт за допомогою наївного алгоритму

Binary Space Partitioning. Розділення області в бінарному типі. Він також дозволяє обходити перетинання клітинок/кімнат у стадії розміщення на області генерації лабіринту. Ділячи поле частинами, порівняно з листями дерева, де розташовує згенеровану кімнату. Алгоритм використовує максимально різні конфігурації кімнат.

Клітинний автомат для Генерації простору кімнат/лабіринтів. Суть запропонованого алгоритму полягає в реалізації всього двох кроків: спочатку все поле заповнюється випадковим чином стінами – тобто для кожної клітини випадковим чином визначається, чи буде вона вільною або непрохідною – а потім кілька разів відбувається оновлення стану карти відповідно до умов, схожих на умови народження / смерті в «Життя».

Генерація в тривимірному просторі. Bake Your Own 3D Dungeons With Procedural Recipes» - основна складність якого полягає в правильній орієнтації елементарних частин лабіринту: коридорів, кімнат і сходів.

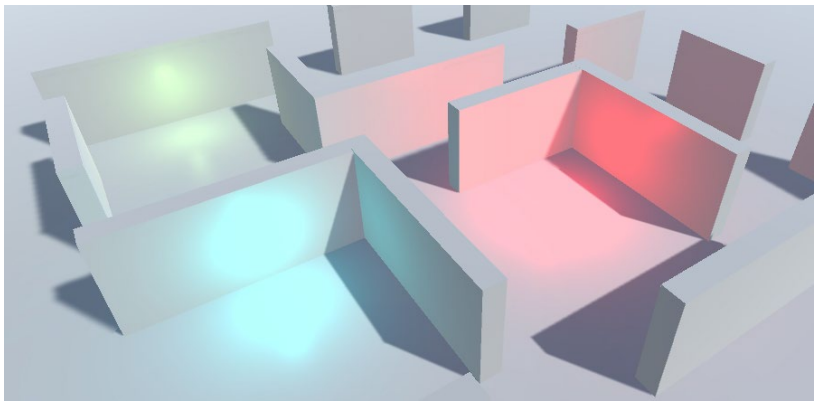


Рисунок 2.5 - Генерація в тривимірному просторі

Кожен з модулів зберігає інформацію про своїх входи і виходи, а також їх орієнтації: перш ніж з'єднати чергову пару елементарних частин, їх потрібно правильно орієнтувати. Зокрема, автор пропонує зберігати x-, y- і z- орієнтацію модулів, щоб потім з'єднувати їх за такими правилами: їх y-осі повинні збігатися, a x i z - мати протилежний напрямок. Це, природно, ставить питання про зберігання інформації про згенеровані карти. Крім того, не вирішена проблема з перетином приміщень - тому ця стаття може бути лише відправною точкою для дослідження питань генерації тривимірних алгоритмів.

Дослідження алгоритмів пошуку виходу з лабіринту

Алгоритм «однієї руки» Найпоширеніший метод для проходження лабіринту є правило на основі алгоритму "однієї руки": рухаючись по лабіринту, треба весь час використовувати праву чи ліву рук його стіни . Він використовувався ще за часів древніх греків. Для проходження лабіринту потребує багато часу, але результат буде позитивним.

1) Роботи алгоритму почнеться з знаходженням першої стінки, в доль якої він буде рухатись. Для цього йому лиш потрібно рухатись вперед, поки не потрапить до перешкоди.

2) Після попадання до першої перешкоди алгоритм починає діяти за правилом “правої руки”.

4) Якщо проходу не знайдено по праву сторону стіни він повертає ліворуч. У разі потрапляння в глухий кут, ще раз повертає ліворуч, таким чином на 180 градусів, і йде в зворотному напрямку.

Якщо ж лабіринт містить окремі стінки, то, застосовуючи правило "однієї руки", не завжди підходить. Лабіринти з окремими стінками і з замкнутими маршрутами називаються багато зв'язними. При цьому багатопланові лабіринти можуть бути розділені на дві групи: без «петлі» навколо цілей (замкнутий маршрут не проходить навколо цілей) і з замкнутої «петлі» навколо мети.

Хвильовий алгоритм на основі пошуку та побудови шляху між двома точками в лабіринті. Він поділяється на 2 етапи:

1) З стартової точки в чотирьох напрямках хвиля починає свій рух в середину лабіринту. Збоку здається наче великий фронт рух великою кількості варіантів пошуку руху. Приклад на малюнку 2.4.

Перші елементи фронту супроводжується вторинним потоком вони і являються джерелом переходів по області. Процес триває до останнього кінцевого положення елементу на області дослідження.

2) Розробка траєкторії рух по площині, їх побудова починається з стартового елементу закінчуючи кінцевим.

Позитивний момент алгоритму в тому що шлях буде знайдений в лабіринті різної складності, при наявності виходу. Недолік, значне використання об'єму пам'яті.

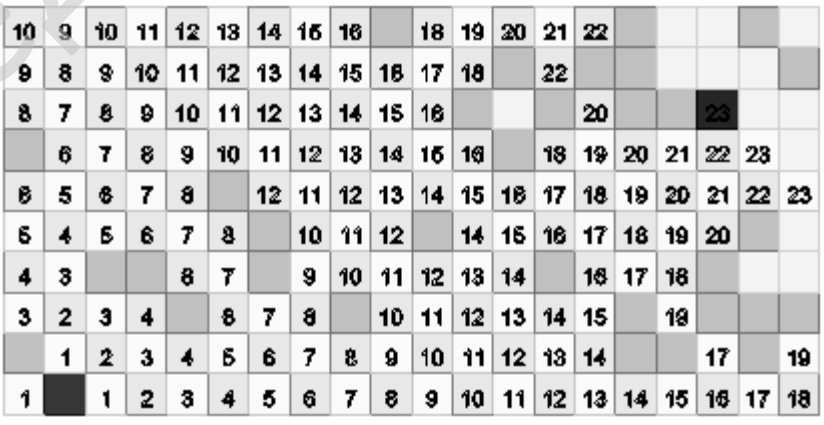
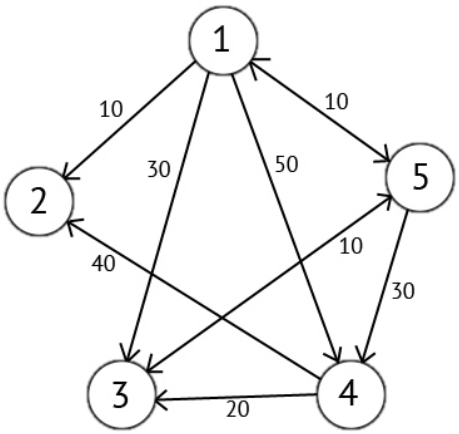


Рисунок 2.4 – Приклад роботи хвильовий алгоритм

Дослідження **Алгоритм Дейкстри** базується на пошуку всіх варіантів найкоротшого шляху за допомогою заданої вершини графа. При достатньої кількості необхідної інформації є можливість дослідити максимально вигідний шлях, послідовність руху обходячи перешкоди. До недолік відносять наявність в графі дуги негативної маси.



	1	2	3	4	5
1		10	30	50	10
2					
3					10
4		40	20		
5	10		10	30	

Рисунок 2.5 – Приклад графу, та його представлення у виді матриці

Обирається в якості джерела вершину 1. Пошук найоптимальнішого маршрути з вершини 1 у вершини 2, 3, 4 і 5.

Даний алгоритм поетапно перебирає всі вершини графа і призначає їм комірки, які є відомим мінімальним відстанню від верши ни джерела до конкретної вершини.

Дослідження комп’ютерних ігор з лабіринтами

Найбільш актуальною темою на теперішній час є розвиток функцій мозку за допомогою досліджень комп’ютерних технології. Позитивний вплив на розвиток цих функції, практикується за допомогою тренажерів, ігрових симуляторів: швидкість мислення, пам'ять, увага, стратегія. Поєднання приємного с корисним викликає у користувача великий інтерес до розвитку. Об’єктом дослідження постають процеси тренування пам’яті, стратегічного та логічного мислення, мозкової активності з використанням сучасних комп’ютерних ігрових тренажерів . Предметом дослідження вбачаємо засоби та методи розробки

комп'ютерних ігор з використанням складних алгоритмів генерації ігрового середовища. Головним завданням є розробка ігрового тренажера на основі системи генерації випадково ігрового лабіринту.

Проаналізуємо наявні аналоги на ринку (зокрема Play Market): Labyrinth 3D – гра генерує тривимірні випадково згенеровані лабіринти з різними стилістичними оточеннями. Коли гравець знайде вихід з лабіринту, йому стануть доступні наступні рівні. Гра подана з видом від 1 -ї особи. 3D Maze (The Labyrinth) – додаток дозволяє грати на попередньо згенерованих лабіринтах та має рівень з рандомною генерацією лабіринтів. Доступними є 2 типи тривимірних лабіринтів: квадратний та круговий. Гра подана з видом від 3 -ї особи . Labyrinth - гра генерує тривимірні рандомні лабіринти з режимом виживання, який додає до лабіринту небезпечні об'єкти.

У першу чергу, мета розробки ігрового додатку полягає в наданні користувачу можливості відпочити, відволіктися від повсякденних турбот та водночас провести час з користю, покращуючи пам'ять, логічне та стратегічне мислення.

Конкурентні переваги розробленого ігрового додатку «Dionis Maze»:

- 100% лабіринтів генеруються випадково, гравець завжди може дістатися до виходу;
- використання сучасної 3D графіки;
- наявність зручного інтерфейсу користувача; генерація рандомного лабіринту з використанням швидкого та ефективного за споживанням пам'яті алгоритму;
- наявність широкої бази ігрових об`єктів (пастки, телепорти, кульки та інші);
- захопливий ігровий процес;
- наявність багаторівневої системи складності;
- можливість перемикання між режимами огляду від 1-ї та 3-ї особи;
- кроссплатформеність, яка забезпечується засобами Unity3D.

Програма має модульну структуру. Роботу програми забезпечують такі класи:

- CameraMove;
- Destroyer;
- GameManager;
- MainMenu;
- MazeBuilder;
- MazeGenerator;
- PlayerMovement;
- trapSprike;
- WallTrapDestroyer.

CameraMove – клас, що відповідає за роботу камери, її переміщення в просторі за гравцем. Destroyer – клас, призначений для знищення ігрових об'єктів. GameManager – клас, що контролює ігровий таймер, рахунок користувача (кількість зібраних жовтих та синіх кульок), у випадку знайдення виходу відображає результат та зберігає його в PlayerPrefs. MainMenu – клас, що реалізує головне меню, яке відображається після запуску програми. MazeBuilder – клас, що реалізує функцію розміщення об'єктів у просторі (стінок, пасток, гравця, виходу з лабіринту, бонусів). MazeGenerator – клас, що виконує безпосередньо генерацію лабіринту за алгоритмом Еллера. Об'єкти даного класу при їх оголошенні приймають аргументи розмірності лабіринту, які попередньо визначаються в класі MazeBuilder. Це передбачено в конструкторі класу, який також ініціалізує матрицю – лабіринт. PlayerMovement – клас, що реалізує рух гравця в лабіринті, а також його взаємодію з об'єктами лабіринту (стінками, пастками). TrapSpike – клас, що реалізує управління анімацією об'єктів «пасток-шипів». WallTrapDestroyer – клас, що реалізує знищення тимчасових стінок, які з'являються при потраплянні гравцем у «замуровану пастку». На рисунку 2.6 подано модель ігрового програмного продукту.

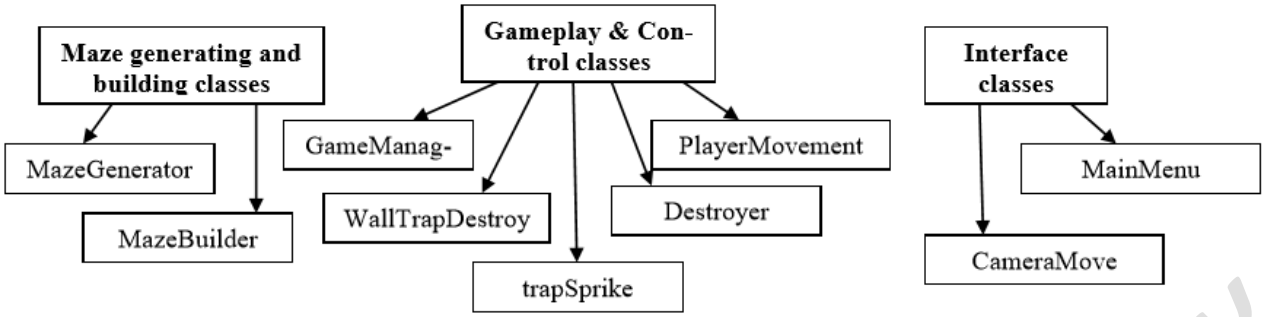


Рисунок 2.6 – Модель програмного ігрового продукту

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програма написана мовою C# в середовищі Microsoft Visual Studio 2021, у склад якої включено інтегроване середовище розробки та різноманітну кількість інструментів засобів для створення ПЗ. За допомогою інтегрованого середовища е велика кількість функцій для розробки програм с візуальним інтерфейсом (тривимірної, двовимірної графіки), консольні програми, підтримка технологій Windows Forms, веб додатків та інші. Кросплатформеність написаної мови програмування скриптів, з подальшим керування на багатьох платформах.

Означення **Ігрового рушія** полягає у ряді функцій, конфігурацій, під конфігурацій та компонентів розробки відео ігор на базі комп'ютерних систем.

Один із сучасних рушіїв Unreal Engine представлений у виді різних об'єктів, які мають характеристик, класи використовує модульну систему залежних компонентів, підтримує різні системи рендерінгу. Використовує технології Windows Live, Xbox Live, GameSpy та інші, для можливості передачі даних по мережі та обробки зображень, адаптивність під різні платформи.

Для порівняння було обрані два популярних ігрових рушія для розробки відеоігор.

Таблиця 2.1 - Порівняльна характеристика ігрових движків

Критерій	**Unity**	**Unreal Engine**
Короткий опис	Багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосунки працюють під системами Windows, OS X, Android, Apple iOS, Linux	Набір інструментів для розроблення ігор, що має широкі можливості: від створення двох-мірних ігор на мобільні до AAA-проектів для консолей
Вартість	Безкоштовна для персонального користування (при виручці за иг-ру до 100000 $ в рік).	Unreal Engine 4 поширюється безкоштовно. До тих пір, по-ка випустите перший комерційний продукт на основі UE4, далі йде платити процент від продажів.

Продовження таблиці 2.1

Платформа	Ios, android, windows, Windows phone, Mac, linux, webgl, ps4, psvite, xbox one, Wii U, Nintendo 3Ds, Oculus Rift, Google Card-board Android & ios, steam, Playstation VR, Gear VR, Win-dows moxed reality, Daydream, Android TV, Samsung smart TV, tvos, Nintendo Switch, fireos, Fa-cebook Gameroom, Apple arkit, Google arcore, Vuforia	Windows PC, Mac, Linux, ios iAn-droid, HTML5. Також є підтримка Віртуальної реальності для Oculus Rift. Крім цього UE4 підтримує Xbox One і playstation4 (включаючи Project Morpheus)
Розробка ігор види	Ігри будь-якого жанру, сімулято-ри та ін.	Ігри (2D-3D; RTS, Action-RPG, Shooter, Racing, MMO-гри і лю-бій інший жанр і напрям), симулятори і навіть програмне забезпечення. Ви можете викорис-товувати UE4 для архітектурної ві-зуалізації і багато іншого.

Продовження таблиці 2.1

Мови програмування	C #, javascript	C ++, Blueprint
Інтерфейс	Drag & Drop інтерфейс	Не має можливості перетягувати об'єкти прямо у вікно редактора.
Відкритість коду	Закритий	Завдяки відкритому вихідному коду, ви можете самостійно додати підтримку додаткових-них пристроїв, або ж оптімізі-ровать / доповнити вже існую-щие.

Після проведеного дослідження , був обраний ігровий движок Unity 2021.

Редактор Unity має зручний та привітний інтерфейс (Drag & Drop), який піддається гнучкому налаштуванню, поділяються на різні вікна с великим функціоналом для більш комфортного користування. Рушій включає в себе декілька мов програмування: C #, JavaScript (UnityScript). В Unity проект поділяється на конфігурацію (рівні, окремих файлів), який має основу ігрового простору з різноманітною кількістю об’єктів, модифікацій, опцій. В області розробки існують як звичайні об’єкти так і порожні (до складу входять моделі). Кожен об’єкт містить ряд компонентів для взаємодії кодів, скриптів. Наприклад компонент Transform зберігає в складі координати точок розташування, повороту та розмірів по трьох осях. Mesh Renderer діє як ви дима оболонка (встановлена за замовченням).

Підтримка фізики рідини та твердих тіл, зокрема також тканини типу

Ragdoll (тканина іграшка). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна — буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того він містить компонент для створення анімації, яку також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

Огляд програмних засобів для розробки 3D моделей

Autodesk Maya являється найпоширенішим програмним засобом для розробки анімації в 3D просторі. На даний час займе лідерську позицію серед програм для створення анімації в кіноіндустрії та ігроіндустрії. Maya дозволяє керувати різноманітними системами моделювання, досить якісну анімацію моделей об’єктів та персонажів. Також є можливість створювати різного рівня ефекти візуального зображення. Програмне забезпечення має зручний інтерфейс, який підходить для користувача з будь -яким рівнем знань в 3 D/2D. Для художників існує можливість користуватись багатою кількістю пензлів, формування та редагування об'єктів в процесі інтерактивного досліду, що схоже на малювання пензлем на листі. Предмети, одяг, вологість, повітря, динамічна передача руху тіл, об’єктів з достатньо сильним планом дій, надало Maya велику популярність серед розробки візуальних ефектів. Maya і досі використовується у багатьох ланках сучасних професій, наприклад: розробка кінофільмів та мультфільмів, рекламних роликів, комп'ютерних ігор, архітектура, розробки бізнес-презентацій, дизайн.

Poser 3D є дуже легким у сприйняті пакетом програмного забезпечення для створення максимально реалістичного вигляду моделі та подальшого руху в

просторі. За допомогою ключових кадрів на шкалі часу є можливість зробити як найшвидше зміну пози та моделювання таймеру руху моделі по тракторі заданою користувачем. Є змога підключення великої кількості бібліотек с готовими анімаціями з подальшою їх модифікацію.

Існує підтримка розробка анімації, імпорт виконується в формат BVH motion capture-файлів. BVH motion capture- суть роботи полягає у прикріплені сенсорів-сигналів на тіла людини для фіксації руху та передачі реалізму.

Досі існує можливість скористатись готовими моделями з бібліотек Poser та модифікувати їх для нових, різних витворів мистецтва. В Poser можна розробляти анімацію моделі, та імпортувати в інші програмах для тривимірної графіки, зберігання виконується у форматі .obj. Для 3D-художник Poser має велику кількість персонажів в позі, одягнених та готових до анімації. Одяг переміщається разом з персонажем. Реалістична передача вигляду персонажу від зачаски до емоції обличчя.

Освітлення сцени виконується за допомогою об'єктів проекцією світла, Можливість встановлення точкового джерела чи комбінацію різних джерел (за замовченням). Колір, баланс білого, експозицію освітлення також можна змінювати, при необхідності. Збережені в форматах JPEG, BMP та PSD. Анімації зберігається як MOV, AVI чи як послідовності зображень PICT, BMP. Моделі експортуватися в формати 3DS, OBJ та VRML.

Щорічне оновлення додаються до функціоналу систем програмного забезпечення. Останнім часом велика кількість оновлень дозволила користувачам працювати у достатньо комфортному інтерфейсі. При виборі редактора більшість людей базується на характеристиці та відгуків користувачів ПЗ. Розвиток на даний час являється перспективним у таких напрямках:

— покращання зручності користування інтерфейсом;

— додавання кращого інтелекту в роботу інтерфейсу;

— підвищення швидкості роботи програми;

— підвищення якості створення моделей, та трансформації зображень;

— забезпечення імпорту та взаємодії між ПЗ та мінімальна втрата деформації якості зображеня.

Один із зручних, легких у навчанні та безкоштовних програм для роботи з моделюванням є Blender 3D .

Особливістю даного продукту є не тільки його безкоштовність, а й висока швидкість передача зображення, малий розмір файлів, легкий у розумінні інтерфейс, існування версій для багатьох ОС . У Пакет програми входить: динаміка твердих об’єктів, передача фізик рідинних та м'яких об’єктів/моделей , взаємодія з мовою програмування Python, велику кількість бібліотек та систему гарячих клавіш. У версії 2.61 додалась можливість переміщення та відстеження камери.

До програмного забезпечення Blender входять ряд функцій:

- Велика кількість тригонометричних примітивів, до яких в ключено полігональна передача зображення об’єкту;
- Режим subdivision surface для швидкого та якісного моделювання гладких, гумових обектів;
- Криві Без’є;
- Векторні шрифти та їх різноманітність редагування;
- Інтеграція з YafRay;
- Арматурна (скелетна) та сіткова деформація;
- ключові кадри, можливість нелінійних анімацій;
- timeline, vertex weighting, constraints;
- Фізика м'яких та рідиних тіл, (колізій форми моделі);
- Ігровий рушій Game Blender.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну магістерську роботу, реалізації підлягає дослідження програмного забезпечення системи генерації та

проходження лабіринтів для розробки відеоігор".

В процесі розробки кваліфікаційної магістерської роботи необхідно виконати наступний обсяг проекту:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Під процесом проектування архітектури ПЗ розуміється розроблення, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування - перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи.

Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури.

Проектування архітектури системи може проводитися різними методами (стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об’єктної й інших моделей за допомогою відповідних конструктивних елементів.

3.2 Розробка структурної схеми

Система повинна являти собою сукупність елементів. Визначенням (головним властивістю) системи є її цілісність: комплекс об'єктів, що розглядаються в якості системи, повинен володіти загальними властивостями і поведінкою. Очевидно, необхідно розглядати і зв'язку системи із зовнішнім середовищем. У найзагальнішому випадку поняття «система» характеризується:

- наявністю безлічі елементів;
- наявністю зв'язків між ними;
- цілісним характером даного пристрою або процесу.

Система повинна являти собою сукупність елементів (об'єктів, суб'єктів), які перебувають між собою в певній залежності і складових деякий єдність

(цілісність), спрямоване на досягнення певної мети. Система може бути елементом іншої системи вищого порядку (над система) і включати в себе системи нижчого порядку (підсистеми). Тобто систему можна розглядати як набір підсистем, організованих для досягнення певної мети і описаних за допомогою набору моделей (можливо, з різних точок зору), а підсистему - як групу елементів, частина яких становить специфікацію поведінки, представленого іншими її складовими. На рисунку 3.1 зображена структурна схема розроблюваної системи.

Розроблена система складається з головного меню та кнопок (Генерація рівня, Опції та Вихід)

З рисунку видно, що після запуску програми спочатку відбувається вивід основного вікна програми. Потім користувач обирає одну з наступних дій:

1. Запуск програми.
2. Налаштування.
3. Генерація лабіринту.
4. Перехід до налаштувань.
5. Рух по лабіринту
6. Пошук кінцевої точки Алгоритм А*..
7. Приклади пошуку шляхів
8. Створення власних перешкод
9. Рух до виходу.
10. Звук увимк/вимк.
11. Повернення до головного вікна.
12. Вихід

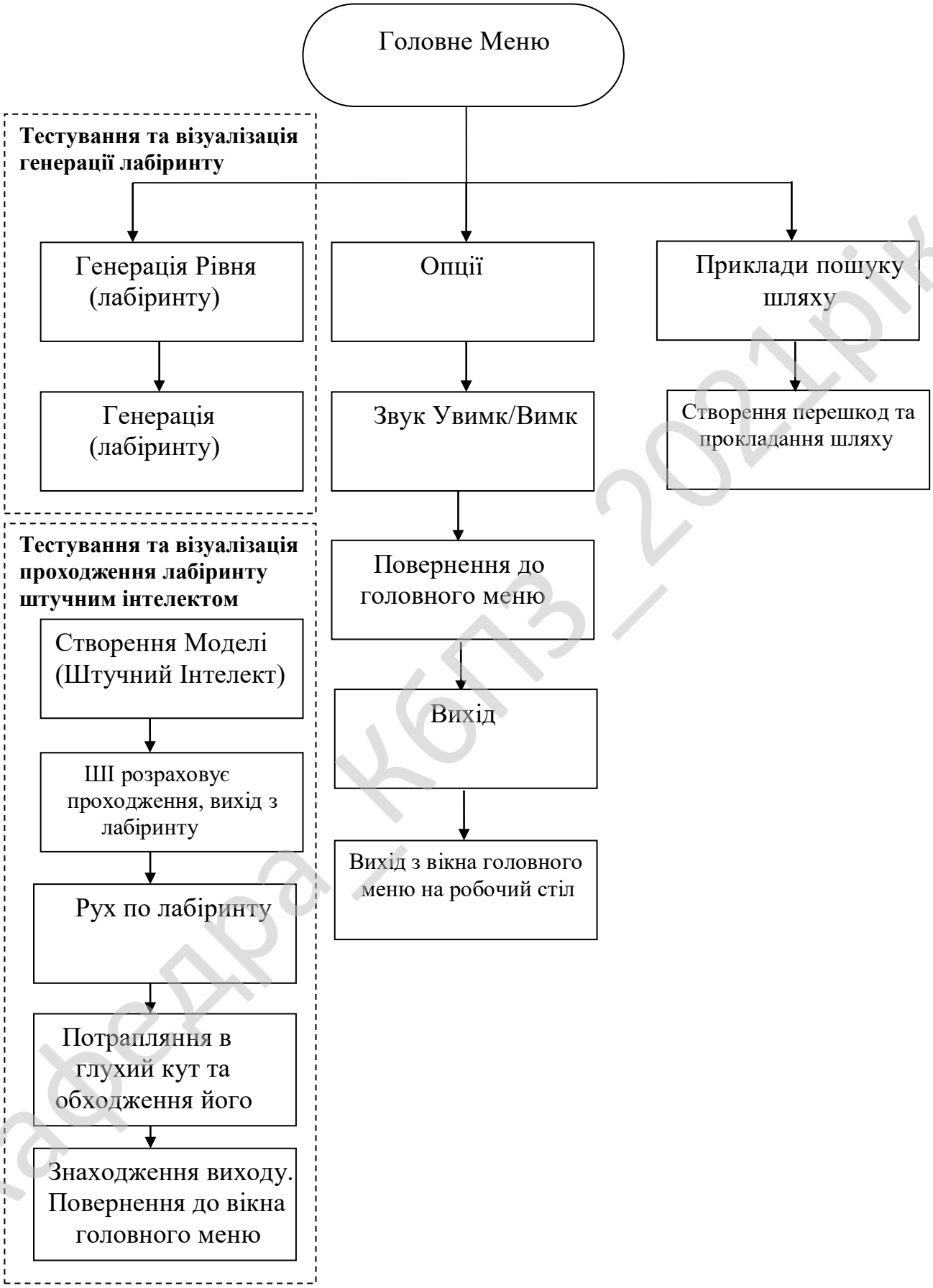


Рисунок 3.1 – Структурна схема системи

Розробка скрипта меню потребує, щоб після натискання кнопки Генерація рівня запускалась сцена з створеним лабіринтом, а після натискання кнопки Вихід з меню, вікно закривалася.

Для MainMenu додаємо новий компонент MenuControls.cs і відриваємо його. Наявність підключити наступних бібліотек :

using UnityEngine.SceneManagement;

За завантаження сцени відповідає SceneManager і у нього є метод LoadScene. Існує кілька перевантажень методу. У підсумку функція буде виглядати наступним чином.

```
public void PlayPressed()
    {
        SceneManager.LoadScene("Generation");
    }
```

метод для виходу з гри:

```
public void ExitPressed()
    {
        Application.Quit();
    }
```

Генерація стінок в лабіринті (Частина коду з генерацією висоти та довжени стінок):

```
MazeGeneratorCell[,] cells = new MazeGeneratorCell[Width, Height];

        for (int x = 0; x < cells.GetLength(0); x++)
        {
            for (int y = 0; y < cells.GetLength(1); y++)
            {
                cells[x, y] = new MazeGeneratorCell {X = x, Y = y};
            }
        }

        for (int x = 0; x < cells.GetLength(0); x++)
        {
            cells[x, Height - 1].WallLeft = false;
        }

        for (int y = 0; y < cells.GetLength(1); y++)
        {
            cells[Width - 1, y].WallBottom = false;
        }
```

Створення тривимірного зображення з двійкових даних є більш складним завданням, побудоване на тому самому принципі роботи. Для того, щоб створити тривимірне зображення, відеокарта спочатку створює структуру зображення з прямих ліній, а далі заповнює ділянки в лініях пікселями, потім включаємо до

зображення параметри фільтрації простору (освітлення, текстуру, кольоровість, тощо). Однак, тривимірне зображення насправді представляє двовимірну структуру, ефект тривимірності якої є результатом мозкової діяльності людини.

Принцип обробки графіки наступний, центральний процесор комп'ютера, який працює з підтримкою певного програмного забезпечення, відправляє інформацію на відеокарту про те, яким має бути зображення. Порт з'єднання з материнською платою, іменований PCI-E або AGP, займається передачею даних з материнської плати на відеокарту. Відеопам'ять тримає інформацію про кожний окремий піксель, а також запам'ятовує і підтримує зображення, яке відображається на моніторі.

Графічна обробка виконується наступними компонентам дивись (рисунок 3.2).

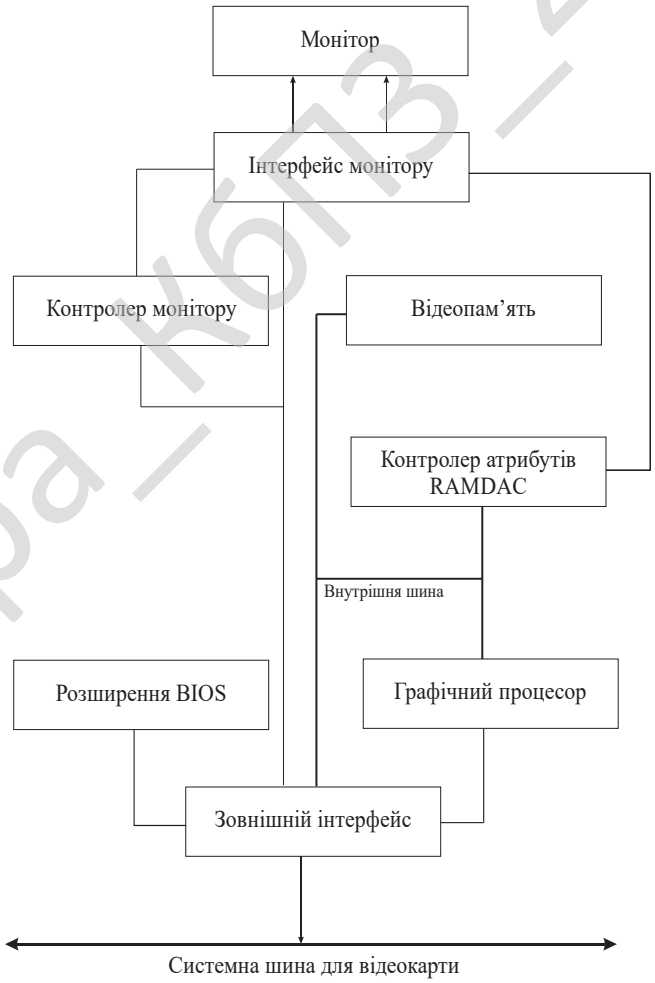


Рисунок 3.2 – Компоненти графічної обробки

- зовнішній інтерфейс;
- системна шина відеокарти;
- розширення BIOS;
- графічний процесор;
- контролер атрибутів RAMDAC
- внутрішня шина відеокарти;
- відеопам’ять;
- контролер монітору;
- інтерфейс монітору;
- екран монітору;
- канали передачі кольору та відео зображень.

Графічні карти з'єднуються з материнською платою через друковану плату. Системна плата відеокарти подає живлення на пристрій і забезпечує зв'язок з центральним процесором. Найбільш потужні відеокарти, як правило, забезпечені додатковим джерелом живлення, яке подається через окремий роз'єм. Нині підключення відеокарти виконується через інтерфейс PCI Express. На даний момент, цей інтерфейс забезпечує найбільш достатню швидкість передачі даних, а також необхідний рівень живлення для карти.

Відеокарта виконує свою взаємодію з іншими компонентами персонального комп’ютера через системну шину для відеоадаптера. Таким чином, відеокарта під’єднується до материнської карти персонального комп’ютера.

Графічний процесор – це компонент відеокарти, який виконує функцію рендера. Також цей елемент відеокарти вирішує, як розміщувати пікселі на моніторі. Процесор графічної карти схожий за своїми функціями з центральним процесором, однак, графічний процесор заточений в більшій мірі на виконання складних геометричних і математичних обчислень, які необхідні для рендеринга зображення. Деякі графічні процесори мають більшу кількість транзисторів, ніж центральний процесор і, як наслідок, володіють більшою продуктивністю. В

процесі роботи, графічний процесор виділяє чималу кількість тепла, тому у відеокарти має бути охолодження.

Контролер атрибутів RAMDAC відеоадаптера призначений для керування кольором зображення, яке виводиться на екран монітору. У текстовому режимі роботи відеоадаптеру він задає колір пікселів в межах символьної матриці на основі змісту байта атрибутів символів. Крім того, в цьому режимі контролер атрибутів дозволяє створювати ефекти миготіння, інверсії кольорів або збільшення яскравості. У графічному 16-кольоровому режимі контролер атрибутів перетворює умовний 4-розрядний номер кольору пікселя, який зберігається у відеопам’яті, в 8-розрядний номер регістру RAMDAC, який вміщує 18-розрядний код поточного кольору.

Відеопам'ять – це внутрішня оперативна пам'ять графічного контролера, в якій розміщені дані, що відповідають зображенню на екрані. У відеопам'яті може міститися, як безпосередньо, растровий образ зображення (екранний кадр), так і окремі фрагменти як в растровій, так і у векторній формі. Відеопам'ять відрізняється від системної оперативної пам'яті більш жорсткими вимогами до ширини шини.

Внутрішня шина є компонентом внутрішнього складу відеокарти, який забезпечує обмін даними між складовими компонентами відеоадаптеру.

Інтерфейс монітору – це апаратний інтерфейс для відображення графічного мультимедіа. Інтерфейс монітору дозволяє передавати цифрові відеодані високої роздільної здатності. Через інтерфейс виконується підключення монітору до системного блоку. Кожен монітор має власний контролер, який керує його режимами роботи. За допомою налаштувань монітору можна регулювати параметри яскравості, контрасності, розширення, тощо. Особливість інтерфейсу монітору та контролеру RAMDAC у тому, що вони працюють незалежно один від одного. Таким чином можна редагувати параметри зображення на апаратному та програмному рівнях.

Відеоадаптер передає дані про колір кадру в системі RGB. RGB представлена, як адитивна колірна модель, що описує спосіб синтезу кольору.

Установка відео та аудіо драйверів

Для установки відео драйверів необхідно дізнатись, яка компанія є виробником відеокарти, встановленої в персональному комп'ютері. Для цього треба запустити диспетчер пристроїв, у якому необхідно відкрити вкладку «дисплейний адаптер». Там буде вказано модель та марку відеокарти персонального комп'ютера.

Якщо виробник відеокарти відомий, то необхідно перейти на офіційний сайт виробника. На сьогоднішній час існує два основних виробника відеокарт: Nvidia та AMD. Ці розробники надають універсальні системи інсталяції та оновлення відеодрайверів.

Установка таких систем проходить у стандартному порядку інсталяції робочих програм, слідуючи інструкціям програми-інсталятора. Такі системи представляють можливість автономного оновлення існуючих драйверів, які потребують лише підтвердження від користувача без постійного завантаження нових драйверів власноручними заходами.

Інсталяція аудіодрайверів є більш простим заходом, оскільки виробників звукових адаптерів набагато більше, тому вони упорядковані в стандартизовані пакети звукообробних систем.

Існує дві найпопулярніщі системи автоматичної установки та оновлення звукових драйверів: VIA HD Audio та Realtek HD. Вони інсталюються у стандартному порядку установки програмного забезпечення. Ці програмні заходи самостійно оновлюють аудіо драйвера системи і забезпечують надійний функціонал роботи драйверів.

Установка даних рішень дозволяє забезпечити функціонал обробки відео та аудіо на програмному рівні.

3.3 Розробка функціональної схеми

На рисунку 3.3 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

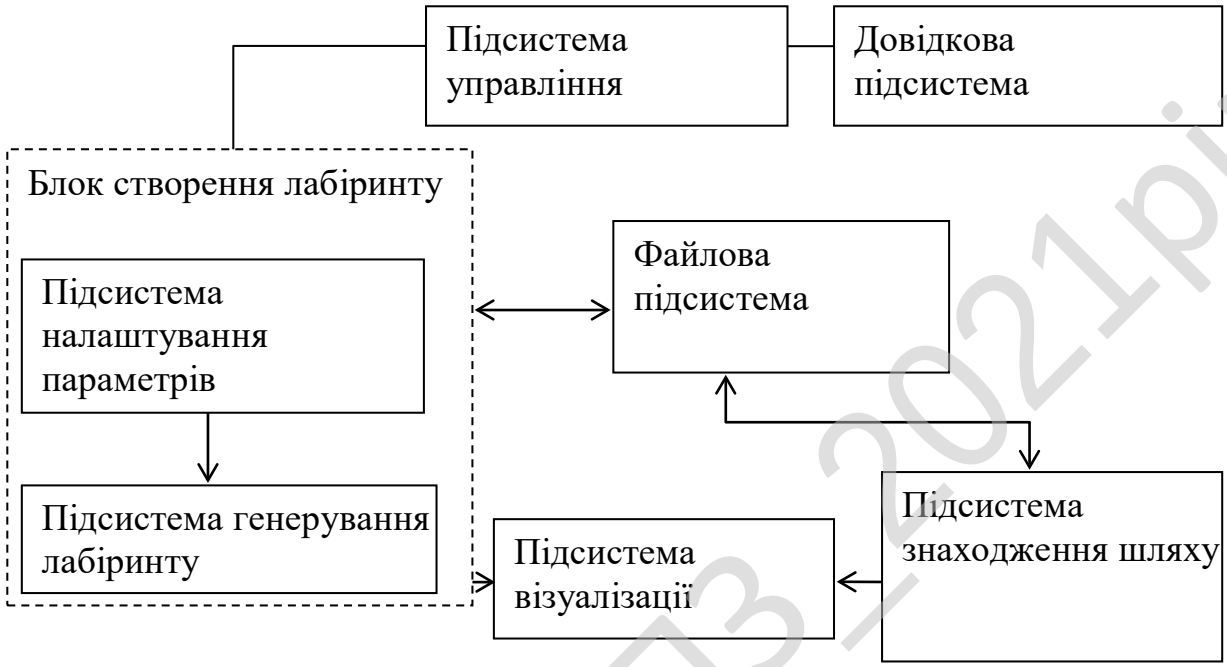


Рисунок 3.3 – Функціональна схема системи

На рис. 3.3 зображена Блок схема взаємодії процесів генерації лабіринту.

Блоки:

– Блок створення лабіринту.

– Блок проходження лабіринту ШІ.

– Блок графічного інтерфейсу.

– Блок візуалізації.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання кваліфікаційної бакалаврської роботи, наведена на рисунку. На рисунку 4.2

зображена діаграма взаємодії процесів серверної частини системи.

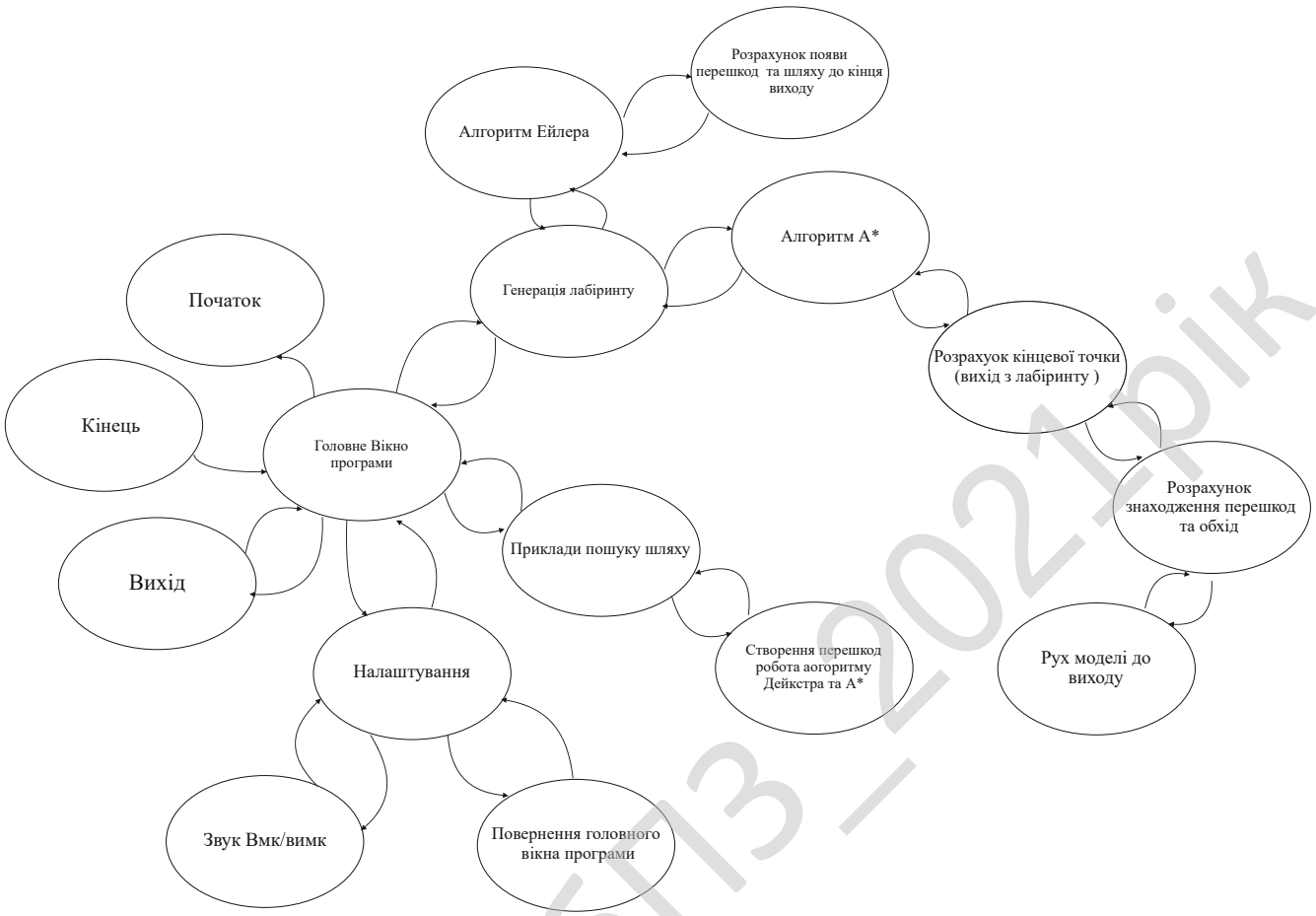


Рисунок 3.4 – Діаграма взаємодії процесів

Як видно з рисунку, діаграма взаємодії процесів серверної частини системи містить наступні процеси:

1. Головне вікно програми.
2. Налаштування.
3. Генерація лабіринту.
4. Алгоритм Ейлера.
5. Розрахунок появи стінок та виходу.
6. Алгоритм А*.
7. Пошук кінцевої точки (вихід з лабіринту).
8. Рух моделі ШІ до виходу.

9. Приклади пошуку шляху

10. Створення перешкод, робота алгоритму Дейкстра та A*

11. Звук вмк/вимк.

13. Повернення до головного вікна.

Для отримання позитивних результатів необхідно вирішити ряд завданнь:

Налаштування конфігурацію лабіринту. Їх завдання вводиться вручну: користувач повинен задати розміри лабіринту по ширині і висоті (від 9 до 27 клітини), непарний розмір є обов'язково;

Меню потребує вибору теми лабіринту (одну з чотирьох); повинен визначити місце розташування входу і виходу з лабіринту, які не повинні збігатися і повинні розташовуватися по периметру лабіринту. При цьому система повинна провести перевірку правильності цієї структури і, в разі невідповідності, видати попереджувальне повідомлення і забезпечити повторне введення параметрів.

Автоматична генерація лабіринту. В системі буде використовуватися два алгоритму генерації лабіринту: обходження перешкод і знаходження «основного шляху».

Робота з файлами. Користувач повинен мати можливість зберегти лабіринт в файл і завантажити його з файлу.

Пошук шляху. Користувач повинен вибрати, за допомогою якого алгоритму буде знайдено шлях.

Існує ряд способів вирішення лабіринтів, кожен зі своїми характеристиками. Ось список конкретних алгоритмів:

Це простіший алгоритм вирішення лабіринту. Він фокусується на користувачі завжди дуже швидкий, і не використовує додаткової пам'яті. Почніть стежити за ходами, і всякий раз, коли ви досягнете перехрестя, завжди поверніть праворуч (або вліво). Еквівалентно людському вирішенню лабіринту, поклавши руку на праву (або ліву) стіну і залишивши її там, коли вони проходять. Якщо вам подобається, ви можете позначити, які клітинки ви відвідали, і які клітинки ви відвідували двічі, де в кінці ви можете відстежити ріше ння, дотримуючись цих

клітинок, відвіданих один раз. Цей метод не обов'язково знайде найкоротше рішення, і він взагалі не працює, коли мета знаходиться в центрі лабіринту, і навколо нього є замкнута схема, оскільки ви обійдете центр і в кінцевому підсумку опинитеся на початку. Наступне за стіною може бути зроблено детермінованим способом в 3D-лабіринті, проектуючи 3D-проходи на 2D площину, наприклад, роблячи вигляд, що проходи насправді ведуть на північний захід і вниз свинцем на південний схід, а потім застосовуючи нормальну стіну за правилами.

Алгоритм застави. Модифікована версія стіни після того, що може стрибати між островами, вирішувати mazes стіни наступні не можуть. Це гарантований спосіб досягти виходу на зовнішній край будь-якого 2D лабіринту з будь-якої точки посередині, однак він не в змозі зробити зворотне, тобто знайти рішення в лабіринті. Це чудово підходить для реалізації роботом-втікаком лабіринту, оскільки він може вибратися з будь-якого лабіринту без необхідності позначати або запам'ятовувати шлях будь-яким чином. Почніть з вибору напрямку, і завжди рухатися в цьому напрямку, коли це можливо. Коли стіна потрапила, почніть стіну, наступний, поки обраний вами напрямок не буде доступний знову. Зверніть увагу, що ви повинні почати стіну, що йде за дальньою стіною, яка потрапила, де, якщо прохід повертає кут там, це може змусити вас розвернутися в середині проходу і повернутися назад, як ви прийшли. Коли стіна слідує, підрахуйте кількість поворотів, які ви робите, наприклад, лівий поворот - - 1, а правий поворот - 1. Зупиніть слідом за стіною і візьміть обраний вами напрямок, коли загальна кількість поворотів, які ви зробили, становить 0, тобто якщо ви повернулися на 360 градусів або більше, тримайте стіну, поки не розвернете себе. Підрахунок гарантує, що ви в кінцевому підсумку зможете досягти зворотного боку острова, на якому ви зараз перебуваєте, і перейти на наступний острів у вибраному напрямку, де ви будете триматися на острові, стрибаючи в цьому напрямку, поки не потрапите в прикордонну стіну, в цей момент стіна після цього приведе вас до виходу. Примітка Алгоритм застави може

змусити вас відвідати уривок або початок більше одного разу, хоча наступний час завжди буде з різними підсумками повороту. Не позначаючи свій шлях, єдиний спосіб дізнатися, чи є лабіринт нерозв'язним, це якщо ваша черга загальна продовжує збільшуватися, хоча загальна черга може дістатися до великої кількості в розв'язуваних лабіринтах у спіральному проході.

Розробка 3D моделей

У склад 3D-моделювання входить процес розробки математичного представлення будь-якої тривимірної поверхні об'єкта за допомогою спеціалізованого ПЗ. Продукт моделювання є 3D-модель. Вона може бути представлена у вигляді програмного коду або відображена у вюпорті чи вювері, як 3D-модель, а також за допомогою двовимірного зображення, що створюється за допомогою процесу рендерингу. 3D-моделі можуть створюватись вручну або автоматично

До розробки персонажів, звичайно, приступають, коли вже відомі основне завдання ролику, цільова група і переваги по стилістиці. За сценарієм розробляємо образ кожного з героїв анімаційного ролику. Іноді окремим етапом ще до промальовування персонажів розробляється опис характеру кожного з персонажів - такий портрет кожного з героїв у текстовій формі. Робиться це для того, щоб при розробці візуального образу уже чітко уявляв собі характер персонажа. На рисунку 3.3 представлено етапи розробки персонажів.

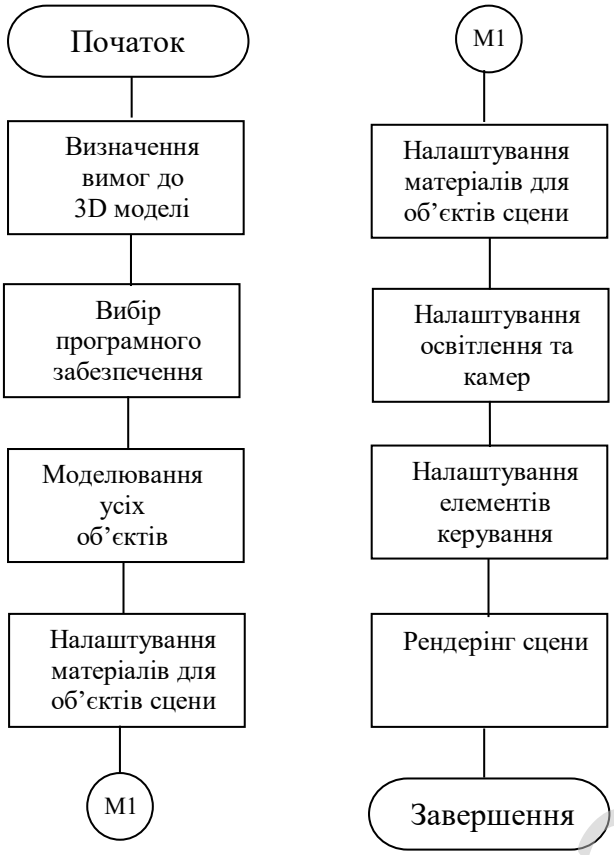


Рисунок 3.5 – Алгоритм створення 3D моделі

На початку, розробляється детальний скетчі, прості замальованих персонажів (3D-ооб’єктів). Створення та редагування моделі персонажів виконується за допомогою програмного середовища Blender3D. Основні функції та переваги Blender3D:

- Підтримка різноманітних геометричних примітивів, включаючи полігональні моделі, систему швидкого моделювання в режимі subdivision surface, криві Без’є, NURBS surfaces, metaballs, видсікання полігонів та векторні шрифти.
- Універсальні вбудовані механізми вимальовування та інтеграція з YafRay.
- Інструменти анімації, серед яких inverse kinematics, арматурна (скелетна) та сіткова деформація, ключові кадри, нелінійна анімація, timeline, vertex weighting, constraints, динаміка м'яких тіл включаючи визначення колізій форми об'єктів при взаємодії, динаміка рідин, Bullet динаміка твердих тіл, система волосся на основі частинок та система частинок при визначенні колізій об'єктів.

- Python використовується як засіб створення інструментів і прототипів, системи логіки в іграх, як засіб імпорту/експорту файлів (наприклад COLLADA), автоматизації завдань.

- Основа системи нелінійного редагування відео та роботи з музикою.

- Game Blender — підпроект Blender, що надає інтерактивні функції, такі як визначення колізій, рушій динаміки та програмована логіка. Також він дозволяє створювати окремі real-time додатки починаючи від архітектурної візуалізації до відео ігор.

Для Створення нової сцени потребує наявність мінімально трьох речей: освітлення, модель та матеріал. Мистецтво створення поверхонь, що візуально схоже на реальний об’єкт, або відповідає представленій цілі є моделювання. Зробити ескіз та план примірного вигляду моделі є первинним етапом. У випадку декількох варіанті, включає велику кількість шарів. Передбачено ефективне та швидке моделювання при використання Blender.

Одна з головних ознак mesh (сітка), поверхня з трьохвимірною структурою (ребра, вершини, грані). Головний базовий елемент для представлення візуального вигляду моделі оточуючим;

NURBS-surface сплайн являє собою поверхню на основі тригонометрично розробленої структури. При вивчені структура виникають складності, погано для розуміння звичайного користувача, (крім практиків математичних аналізів), але достатньо зрозуміла комп'ютеру. Поширюється для створення більш природної форми об’єкту чи ефекту.

Побудова арматури може бути дуже простим і надзвичайно складним, в залежності від того, як сильно аніматор бажає контролювати частини тіла персонажа і як багато обмежень і заборон задумано для рухів персонажа. Наприклад, арматура, в якій коліна можуть згинатися в обох напрямках, буде простіше, ніж та, в якій суглоби забезпечені обмеженнями.

Ступінь необхідної складності арматури визначається в залежності від вимог анімації. Хороша арматура повинна забезпечувати простоту управління персонажем при достовірності рухів.

Процес «ріггінга» (rigging) персонажа означає створення арматури і пов'язаних з нею обмежень. «Скіннінгом» (skinning) називається приєднання сітки до арматури таким чином, щоб маніпуляції з арматурою приводили до відповідних деформацій сітки. Сітка повинна слідувати рухам арматури прийнятним чином. У загальних рисах це досягається шляхом призначення кожної кістки впливу на рух певних вершин. Є кілька способів призначити вплив на вершини, які ми розглянемо в цьому розділі.

Арматура в Blender складається з кісток, їх впливу на кістки, які стоять вище за ієрархією (parent relationships), з'єднань і різних обмежень, які керують їх рухом і взаємодією. Добре зібрану арматуру можна використовувати для швидкого і легкого створення реалістичних поз. Зараз ми коротко ознайомимося з основними типами кісток і обмежень, які використовуються при створенні складних арматур.(кістки)

Система арматури в Blender включає безліч опцій для поведінки і відображення кісток. Залежно від опцій, обраних для кожної кістки, можна визначити кілька базових типів кісток, які використовуються в ріггінге.

Deform bones діє, як опція кісток (за замовчуванням). Арматури деформують кістки дозволяють потрапити на вершини сітки.

B-bones кістки, які мають розділятися на велику розмірність сегментів, що дозволяє їм гнучко рухати та змінювати. B-bones працюють схоже на сплайни чи Безьє. Кожна арматура відіграє роль контрольної точки кривої (curve handle), придатної для створення викривлених поз.

Вони можуть бути корисні для створення гнучких хребтів і викривляє кінцівкі. Їх також можна використовувати для поширення крутного моменту на площу, більшу, ніж просто суглоб, що може бути корисно для кульових (ball-and-socket) з'єднань, таких як плечі, які створюють особливі завдання для ріггінга.

IK solvers використовуються для управління кінцевою точкою ланцюжка інверсної кінематики (IK), в якій положення кісток ланцюга розраховується на основі розташування кінця ланцюга.

Stride bone (крокова кістка) використовується для зв'язування руху арматури вздовж шляху з рухами кісток в арматурі. Вона використовується для того, щоб гарантувати, що опускання ніг на землю буде відбуватися вчасно, при русі арматури вперед в ході циклу ходьби.

Ipo drivers. Кістки можуть використовуватися для управління будь-яким аніміруемим властивістю, шляхом створення драйвера інтерполяції (Ipo), який зв'язує положення або обертання кістки з відповідною Ipo.

Custom controls. Можливі налаштування управління (custom controls) є не стільки типом кісток, скільки особливим способом налаштування зовнішнього вигляду кісток. У вас можуть бути кістки, з'являються в режимі Pose у вигляді спеціально призначених сіток, які можуть бути корисні для візуального відділення кісток спеціального призначення (таких як драйвера Ipo) від звичайних кісток. Для відображення і редагування кісток існує безліч опцій.

Обмежувачі (constraints) служать одночасно і керуючими елементами, що роблять рух об'єкта, або кістки залежними від іншого об'єкта, або певного моменту. Створення арматури з добре продуманими обмежувачами допоможе спростити настройку поз, в порівнянні з управлінням рухом кожної кістки вручну. Найважливіші обмежувачі:

IK Constraint працює з IK solver для створення ланцюга кісток, чия позиція може бути визначена по розташуванню кінця ланцюга.

Copy Location / Rotation / Scale обмежує переміщення, обертання або масштабування однієї кістки або об'єкта по відношенню до іншого. Обмежує сильніше, ніж парентінг (коли один об'єкт є предком другого), де обмежений об'єкт або кістка не може бути переміщений, повернутий або масштабований незалежно від іншого.

Track-to Constraint. Змушує об'єкт або кістка вказувати в напрямку («стежити» - «track to») На рисунку 3.4 представлено процес накладаня арматури та створення самого скелету персонжа (рік)

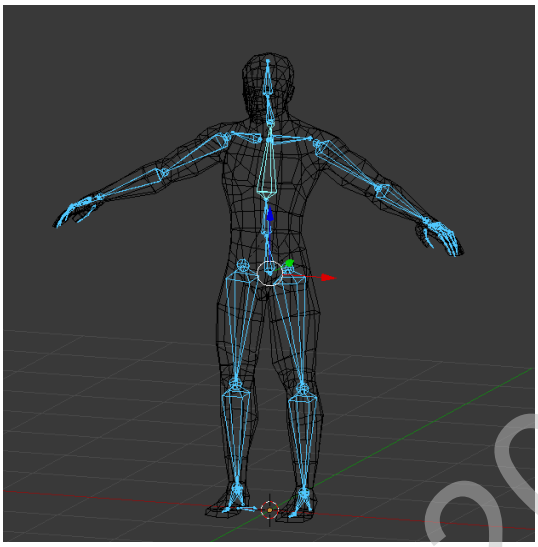


Рисунок 3.6 - Процес накладання арматури та створення самого скелету персонажа (рік)

Для анімації персонажів треба організувати робоче середовища, помістивши модель, об'єкти на різні п оля робочої області. Можна заплутатись при використані одночасного показу багатьох об’єктів. За допомогою інструменту шар є можливість приховати не потрібні на етапі проектування об'єкти, та при потребі роботи їх видимими. Blender надає двадцять шарів для допомоги в організації основної роботи. Користувач можете бачити, які з шарів поточно видимі серед групи двадцяти кнопок у заголовку 3D-вікна (дивіться ілюстрацію контролери видимості шарів). Ви можете змінювати видимість шару для більш легкої роботи з моделю.

До створення 3D зображень можна виділити:

- 3D моделювання окремих елементів;
- Проектування навігації, розробка об'ємних елементів;
- Розробка зображення об'ємних логотипів;

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

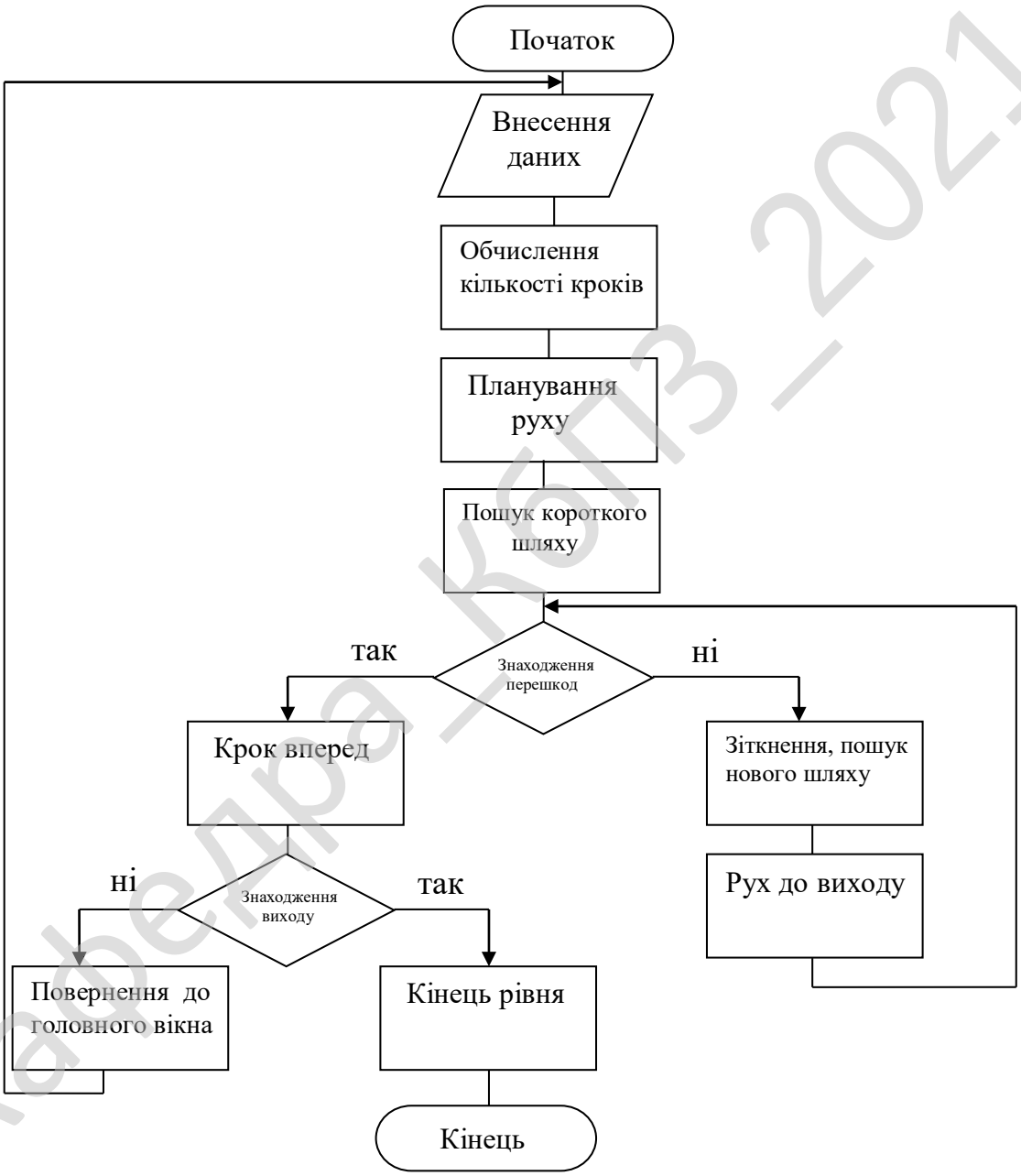


Рисунок 4.1 – Блок схема алгоритму функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1. З рисунку видно, що після запуску програми спочатку відбувається вивід вікна програми. Потім ШІ виконує наступні дії:

1. Пошук виходу.
2. Внесення даних.
3. Обчислення кількості корків.
4. Планування руху.
5. Пошук руху.
6. Пошук короткого шляху.
7. Знаходження перешкод.
8. Обхід перешкод.
9. Знаходження виходу.
10. Повернення до головного вікна.

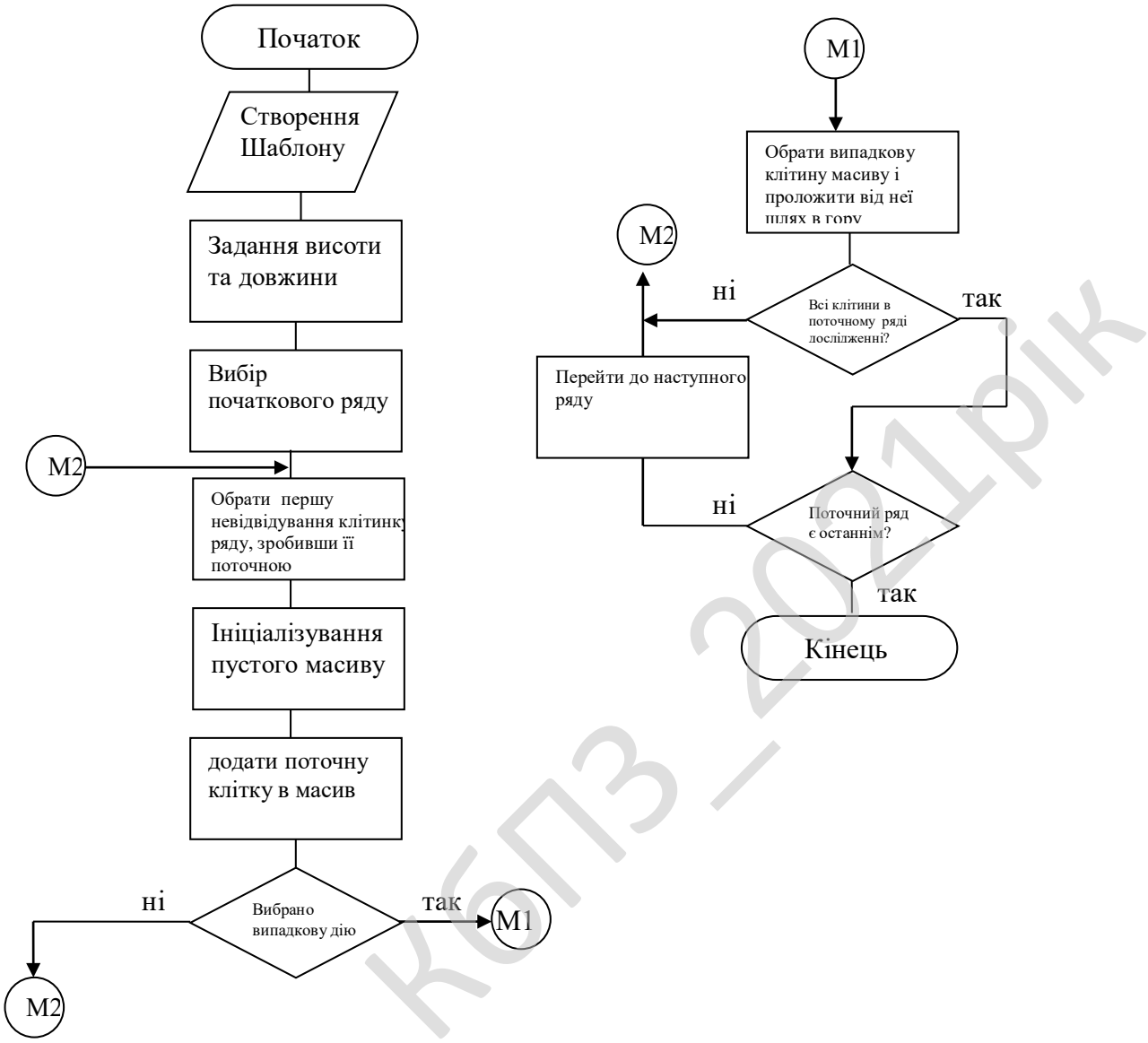


Рисунок 4.2 – Блок схема алгоритму генерації лабіринту

Важливим моментом алгоритму є очищення першого ряду від стінок. Таким чином початковим поруч вважається ряд, наступний за самим першим. На графічно представлений розбір алгоритму на прикладі поля. Поле відображає процес очищення першого ряду від стінок. На полях представлено обрану дію «додати в масив ще одну клітку ряду». Проводиться вибір випадкової клітини з масиву, після чого всі стінки між клітинами масиву, а також верхня стінка випадково обраної клітини знищуються. Представлена ініціалізація масиву,

додавання першої клітини в масив, після чого перехід до дії «вибрати з м асиву одну з кліток і з'єднання того масива клітин з верхнім рядом через обрану клітку». У даній ситуації руйнується тільки верхня стінка над поточною клітиною

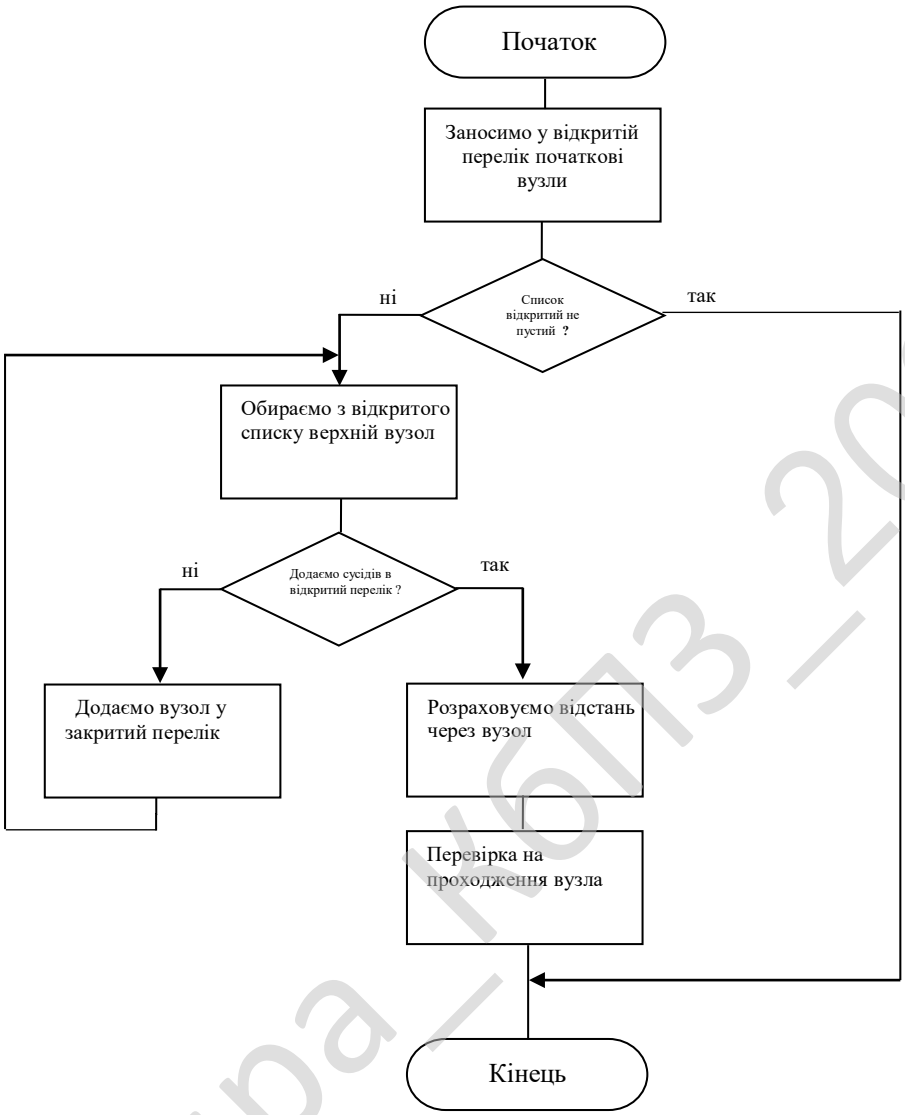


Рисунок 4.3 - Блок схема алгоритму A*

Покроковий опис алгоритму, граф збіг алгоритм лабіринт

1. Поміщаємо у відкритий список стартовий вузол.

2. Основний цикл (поки відкритий список не порожній).

3. Беремо із відкритого списку верхній вузол.

4. Якщо цей вузол дорівнює кінцевому, будуємо шлях і виходимо з основного циклу.

5. Обробляємо ланки вибраного вузла. Для кожного сусіда:

- перевіряємо його на прохідність, для конкретного юніту, якщо непрохідний, то, ясна річ, переходимо до наступного сусіда;

- обчислюємо оцінку шляху від стартового вузла через поточний сусіда;

- шукаємо сусіда у відкритому та закритому списках, якщо знайдено, то порівнюємо його оцінку шляху від стартового вузла з обчисленою, якщо його оцінка краще то йдемо до наступного сусіда, інакше видаляємо вузол з відповідного списку, перераховуємо оцінку шляху до кінцевого вузла та поміщаємо цього сусіда в Open список, у відповідне місце;

6. Поміщаємо поточний вузол до закритого списку.

7. Переходимо до п.3 (кінець основного циклу).

4.2 Захист розробленого програмного забезпечення

У вас є можливість повернути свій контент назад. Наприклад, є інструменти, що дозволяють зберігати тривимірні дані на рівні користувачів. Можна створити свій власний спосіб шифрування даних для AssetBundle файлів.

Одним із способів досягнення цього, є використання TextAsset типу для зберігання ваших даних в байтах. Можна зашифрувати потрібні вам файли, зберігши їх з розширенням .bytes, яке Unity буде сприймати як TextAsset тип. Після імпортування в редактор, дані файли будуть включені в ваші AssetBundles як TextAssets і розміщені на сервері. Потім на стороні клієнта буде викачаний потрібний AssetBundle, з TextAsset типів файлів якого будуть вилучені необхідні клієнту дані. При такому підході будуть шифруватися не власними AssetBundles, а лише TextAsset файли, що знаходяться в них.

```
string url = "http://www.mywebsite.com/mygame/assetbundles/assetbundle1.unity3d";
IEnumerator Start () {
    while (!Caching.ready)
        yield return null;

    // Start a download of the encrypted assetbundle
    WWW www = new WWW.LoadFromCacheOrDownload (url, 1);

    // Wait for download to complete
```

```
    yield return www;

    // Load the TextAsset from the AssetBundle
    TextAsset textAsset = www.assetBundle.Load("EncryptedData", typeof(TextAsset));

    // Get the byte data
    byte[] encryptedData = textAsset.bytes;

    // Decrypt the AssetBundle data
    byte[] decryptedData = YourDecryptionMethod(encryptedData);

    // Use your byte array. The AssetBundle will be cached
}
```

В якості альтернативи можна повністю зашифрувати AssetBundles в сховище і потім завантажити його шляхом використання WWW класу. До тих пір, поки ваші файли зберігаються на сервері в якості бінарних даних, вони можуть мати будь-яке розширення. Як тільки потрібний AssetBundle буде викачаний, вам потрібно буде декодувати дані, використовуючи властивість .bytes вашого WWW примірника, щоб отримати розшифровані дані з AssetBundle і відтворити з пам'яті AssetBundle, використовуючи AssetBundle.CreateFromMemory.

```
string url = "http://www.mywebsite.com/mygame/assetbundles/assetbundle1.unity3d";
       IEnumerator Start () {
           // Start a download of the encrypted assetbundle
           WWW www = new WWW (url);

           // Wait for download to complete
           yield return www;

           // Get the byte data
           byte[] encryptedData = www.bytes;

           // Decrypt the AssetBundle data
           byte[] decryptedData = YourDecryptionMethod(encryptedData);

           // Create an AssetBundle from the bytes array

           AssetBundleCreateRequest
       acr = AssetBundle.CreateFromMemory(decryptedData);
           yield return acr;

           AssetBundle bundle = acr.assetBundle;

           // You can now use your AssetBundle. The AssetBundle is not cached.
       }
```

Перевага другого підходу в порівнянні з першим в тому, що ви можете використовувати будь-який метод для передачі своїх байтів і даних в якості (крім AssetBundles.LoadFromCacheOrDownload) закодованих - наприклад сокети в плагіні. Мінусом тут буде те, що дані не будуть кешироватися при використанні

автоматичного кешування Unity. Можна додати будь-які програвачі, крім WebPlayer-a, який повинен бути завантажений вручну з вашого жорсткого диска з використанням AssetBundles.CreateFromFile.

Третій спосіб буде кращий з двох інших, шляхом зберігання самого AssetBundle як TextAsset в інших, нормальних AssetBundles. Чи не закодований AssetBundle, що містить в собі закодований буде кешироватися. Оригінальний AssetBundle може бути потім з авантажений в пам'ять, стверджує і інстанціює, виконує AssetBundle.CreateFromMemory.

```
string url = "http://www.mywebsite.com/mygame/assetbundles/assetbundle1.unity3d";
       IEnumerator Start () {
           while (!Caching.ready)
               yield return null;

           // Start a download of the encrypted assetbundle
           WWW www = new WWW.LoadFromCacheOrDownload (url, 1);

           // Wait for download to complete
           yield return www;

           // Load the TextAsset from the AssetBundle
           TextAsset     textAsset     =     www.assetBundle.Load("EncryptedData",
typeof(TextAsset));

           // Get the byte data
           byte[] encryptedData = textAsset.bytes;

           // Decrypt the AssetBundle data
           byte[] decryptedData = YourDecryptionMethod(encryptedData);

           // Create an AssetBundle from the bytes array
           AssetBundleCreateRequest
       acr = AssetBundle.CreateFromMemory(decryptedData);
           yield return acr;

           AssetBundle bundle = acr.assetBundle;

           // You can now use your AssetBundle. The wrapper AssetBundle is cached
       }
```

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програмне забезпечення призначене для створення, генерація лабіринтів та проходження. Для запуску ПЗ існує файл .exe в папці який був розпакований. Після запуску користувач бачить головний інтерфейс меню ПЗ

Інтерфейс розробленого програмного забезпечення зображено на рисунках 5.1-5.3. З рисунків видно, що головне вікно програми містить наступні вкладки:

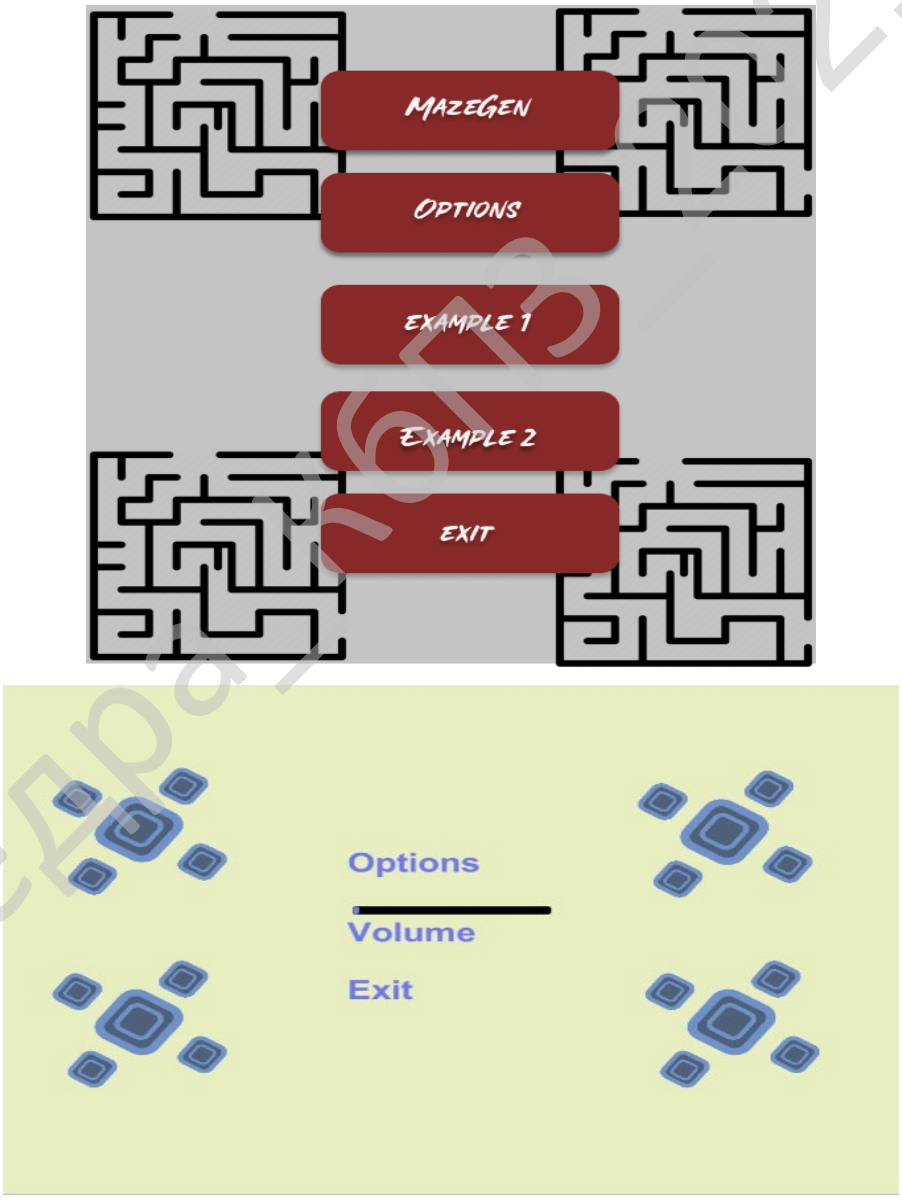


Рисунок 5.1 - Головне меню програми та меню опцій

Для початку роботи треба натиснути певні елементи (кнопки) інтерфейсу.

Користувач натиснувши кнопку MazeGen має можливість переглянути генерацію лабіринтів, побачити траєкторію, яка показує шлях до виходу (рисунок 5.2). Кнопка Example дозволяє побачити приклади пошуку шляху, та створити власні перешкоди для ШІ (рисунок 5.3). Перехід до меню Options надає можливість змінити гучність в грі . Exit – Вихід з головного меню до робочого стола.

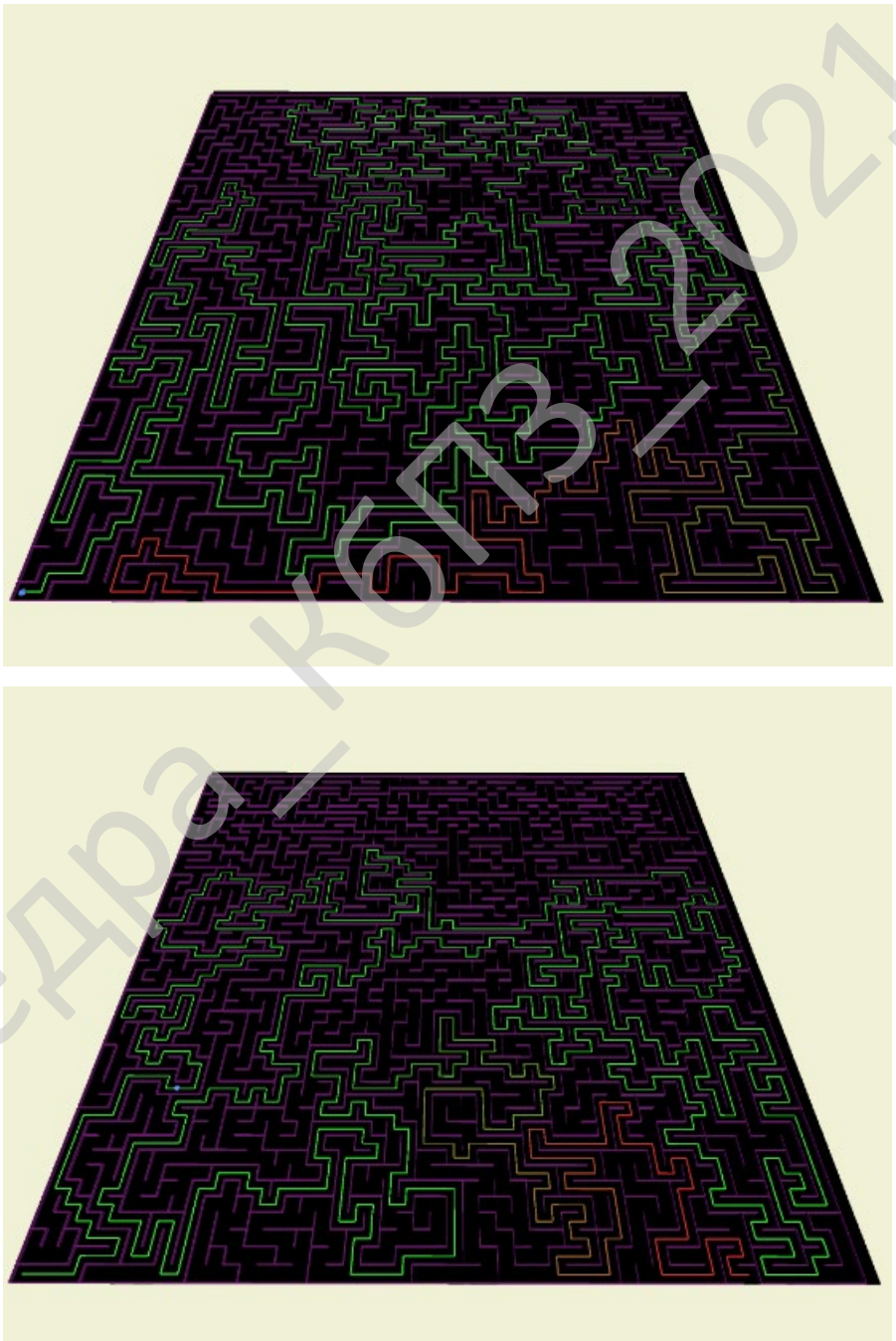


Рисунок 5.2 – Вікно сцени Генерації лабіринту та пошук виходу

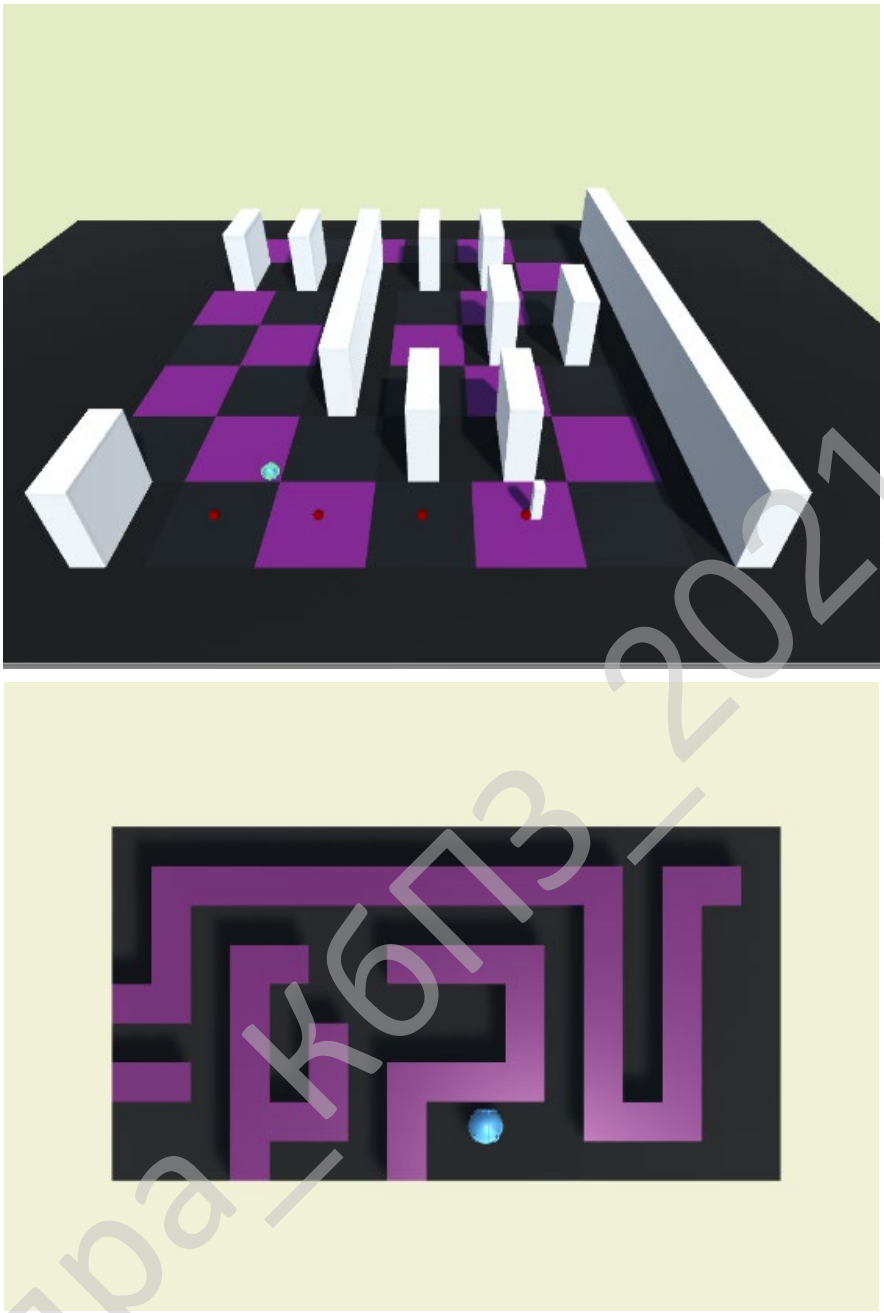


Рисунок 5.3 – Вікно прикладу роботи алгоритму пошуку шляхів

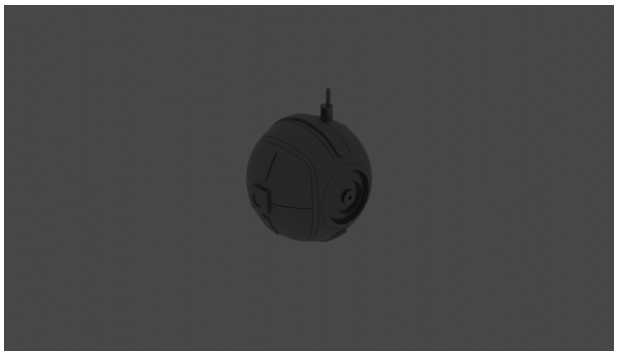


Рисунок 5.4 – 3Д Модель персонажу

6 НАУКОВА НОВИЗНА

Метою даної роботи є дослідження та розробка методів, алгоритмів та програмного забезпечення системи генерації та проходження лабіринтів для розробки відеоігор.

Програмне забезпечення системи генерації та проходження лабіринтів для розробки відеоігор, є актуальною задачею, яка потребує вирішення.

Під час реалізації технічного завдання даної магістерської роботи було проведено дослідження областей машинного навчання ШІ та генерації простору, сучасні рішення проблем розробки в цих напрямах.

Об'єктом дослідження є процеси генерації лабіринтів та пошуку руху у них.

Предметом дослідження є методи генерації лабіринтів та пошуку шляху у них за допомогою компонентів та інтерфейсів системи ігрового рушія Unity та їх оптимізація для роботи на ПК.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії ймовірності та теорії штучного інтелекту.

Наукова новизна отриманих результатів. Згідно обумовлених цілей дослідження у процесі рішення завдань, отримані такі результати:

- Удосконалено методи генерації та проходження лабіринтів на основі штучного інтелекту для застосування у комп'ютерних іграх;

- Розроблено вітчизняний продукт для генерації та проходження лабіринтів у комп'ютерних іграх на основі штучного інтелекту, який є більш легким у використанні, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає у тому, що розроблене програмне забезпечення дозволяє генерувати зовнішнє середовище у комп'ютерних іграх та керувати рухом ігрових об'єктів для пересування по ньому. Розроблені алгоритми можна використовувати в ігровому 3D та 2D рушії.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи.

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці).

В магістерській роботі проведене дослідження та виконана програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор.

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

а) невеликий розмір;

б) невеликі системні потреби;

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

г) зручність у користуванні та надійність

Таблиця 7.1 - Початкові данні

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	84
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Г
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	5

Продовження таблиці 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	_	3
13. Рівень спрацьованості колективу (1-6)	_	3
14. Ступінь вимірності процесів (1-6)	_	2
15. Необхідна надійність програмного забезпечення (1-6)	_	3
16. Розмір бази даних (порівняно з розміром програми) (1-6)	_	1
17. Складність кінцевого програмного продукту (1-6)	_	2
18. Необхідний рівень забезпечення повторного використання (1-6)	_	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	_	3
20. Вимоги до швидкодії ПП (1-6)	_	3
21. Обмеження на розміри основного сховища даних (1-6)	_	1
22. Різноманітність використовуваних обчислювальних платформ (1-6)	_	2
23. Професійний рівень аналітиків (1-6)	_	3
24. Професійний рівень програмістів (1-6)	_	3
25. Постійність складу команди розробників (1-6)	_	3
26. Досвід розробки додатків (1-6)	_	3
27. Досвід роботи з обчислювальною платформою (1-6)	_	4

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	4
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	1
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	3
32. Вартість ПЗ у розробника (НМА), грн	–	84000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22%
35. Норматив загальногосподарських витрат, %	Нг	10
36. Норматив витрат на освоєння нових мов програмування, %	Нп	10
37. Рівень рентабельності програмної продукції, %	Ре	60
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

де А - коефіцієнт Боема, А=2,45;

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

B - показник ступеня, що визначається співвідношенням

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

де Wi - сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+2{,}53+2{,}97+3{,}64) = 1{,}162$$

$$T_{ном} = 2{,}45 \cdot 2{,}0^{1{,}162} = 5{,}48 \text{ люд-міс.}$$

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

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

де $ПВ$j - добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 5{,}48\cdot(1\cdot 0{,}93\cdot 0{,}88\cdot 0{,}91\cdot 1\cdot 1\cdot 1\cdot 0{,}87\cdot 1\cdot 1\cdot 1\cdot 1\cdot$$
$$\cdot 0{,}88\cdot 0{,}91\cdot 1\cdot 1{,}25\cdot 1)=3{,}55 \text{ люд-міс.}$$

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

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

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

де C - визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S - коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot 2{,}75\cdot 3{,}55^{0{,}33+0{,}2(1{,}162-1{,}01)}\cdot 150=195 \text{ люд/день}$$

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

Таблиця 7.2 - Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	4	Д5
Ескізний проект	5	Д6
Технічний проект	6	Д7
Робочий проект	195	Ф 7.1-7.4
Впровадження	13	Д13
Всього	223	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

де $F_{рq}$ - плановий фонд робочого часу одного спеціаліста, днів,

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

$$Ч = \frac{223 \cdot 1}{48 - 5} = 5 \text{ ставок}$$

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

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

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

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість обладнання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	385	1	385	6
Монітор	160	1	160	3
Клавіатура	140	1	140	2
Маніпулятор «мишка»	30	1	30	0,5
Принтер лазерний	355	1	355	6
Сканер	155	1	155	3
3D принтер Creality Ender 3	355	1	355	6
Усього за рік:			$З_ч$	26,5

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{mic}}{1,2} \tag{7.6}$$

$$Ф_{др}^{с} = \frac{26,5 \cdot 2}{1,2} = 44 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}} \tag{7.7}$$

$Ч_{ел}$=44/(48·5)=0,18 ставки

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

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

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

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка	1	
	Оформлення договорів	0,5	
	Контроль взаєморозрахунків з замовником	0,5	
Всього		4	
Дизайнер UI інтерфейс	Розробка концепції оформлення та інтерфейсу додатку, оптимізація дизайну існуючих, проектує їх структуру та навігацію	0,5	0,2
	Створення графічних і стилістичних елементів додатку	0,5	
	Оформлення промо-ролику	0,3	
	Розміщення графіки і контенту в документації	0,3	
Всього		1,6	
Аналітик ефективності алгоритмів	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,2
	Верстка друкованих видань	0,2	
	підготовка пробних макетів	0,2	
	Розміщення об'єктів на сцені	0,2	
Всього		1,6	

Складемо штатний розклад виконавців у таблицю 7.5.

Таблиця 7.5 - Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (IT-менеджер)	1	14000	28000
Продакт-менеджер	0,5	12000	12000
Інженер-програміст	5	11000	55000
Інженер по техпідтримці	0.2	9000	3600
Адміністратор проекту	0,2	10000	4000
Системний програміст	0,2	9000	3600
Дизайнер UI інтерфейсу	0,2	9000	3600
Аналітик ефективності алгоритмів	0,2	9000	3600
Бухгалтер-економіст	0,2	10000	4000
Всього за період розробки	$R_{сп}$=7,5	-	$Ф_{роб}$=117400

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{117400}{7,5 \cdot 48} = 360 \text{ грн}$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

$$Б_{уд} = R_{сп}^{1} S_{у} Ц_{пл}, \quad (7.9)$$

де $R_{сп}^{1}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць.

$S_{у}$ – питома площа на одне робоче місце, $м^2$,

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

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 800…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 27 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$I_{нв} = R_{сп}^{1} \cdot Ц_{м}, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались за комерційною пропозицією фірми Brain за 05.11.20 – джерело http://brain.com.ua/

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		27817
Системний блок		7771
Процесор	Intel Core i3-9100 3.6GHz/8GT/s/6MB	4 699
Системна плата	ASRock Intel s1151 B365 4DDR4 2PCIex16 2M.2 TypeC mATX (B365M PRO4-F)	1985
Жорсткий диск	Western Digital Blue 1TB 7200rpm 64MB WD10EZEX 3.5 SATA III	1290
Оперативна пам’ять	DDR4 16GB (2X8GB) 3200 MHZ FURY BEAST RGB KINGSTON FURY (EX.HYPERX) (KF432C16BBAK2/16)	2 129
відеокарта	GIGABYTE GeForce GT1030 2GB DDR4	7132
Корпус	Logicpower 8702 - 550w 12cm	1411
Кардрідер внутрішній	Transcend TS-RDF8K USB 3.0	220
інше	Клавіатура, мишка	додатково
Монітор	Монітор BenQ GL2450HM Black	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Сканер	Epson Perfection V37	2800
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
3d прінтер	Creality Ender 3	7000
Пристрій безперебійного живлення	Powercom BNT-600AP USB	1400

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 - Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	5	27817	9416,8	186169
Принтер лаз.	1	2700	540	3240
3d прінтер	1	7000	900	7900
Сканери	1	2800	280	3080
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	206950,5

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	206950,5	-	-
Всього по групі	206950,5	50	103475,25

Продовження таблиці 7.8

Група 5,6			
4. Вимірювальні пристрої	5190	-	-
5. Транспортні засоби	143000	-	
6. Господарський інвентар	28000	-	-
Всього по групі	176190	20	35238
7. Нематеріальні активи	84000	10	8400
Разом	$К_р$= 1875140,5		$А_р$= 217513,25

Примітка: вартість автомобіля Standart 4-дв.седан / бензин / 1,6 л 8 кл . взята по даним з автосалону «Кіровоград-Авто», джерело https://kirovograd-avto.ukravto.ua/cars/granta, складає 235440 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

де $N_е$ – Кількість екземплярів програм, шт.

$$З_о = (360 \cdot 223)/84 = 955,71 \text{ грн.}$$

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

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

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 955,71 \cdot 10 \cdot 0,01 = 95,57 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

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

де H_c – відрахування на соціальні потреби, %

$$C_{оц} = 0{,}01 \cdot 22(955{,}71+95{,}57) = 231{,}28 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом *Нг*=15% від основної зарплати

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

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 955{,}71 \cdot 10 \cdot 0{,}01 = 95{,}57 \text{ грн}$$

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

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

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картріджей, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм приймаємо одну пачку паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=145 грн., визначаємо вартість паперу за період розробки $N_м$=2 міс:

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

$$З_{М1} = 145 \cdot 2 = 290 \text{ грн.}$$

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

$$З_{М2} = \sum Ц_д., \quad (7.17)$$

де $Ц_д$ – вартість дисків DVD-R LG 4,7Gb, 16x speed Cake box – 6,62 грн/шт.

$$З_{М2} = 84 \cdot 8 + 10 = 682 \text{ грн.}$$

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

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; заправка 3d принтера – 1000 грн.

$З_{МЗ}$ =574+1000=1574 грн.

$З_М$ = (145 + 682 + 1574)/84=27 грн.

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 10%) від основної зарплати виконавців

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \tag{7.19}$$

де $Н_п$ - норматив витрат на освоєння нових мов програмування, %

$О_п$ =955,71 10·0,01=95,57 грн.

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 84 прим.)

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

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

$А_м$ =217513,25·2/(84·12)=431,57 грн.

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

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

$С_п$ =955,71+95,57+231,28+95,57+27+95,57+431,57=1932,27 грн.

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

Для даного програмного забезпечення рівень рентабельності складає 50%

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \tag{7.22}$$

де $Р_с$ – рівень рентабельності, %

$П_р$ = 0,01·50·1932,27 = 966,13 грн.

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	955,71
2. Додаткова зарплата виконавців	$З_д$	95,57
3. Відрахування на соціальні потреби	$С_{оц}$	231,28
4. Загальногосподарські витрати	$Г_{осп}$	95,57
5. Витрати на матеріали	$З_М$	27
6. Освоєння нових операційних систем, мов програмування	$О_п$	95,57
7. Амортизація основних фондів	$А_м$	431,57
8. Повна собівартість програмного забезпечення	$С_п$	1932,27
9. Плановий прибуток	$П_р$	966,13
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	2898,4
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	579,68
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	3478

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних

робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 - Розрахунок об'єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	3478
Всього капітальних витрат	–	3478

7.7 Визначення експлуатаційних витрат

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

Таблиця 7.11 - Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	29524	16104
2. Витрати на електроенергію	$З_{ел}$	0	0
Витрати на амортизацію	$З_{ам}$	0	1739
Всього витрат за рік	$І$	29524	21075

Витрати на технічне обслуговування:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де $Т_р$ – кількість годин обслуговування системи за рік, *год.*,

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 220 годин на рік до 120 годин на рік, тому витрати на технічне обслуговування зменшилися з

$$З_{р\,баз} = 220 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 29524\ грн.$$

до

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

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

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

Зел баз= Зел нов

Витрати на електроенергію не змінюються.

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

Таблиця 7.12 - Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	3478	–	1739
Всього відрахувань	-	–	3478	–	1739

7.8 Визначення економічної ефективності програмної продукції

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

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

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

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

$$Е_в = (2898{,}4 - 1932{,}27) \cdot 84 - (0{,}05 \cdot 1875140{,}5 + 0{,}5 \cdot 206950{,}5$$
$$+0{,}2 \cdot 176190 + 0{,}1 \cdot 84000) \cdot 2 \div 12 = 26619$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $К_р$– балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи.

$$Т_в \frac{1875140{,}5}{(2898{,}4-1932{,}27) \cdot 84 \cdot 12 \div 2} = 0{,}26 \text{ років}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (29524\text{-}21075)\text{-}0{,}5 \cdot 3478 = 6710 \text{ грн.}$$

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

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

$$Т_{сп} = \frac{3478}{29524-21075} = 2{,}4 \text{ року}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 - Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	84
2. Повна собівартість розробленої програми	Грн.	1932,27
3. Ціна розробленої програми	Грн.	2898,4
4. Плановий прибуток від реалізації розробленої програми	Грн.	966,13
5. Рентабельність програмної продукції	%	60
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1875140,5
7. Загальний прибуток від реалізації програмної продукції	Грн.	81155
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	142172
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,26
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	3478
11.Величина економічного ефекту у користувача програмної продукції	Грн.	6710
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	2,4

7.9 Висновки

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

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

В основу охорони праці включають санітарно-гігієнічні, лікувально-профілактичні та організаційно-технічні системи правових і соціально-економічних заходів.

В кожній ІТ компанії є трудові відносини з працівниками. Згідно закону України “Про охорону праці” кожна компанія впроваджує заходи з охорони праці [8]. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

- Інструкцій з охорони праці по кожній професії і загальні;
- Положення про охорону праці;
- Накази з охорони праці;
- Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Існує інструкція з охорони праці фахівців з інформаційної технології. Є нормативний документ, у якому міститься вимоги охорони праці для ІТ фахівця при виконання робіт, які визначені його функціональними обов’язками. Ця інструкція розроблена згідно з Положенням про розробку інструкцій з охорони праці, затвердженого наказу Держнаглядохоронпраці від 29.01.1998 р. №9; вимог що до безпеки та захисту здоров’я працівників підчас роботи з екранними

пристроями, затверджених наказом міністерства соціальної політики України від 14.02.2018 р. №207; Державних санітарних правил і норм роботи з візуальними, дисплейними, терміналами, електроно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.1998р. №7 [52].

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Наявний в даний час в нашій країні комплекс розроблених організаційних заходів та технічних засобів захисту, накопичений передовий досвід роботи ряду обчислювальних центрів показує, що є можливість домогтися значно більших успіхів у справі усунення впливу на працюючих небезпечних і шкідливих виробничих факторів. Проте стан умов праці та його безпеки в ряді IT компаній ще не задовольняють сучасним вимогам. Оператори ЕОМ, оператори по підготовці даних, програмісти та інші працівники IT стикаються з впливом таких фізично небезпечних і шкідливих виробничих факторів, як підвищений рівень шуму, підвищена температура в приміщенні, відсутність або недостатня освітленість робочої зони, електричний струм, статична електрика і інші [52].

Всі працівники пов'язані з впливом таких психофізичних факторів, як розумова перенапруга, перенапруження зорових і слухових аналізаторів, монотонність праці, емоційні перевантаження. Вплив зазначених несприятливих факторів призводить до зниження працездатності, викликає розвиток втоми. Поява і розвиток втоми пов'язане зі змінами, які виникають під час роботи в центральній нервовій системі, з гальмівними процесами в корі головного мозку. Наприклад сильний шум викликає труднощі з розпізнаванням колірних сигналів, знижує швидкість сприйняття кольору, гостроту зору, зорову адаптацію, порушує сприйняття візуальної інформації, зменшує на 5 - 12% продуктивність праці. Тривала дія шуму з рівнем звукового тиску 90 ДБ знижує продуктивність праці на 30 - 60% [3].

Медичні обстеження працівників показали, що крім зниження продуктивності праці високі рівні шуму призводять до погіршення слуху. Тривале перебування людини в зоні комбінованого впливу різних несприятливих факторів може призвести до професійного захворювання. Аналіз травматизму серед працівників показує, що в основному нещасні випадки відбуваються від впливу фізично небезпечних виробничих факторів при заправці носія інформації на обертовий барабан при знятому кожусі, під час співробітниками невластивих їм робіт. На другому місці випадки, пов'язані з дією електричного струму [52].

8.3 Забезпечення електробезпеки

Електричні установки, до яких відноситься практично все обладнання в IT компаніях, представляють для людини велику потенційну небезпеку, так як в процесі експлуатації , або проведенні профілактичних робіт людина може торкнутися частин, що знаходяться під напругою. Специфічна небезпека електроустановок є струмоведучі провідники, корпуси стійок ЕОМ і іншого устаткування, яка під напругою в результаті пошкодження (пробою) ізоляції, не подає будь-яких сигналів, які попереджають людину про небезпеку. Реакція людини на електричний струм виникає лише при протіканні останнього через тіло людини. Виключно важливе значення для запобігання електротравматизма має правильна організація обслуговування діючих комп’ютерів, проведення ремонтних, монтажних і профілактичних робіт. При цьому під правильною організацією розуміється строге виконання ряду організаційних технічних заходів і засобів, встановлених діючими "Правилами технічної експлуатації електроустановок споживачів і правил техніки безпеки при експлуатації електроустановок споживачів" (ПТЕ і ПТБ споживачів) і "Правила установки електроустановок" (ПУЕ) Залежно від категорії приміщення необхідно вжити певних заходів, що забезпечують достатню електробезпеку при експлуатації та ремонті електроустаткування [4]. В приміщеннях з підвищеною небезпекою

електроінструменти, переносні світильники повинні бути виконані з подвійною ізоляцією, або напруга живлення їх не повинна перевищувати. До таких приміщень можуть бути віднесені приміщення машинного залу, приміщення для розміщення сервісної та периферійної апаратури. В особливо небезпечних приміщеннях напруга живлення переносних світильників не повинна перевищувати 12В, а робота з електротранспортуємою напругою не вище 42В дозволяється тільки із застосуванням (діелектричних рукавичок, килимків і т.п.). Роботи без зняття напруги на струповидних частинах і поблизу них, роботи проводяться безпосередньо на цих частинах або при наближенні до них на відстань менше встановленого ПЕУ. До цих робіт можна віднести роботи з налагодження окремих вузлів, блоків. При виконанні такого роду робіт в електроустановках до 1000 В необхідне застосування певних технічних і організаційних заходів, таких як: огорожі, розташовані поблизу робочого місця та інших струмоведучих частин, до яких можливо випадковий дотик; робота в діелектричних рукавичках або стоячи на діелектричному килимку; застосування інструменту з ізолюючими рукоятками, за відсутності такого інструменту слід користуватися діелектричними рукавичками. Роботи цього виду повинні виконуватися не менше ніж двома працівниками [52].

Відповідно до ПТЕ і ПТВ споживачам і обслуговуючому персоналу електроустановок пред'являються наступні вимоги:

-особи, які не досягли 18-річного віку, не можуть бути допущені до робіт в електроустановках;

-особи не повинні мати каліцтв і хвороб, що заважають виробничій роботі;

-особи повинні після відповідної теоретичної та практичної підготовки пройти перевірку знань і мати посвідчення на доступ до робіт в електроустановках.

Розрядні струми статичної електрики частіше за все виникають при дотику до будь-якого з елементів ЕОМ. Такі розряди небезпеки для людини не представляють, але крім неприємних відчуттів вони можуть привести до виходу з ладу ЕОМ. Для зниження величини виникаючих зарядів статичної електрики в ОЦ покриття технологічних статей повинно бути з одношарового полівінілхлоридного

антистатичного лінолеуму. Іншим методом захисту є нейтралізація заряду статичної електрики іонізованим газом. У промисловості широко застосовуються радіоактивні нітралізатори. До загальних заходів захисту від статичної електрики в ОЦ можна віднести загальні та місцеве зволоження повітря [52].

8.4 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Приміщення та їх розміри (площа, об'єм) повинні в першу чергу відповідати кількості працюючих і розміщених в них комплекту технічних засобів. У них передбачаються відповідні параметри температури, освітлення, чистоти повітря, забезпечують ізоляцію від виробничих шумів і т.п. Для забезпечення нормальних умов праці санітарні норми встановлюють на одного працюючого обсяг виробничого приміщення не менше 15 м3, площа приміщення огороджена стінами або глухими перегородками не менше 4,5 м3.

Для експлуатації ЕОМ слід передбачати такі приміщення:

- машинний зал,
- приміщення для зберігання запасних деталей, інструментів, приладів ;
- приміщення для розміщення припливно-витяжних вентиляторів;
- приміщення для персоналу;
- приміщення для прийому-видачі інформації.

Основні приміщення робочих комп’ютерних станцій розташовуються в безпосередній близькості один від одного. Їх обладнають загальною вентиляцією і штучним освітленням. До приміщення машинного залу і зберігання магнітних носіїв інформації пред'являються особливі вимоги. Площа машинного залу повинна відповідати площі необхідної за заводськими технічним умовам даного типу ЕОМ.

У тих випадках, коли одного природного освітлення не вистачає, встановлюється суміщене освітлення. При цьому додаткове штучне освітлення застосовується не тільки в темний, але і в світлий час доби.

Штучне освітлення по характеру виконуваних завдань ділиться на робоче, аварійне, евакуаційне [51].

Раціональне колірне оформлення приміщення направлено на поліпшення санітарно-гігієнічних умов праці, підвищення його продуктивності і безпеки. Забарвлення приміщень, впливає на нервову систему людини, його настрій і в кінцевому рахунку на продуктивність праці. Основні виробничі приміщення доцільно фарбувати відповідно до кольору технічних засобів. Освітлення приміщення і устаткування повинне бути м'яким, без блиску.

Зниження шуму, створюваного на робочих місцях внутрішніми джерелами, а також шуму проникаючого ззовні, є дуже важливим завданням. Зниження шуму в джерелі випромінювання можна забезпечити застосуванням пружних прокладок між підставою машини, приладу і опорною поверхнею. Як прокладок використовуються гума, повсть, пробка, різної конструкції амортизатори. Під настільні шумливі апарати можна підкладати м'які килимки з синтетичних матеріалів, а під ніжки столів, на яких вони встановлені, - прокладки з м'якої гуми, повсті, завтовшки 6 - 8 мм. Кріплення прокладок можливо шляхом приклеювання їх до опорних частин.

Застосовують спеціальні звукоізолюючих кожухи які не змінюють технологічний процес. Зниження шуму регулюють завдяки своєчасному змащуванню і заміні механічних вузлів шумливого обладнання.

Правильне планування приміщення є одним з важливих факторів зниження шумів при використані і устаткуванні ЕОМ.

Зниження рівня шуму, що проникає у виробниче приміщення ззовні, може бути досягнуто збільшенням звукоізоляції огороджувальних конструкцій, ущільненням по периметрі притворів вікон, дверей [3].

Для зниження шуму, створюваного на робочих місцях внутрішніми джерелами, а також шуму, проникаючого з поза слід:

—послабити шум самих джерел (застосування екранів, звукоізолюючих кожухів);

—знизити ефект сумарної дії відбитих звукових хвиль (звуко поглинаюча поверхня конструкцій);

—застосовувати раціональне розташування обладнання;

—використовувати архітектурно-планувальні та технологічні рішення ізоляцій джерел шуму.

8.5 Розробка заходів з умов поліпшення охорони праці

Одним з найбільш важливих завдань при розробці нових систем виробництва і технологій, відбувається за допомогою вивчення і вирішення проблем пов'язаних із забезпеченням безпечних умов праці людини. Вивчаючи причини виробничих нещасних випадків, професійних захворювань, пожеж розроблюють заходи спрямованні на усунення причин які створюють не безпеку. Одним з основних факторів безпеки праці є комфортні і безпечні умови праці [2].

Аналізуючи останні дослідження і публікації, модна виділити таких вчених як Д.П. Богиня, О.А. Грішнова, Є.П. Желібо, які зробили значний внесок в дослідженнях проблем з удосконалення системи охорони праці, вони проаналізували сучасний стан умов праці і виявили міру впливу їх на результати і ефективність виробництва. В ході досліджень виявили, що на підприємствах більше витрат припадає на пільги і компенсації, які по'вязані з небезпечними шкідливими умовами праці, ніж на заходи, що запобігають виробничому травматизму і захворюваності, а також нормалізацію умов праці.

Всі працівники проходять інструктаж з охорони праці, правил поведінки при виникненні пожеж, надання першої допомоги потерпілим відповідно до Типового положення про навчання з питань охорони праці. До капітальних витрат відносять,

такі як створення основних фондів праце охоронного призначення, вдосконалення техніки і технологій з мет ою поліпшення умов охорони праці, підвищувати кваліфікацію і відповідне зацікавлення працівників. Велику користь дає преміювання працівників за тривалу роботу без порушень правил охорони праці.

Керівнику треба звернути увагу на посилення контролю з організації служб охорони праці. Правильний підхід до організації, грамотне використання різних способів стимулювання працівників дають почуття надійності, стабільності і зацікавленості керівництва у працівниках [4].

8.6 Розрахунок освітлення виробничого приміщення

Освітлення відіграє важливу роль у житті людини, адже біля 90% інформації сприймається за допомогою зору. Тому правильно виконане раціональне освітлення має важливе значення всіх видів робіт, а також для попередження виробничого травматизму.

Раціональне освітлення повинно відповідати таким умовам:

1) бути достатнім (відповідним нормі);

2) рівномірним;

3) не утворювати тіней на робочій поверхні;

4) не засліплювати працюючого;

5) напрямок світлового потоку повинен відповідати зручному виконанню роботи.

Залежно від джерела світла виробниче освітлення може бути:

1) природне – передбачається у всіх приміщеннях з постійним перебуванням людей;

2) штучне – здійснюється штучними джерелами світла (газорозрядними лампами, світлодіодними лампами чи лампами розжарювання) і призначене для освітлення приміщень у темні години доби, або приміщень, які не мають природного освітлення;

3) комбіноване (сполучене або суміщене) – одночасне поєднання природного і штучного освітлення.

Освітлення робочого місця нормується згідно з Державними будівельними нормами України: ДБН В.2.5-28-2006 Інженерне обладнання будинків і споруд. Природне і штучне освітлення.

Мінімальна освітленість встановлюється в залежності від розряду виконуваних зорових робіт. Для IV розряду зорових робіт вона складає 300...500 лк.

Перевіримо освітленість робочого місця користувача ПК на відповідність розряду зорової роботи. За даними вимірювань рівень природної освітленості поверхні, де розташований ПК, складає 200 лк за освітленості тієї же поверхні відкритим небосхилом в 20000 лк, тобто КПО = 1%, що не відповідає нормативному КПО.

Для штучного освітлення у приміщенні використовуються люмінесцентні лампи.

Розрахунок штучного освітлення проведемо для кімнати площею 35 м^2, ширина якої складає 5м, довжина – 7м, висота – 3,5м.

Скористаємося методом використання світлового потоку. Для визначення потрібної кількості світильників, які повинні забезпечити нормований рівень освітленості. При проектуванні штучного освітлення необхідно виділити наступні питання:

1) вибрати систему освітлення;

2) вибрати тип джерела;

3) вибрати тип світильника;

4) визначити розташування світлових приладів;

5) виконати розрахунки штучного освітлення.

Вихідні дані для виконання розрахунку штучного освітлення:

— довжина приміщення 7000 мм;

— ширина приміщення 5000 мм;

— висота приміщення 3500 мм;

— висота робочої поверхні 1200 мм.

Для освітлення вибираємо світильник світлодіодного типу, у яких нормоване мінімальне освітлення складає 200 лк.

Висоту світильника над робочою поверхнею можна визначити за формулою 8.1.

$$h = h_{пр} - h_{рп}, \quad (8.1)$$

де h – висота світильника над робочою поверхнею;

hпр – висота приміщення, мм;

hрп – висота робочої поверхні, мм.

h = 3500 – 1200 = 2300 мм.

Коефіцієнт нерівномірності освітлення розраховується за формулою

$$Z = \frac{a \cdot b}{h \cdot (a + b)}, \quad (8.2)$$

де Z – відношення середньої освітленості до мінімальної.

$$Z = \frac{7000 * 5000}{2300 * (7000 + 5000)} = 1{,}2 \quad (8.3)$$

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення S=35м^2 ;

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1,1... 1,2, в нашому випадку Z =1,1);

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

η – коефіцієнт використання світлового потоку, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп, і обчи слюється в долях одиниці; залежить від характеристик

світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{ст.}$) і стелі ($\rho_{стелі}$)), значення коефіцієнтів дорівнюють $\rho_{ст}$ = 40% і ρстелі=60%.

При визначенні коефіцієнта нерівності освітленість дорівнює 1,26, а коефіцієнт використання світлового потоку світильника зі світлодіодними лампами складає 1(згідно довідників представлених інтернет ресурсів [6]).

Необхідну кількість світильників розраховують за формулою 8.3

$$F = \frac{E \cdot S \cdot K_з \cdot Z}{\eta}, \quad (8.4)$$

E – мінімальне освітлення, лк;

S – площа приміщення, м²;

Кз – коефіцієнт запасу;

Коефіцієнт запасу для штучного освітлення приймаємо рівним 1,3;

η – коефіцієнт використання світлового потоку.

Знаючи індекс приміщення I, за таблицею 4 [ДБН В.2.5-28-2006] знаходимо η =0,22.

Підставимо всі значення у формулу для визначення світлового потоку

$$F = \frac{200*35*1,3*1,2}{0,22} \approx 49000. \quad (8.5)$$

Для освітлення використані Світлодіодний світильник Лінійний еліпс -2 72вт 8640лм 1500мм [53]. Розрахуємо необхідну кількість ламп у світильниках за формулою:

$$N = \frac{F}{Fл}, \quad (8.6)$$

N – кількість ламп, що визначається; F - світловий потік, F = 45000 Лм; Fл- світловий потік лампи, Fл = 8640 Лм

(8.7)

$$N = \frac{49000}{8640} \approx 6 \text{ шт.}$$

На основі виконаних розрахунків можна зробити висновок, що для освітлення приміщення необхідно використати одну лампу світлодіодного типу, яку для забезпечення рівномірного освітлення необхідно розташувати по середині. Розміщення лампи зображено на рисунку 8.1.

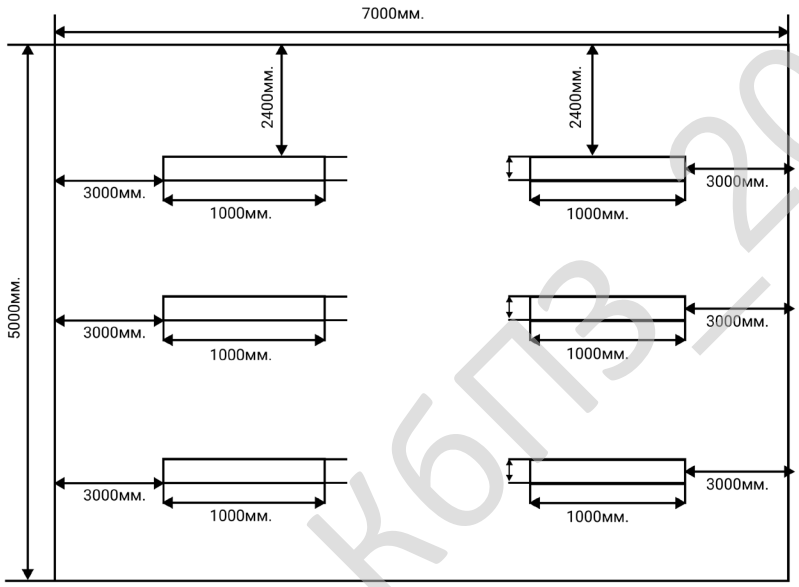


Рисунок 8.1 – Розміщення лампи в робочому приміщенні

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи, призначено для реалізації програмного забезпечення системи генерації та проходження лабіринтів для розробки відеоігор.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

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

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

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

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 7/8/10. Даються необхідні рекомендації з установки розробленого програмного забезпечення. Для підвищення рівня безпеки запропоновано застосовувати алгоритм AssetBundle in unity editor.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного

завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Довідник по C # | Mic rosoft Docs, Ключові слова C # , Директиви препроцесора, Параметри компілятора C #

2. Архітектура персонального комп'ютера, Тема 3 - Загальні принципи архітектури комп’ютерів, 3.1 Принципи побудови комп’ютера. Архитектура Фон Неймана, 3.3 Архітектура і структура ПК

3. Седерхольм, Д. Пуленепробиваемый дизайн. Библиотека специалиста / Д. Седерхольм. - СПб.: Питер, 2012. - 304 с.

4. Джозеф Хокинг, Unity в дії. Глава 10 Звукові ефекти та музика 242с.

5. Джозеф Хокинг, Unity в дії. Глава 11 Об'єднання фрагментів в готову гру 267с.

6. Джозеф Хокинг, Unity в дії. Глава 12 Розгортання ігор на пристроях гравців 298с.

7. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, глава 5. Рівень архітектури команд 334с.

8. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Глава 8. Архітектури комп'ютерів паралельної дії 556с.

9. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток А. Двійкові числа 663с.

10. Е. Таненбаум, Архітектура комп’ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток Б. Числа з плаваючою точкою 674с.

11. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 2. Лінійна алгебра 44с.

12. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 3. Теорія ймовірності і теорія інформації 61с.

13. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 5. Основи машинного навчання 96с.

14. Все про лабіринтах KUMON. Електроне джерело, Глава 1, Створення лабіринту 10с.

15. Бхаргава А. Грокаем алгоритми. (Пітер)

16. Кормен Томас Х., Лейзерсон Чарльз І., Ривест Рональд Л., Штайн Кліффорд, Алгоритми. Побудова і аналіз.

17. Генерація чисел C# [Електронний ресурс]. – Режим доступу: https://exponenta.ru/MLEM

18. Документація Unity [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/current/Manual/class-Collider.html

19. Колізії та межи [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/physics-in-unity-best-practice/

20. Приклади генераторів лабіринтів [Електронний ресурс]. – Режим доступу: https://nanotech-coder.ru/mazegnerator-dfs-1/

21. Maze Алгоритм генерації [Електронний ресурс]. – Режим доступу: https://ru.qwe.wiki/wiki/Maze_generation_algorithm/

22. Засоби генерацій лабіринтів в сфері відеоігор [Електронний ресурс]. – Режим доступу: https://cyberleninka.ru/article/n/sposoby-generatsii-labirintov-v-industrii-kompyuternyh-igr

23. Генерація Лабіринтів та їх проходження [Електронний ресурс]. – Режим доступу: https://progressor-blog.ru/qt/generatsiya-labirinta-i-ego-prohozhdenie/

24. Як пройти через лабіринт не заблукавш [Електронний ресурс]. – Режим доступу: http://www.ega-math.narod.ru/Nquant/Maze.htm

25. Теорія Лабіринтів [Електронний ресурс]. – Режим доступу: https://biography.wikireading.ru/182949

26. Лекція | Ейлерови графи [Електронний ресурс]. – Режим доступу: https://www.intuit.ru/studies/courses/58/58/lecture/1714?page=3

27. Проходження лабіринтів. Правила та алгоритми [Електронний ресурс].– Режим доступу: https://myrobot.ru/articles/logo_mazesolving.php

28. Фізика в Unity [Електронний ресурс].– Режим доступу: https://xgm.guru/p/unity/unityphysics

29. Оптимізація фізики робота Unity [Електронний ресурс].– Режим доступу: https://tproger.ru/translations/physics-in-unity-best-practice/

30. Як працює Drag в фізиці Unity [Електронний ресурс].– Режим доступу: https://habr.com/ru/post/318254/

31. Як створити фізичні ваги в Unity [Електронний ресурс].– Режим доступу: https://unity3dschool.ru/kak-sozdat-fizicheskiye-vesy-v-unity.html

32. Оптимізація в ігор Unity [Електронний ресурс].– Режим доступу: https://3dpapa.ru/optimize-the-game-for-unity-part-1/

33. Введення в Unity Develop [Електронний ресурс].– Режим доступу: https://students.njay.ru/article/Vvedenie-v-Unity3D-Sozdanie-2D-Igry-NLO

34. Mesh Components in Unity [Електронний ресурс].– Режим доступу: https://tproger.ru/translations/physics-in-unity-best-practice/

35. Проходження крізь стіни [Електронний ресурс].– Режим доступу: http://unity3d.ru/distribution/viewtopic.php?f=105&t=43538

36. Взаємодія об’єктів в Unity [Електронний ресурс].– Режим доступу: http://unity3d.ru/distribution/viewtopic.php?f=89&t=2746

37. AddForce відношення до об’єкту в Unity [Електронний ресурс].– Режим доступу: https://qna.habr.com/q/619526

38. Використання імпульсу [Електронний ресурс].– Режим доступу: https://husl.ru/questions/581717

39. Рух об’єкту в просторі Unity [Електронний ресурс].– Режим доступу: https://3dgame-creator.ru/catalog/uroki/unity5/unity-2d-dvizhenie-personazha/

40. Приклади скрипту рух об’єкту в просторі Unity [Електронний ресурс].– Режим доступу: https://null-code.ru/scripts/45-skript-dvizheniya-2d-personazha.html

41. Основи створення персонажів, об’єкту в просторі Unity [Електронний ресурс].– Режим доступу: https://habr.com/ru/post/211847/

42. Проблеми руху об’єкта [Електронний ресурс].– Режим доступу: http://www.unity3d.ru/distribution/viewtopic.php?f=18&t=2330

43. Система управління в 3D платформері [Електронний ресурс].– Режим доступу: https://unity3dschool.ru/sistema-upravlenija-v-3d-platformerah.html

44. Camera follow in unity3d [Електронний ресурс].– Режим доступу: https://answers.unity.com/questions/1370615/camera-follow-in-c-1.html

45. Movement Basics in unity3d [Електронний ресурс].– Режим доступу: https://learn.unity.com/tutorial/movement-basics

46. Unity3D: Third-Person Cameras [Електронний ресурс].– Режим доступу: https://code.tutsplus.com/tutorials/unity3d-third-person-cameras--mobile-11230

47. Керівництво Character Controller Unity [Електронний ресурс].– Режим доступу: https://docs.unity3d.com/ru/current/Manual/class-CharacterController.html

48. Камера третьої особи, її обертання навколо персонажа [Електронний ресурс].– Режим доступу: https://answers.unity.com/questions/1408897/third-person-camera-how-to-rotate-around-character.html

49. Navigation and Pathfinding in Unity3D [Електронний ресурс].– Режим доступу: https://docs.unity3d.com/Manual/Navigation.html

50. A* Pathfinding in Unity3D [Електронний ресурс].– Режим доступу: https://arongranberg.com/astar/

51. ДБН В.2.5-28:2018 [Електронний ресурс]. – Режим доступу : https://www.minregion.gov.ua/wp-content/uploads/2018/09/DBN_Osvitlennya-ostatochna.pdf

52. “З А К О Н У К Р А Ї Н И Про охорону праці” [Електронний ресурс]. – Режим доступу: https://xn--80aagahqwyibe8an.com/ukrajiny-zakony/zakon-ukrajini-

53. Інтернет-магазин fedsvet [Електронний ресурс]. – Режим доступу: https://fedsvet.com.ua/p1392245999-svetodiodnyj-svetilnik-linejnyj.html

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-122.21.0084.00.00.ТЗ			
Вим.	*Арк.*	*№ документа*	*Підпис*	*Дата*				
Розробив		*Пономаренко А.С*			*Дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор*	*Літ.*	*Аркуш*	*Аркушів*
Перевірив		*Мелешко Є.В.*				*М*	*1*	*6*
						ЦНТУ КН-20МЗ		
Н. Контр.		*Гермак В.С.*						
Затв.		*Смірнов О.А.*						

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи генерації та проходження лабіринтів для розробки відеоігор.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу, видане на кафедрі кібербезпеки та програмного забезпечення (наказ № 278-02 від 28.12.2019 року.).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи є дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор.

4 Джерела розробки

Джерелом випускної кваліфікаційної роботи є розробки, які ведуться спільнотою розробників компіляторів, ігрових рушіїв і стосовна до теми технічна література.

5 Технічні вимоги

5.1 Вміст проекту

Складовими розробки є:

– аналіз існуючих програмних аналогів;

– вибір і обґрунтування методики побудови додатків і засобів їхньої реалізації;

– розробка структур даних і механізму їхньої взаємодії, робочих форм і засобів;

– розробка головного вікна програми;

– розробка моделі штучного інтелекту для пошуку шляху;

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

– аналіз умов праці;

– розробка програми, яка реалізує алгоритми роботи компоненту.

5.2 Показники призначення

Система повинна забезпечувати:

– створення, генерації та проходження лабіринтів;

– простий, інтуїтивно зрозумілий інтерфейс з користувачем;

– Приклади пошуку шляхів та можливість створити перешкоди.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення повинно дозволити отримати систему генерації та проходження лабіринтів, приклади пошуку шляхів та створення перешкод.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен підтримувати роботу з системними компонентами ОС Windows.

5.5 Вимоги до надійності

Компонент повинен використати існуючі угоди по стандартним викликам процедур, функцій, засобів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Автоматизовані робочі місця користувачів системи повинні задовольняти наступним умовам експлуатації:

– температура повітря: 18-22^0 С;

– відносна вологість повітря при 20^0 С до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу і параметрів технічних засобів

ПЗ повинно бути реалізоване на ЕОМ типу IBM PC в операційному середовищі Windows 10 і орієнтований на сумісні з цією платформою зовнішні пристрої, мережне обладнання і прикладне програмне забезпечення.

5.8 Вимоги до інформаційної та програмної сумісності

Сумісність програмного забезпечення повинна бути забезпечена за рахунок використання бібліотеки UnityEnigen, System.Collections, Generic , яка сумісна з усіма останніми версіями операційної системи Microsoft Windows.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/2 Gb/128 Gb/SVGA 14” 512 Mb або сумісні з ним.

5.8.2 Мова програмування

C# з використанням можливостей програмної технології Microsoft .NET.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у вигляді опису схем і алгоритмів, інструкції користувача, а також текстів вхідних модулів програмного забезпечення в відповідності з ЄСПД.

7 Економічні вимоги

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

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2021 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинні бути розглянуті умови праці розробників при розробці та підтримці запропонованого програмного забезпечення.

9 Перелік документів, які необхідно розробити

– структурна схема системи повна;

– функціональна схема системи;

– блок-схеми алгоритму роботи системи;

– діаграма процесів;

– пояснювальна записка.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії структур даних.

10.5 Створення прототипу компоненту. Створення програмного продукту.

10.6 Відлагодження компоненту, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю і приймання

11.1 Подання випускної кваліфікаційної роботи на попередній захист 10.12.2021 р.

11.2 Подання випускної кваліфікаційної роботи на захист .12.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

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

Дослідження та програмна реалізація системи генерації та проходження лабіринтів для розробки відеоігор

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 31

Літера: РП

Кропивницький – 2021 року

Maze Generation.script

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

public class MazeGenerator
{
    public int Width = 50;
    public int Height = 50;

    public Maze GenerateMaze()
    {
        MazeGeneratorCell[,] cells = new MazeGeneratorCell[Width, Height];

        for (int x = 0; x < cells.GetLength(0); x++)
        {
            for (int y = 0; y < cells.GetLength(1); y++)
            {
                cells[x, y] = new MazeGeneratorCell {X = x, Y = y};
            }
        }

        for (int x = 0; x < cells.GetLength(0); x++)
        {
            cells[x, Height - 1].WallLeft = false;
        }

        for (int y = 0; y < cells.GetLength(1); y++)
        {
            cells[Width - 1, y].WallBottom = false;
        }

        RemoveWallsWithBacktracker(cells);

        Maze maze = new Maze();
        maze.cells = cells;
        maze.finishPosition = PlaceMazeExit(cells);

        return maze;
    }

    private void RemoveWallsWithBacktracker(MazeGeneratorCell[,] maze)
    {
        MazeGeneratorCell current = maze[0, 0];
        current.Visited = true;
        current.DistanceFromStart = 0;

        Stack<MazeGeneratorCell> stack = new Stack<MazeGeneratorCell>();
        do
        {
            List<MazeGeneratorCell> unvisitedNeighbours = new
List<MazeGeneratorCell>();

            int x = current.X;
            int y = current.Y;

            if (x > 0 && !maze[x - 1, y].Visited) unvisitedNeighbours.Add(maze[x
- 1, y]);
            if (y > 0 && !maze[x, y - 1].Visited)
unvisitedNeighbours.Add(maze[x, y - 1]);
            if (x < Width - 2 && !maze[x + 1, y].Visited)
unvisitedNeighbours.Add(maze[x + 1, y]);
            if (y < Height - 2 && !maze[x, y + 1].Visited)
```

```
unvisitedNeighbours.Add(maze[x, y + 1]);

            if (unvisitedNeighbours.Count > 0)
            {
                MazeGeneratorCell chosen =
unvisitedNeighbours[UnityEngine.Random.Range(0, unvisitedNeighbours.Count)];
                RemoveWall(current, chosen);

                chosen.Visited = true;
                stack.Push(chosen);
                chosen.DistanceFromStart = current.DistanceFromStart + 1;
                current = chosen;
            }
            else
            {
                current = stack.Pop();
            }
        } while (stack.Count > 0);
    }

    private void RemoveWall(MazeGeneratorCell a, MazeGeneratorCell b)
    {
        if (a.X == b.X)
        {
            if (a.Y > b.Y) a.WallBottom = false;
            else b.WallBottom = false;
        }
        else
        {
            if (a.X > b.X) a.WallLeft = false;
            else b.WallLeft = false;
        }
    }

    private Vector2Int PlaceMazeExit(MazeGeneratorCell[,] maze)
    {
        MazeGeneratorCell furthest = maze[0, 0];

        for (int x = 0; x < maze.GetLength(0); x++)
        {
            if (maze[x, Height - 2].DistanceFromStart >
furthest.DistanceFromStart) furthest = maze[x, Height - 2];
            if (maze[x, 0].DistanceFromStart > furthest.DistanceFromStart)
furthest = maze[x, 0];
        }

        for (int y = 0; y < maze.GetLength(1); y++)
        {
            if (maze[Width - 2, y].DistanceFromStart >
furthest.DistanceFromStart) furthest = maze[Width - 2, y];
            if (maze[0, y].DistanceFromStart > furthest.DistanceFromStart)
furthest = maze[0, y];
        }

        if (furthest.X == 0) furthest.WallLeft = false;
        else if (furthest.Y == 0) furthest.WallBottom = false;
        else if (furthest.X == Width - 2) maze[furthest.X + 1,
furthest.Y].WallLeft = false;
        else if (furthest.Y == Height - 2) maze[furthest.X, furthest.Y +
1].WallBottom = false;

        return new Vector2Int(furthest.X, furthest.Y);
    }
}
```

MazeSpawner.script

```
using UnityEngine;

public class MazeSpawner : MonoBehaviour
{
    public Cell CellPrefab;
    public Vector3 CellSize = new Vector3(1,1,0);
    public HintRenderer HintRenderer;

    public Maze maze;

    private void Start()
    {
        MazeGenerator generator = new MazeGenerator();
        maze = generator.GenerateMaze();

        for (int x = 0; x < maze.cells.GetLength(0); x++)
        {
            for (int y = 0; y < maze.cells.GetLength(1); y++)
            {
                Cell c = Instantiate(CellPrefab, new Vector3(x * CellSize.x, y *
CellSize.y, y * CellSize.z), Quaternion.identity);

                c.WallLeft.SetActive(maze.cells[x, y].WallLeft);
                c.WallBottom.SetActive(maze.cells[x, y].WallBottom);
            }
        }

        HintRenderer.DrawPath();
    }
}
```

HintRenderer.script

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

public class HintRenderer : MonoBehaviour
{
    public MazeSpawner MazeSpawner;

    private LineRenderer componentLineRenderer;

    private void Start()
    {
        componentLineRenderer = GetComponent<LineRenderer>();
    }

    public void DrawPath()
    {
        Maze maze = MazeSpawner.maze;
        int x = maze.finishPosition.x;
        int y = maze.finishPosition.y;
        List<Vector3> positions = new List<Vector3>();

        while ((x != 0 || y != 0) && positions.Count < 10000)
        {
            positions.Add(new Vector3(x * MazeSpawner.CellSize.x, y *
MazeSpawner.CellSize.y, y * MazeSpawner.CellSize.z));

            MazeGeneratorCell currentCell = maze.cells[x, y];

            if (x > 0 &&
                !currentCell.WallLeft &&
                maze.cells[x - 1, y].DistanceFromStart <
currentCell.DistanceFromStart)
            {
                x--;
            }
            else if (y > 0 &&
                !currentCell.WallBottom &&
                maze.cells[x, y - 1].DistanceFromStart <
currentCell.DistanceFromStart)
            {
                y--;
            }
            else if (x < maze.cells.GetLength(0) - 1 &&
                !maze.cells[x + 1, y].WallLeft &&
                maze.cells[x + 1, y].DistanceFromStart <
currentCell.DistanceFromStart)
            {
                x++;
            }
            else if (y < maze.cells.GetLength(1) - 1 &&
                !maze.cells[x, y + 1].WallBottom &&
                maze.cells[x, y + 1].DistanceFromStart <
currentCell.DistanceFromStart)
            {
                y++;
            }
        }
        positions.Add(Vector3.zero);
        componentLineRenderer.positionCount = positions.Count;
        componentLineRenderer.SetPositions(positions.ToArray());
    }
}
```

HuntAndKillMazeAlgorithm.script

```
using UnityEngine;
using System.Collections;

public class HuntAndKillMazeAlgorithm : MazeAlgorithm {

	private int currentRow = 0;
	private int currentColumn = 0;

	private bool courseComplete = false;

	public HuntAndKillMazeAlgorithm(MazeCell[,] mazeCells) : base(mazeCells) {
	}

	public override void CreateMaze () {
		HuntAndKill ();
	}

	private void HuntAndKill() {
		mazeCells [currentRow, currentColumn].visited = true;

		while (! courseComplete) {
			Kill(); // Will run until it hits a dead end.
			Hunt(); // Finds the next unvisited cell with an adjacent
visited cell. If it can't find any, it sets courseComplete to true.
		}
	}

	private void Kill() {
		while (RouteStillAvailable (currentRow, currentColumn)) {
			// int direction = Random.Range (1, 5);
			int direction = ProceduralNumberGenerator.GetNextNumber ();

			if (direction == 1 && CellIsAvailable (currentRow - 1,
currentColumn)) {
				// North
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn].northWall);
				DestroyWallIfItExists (mazeCells [currentRow - 1,
currentColumn].southWall);
				currentRow--;
			} else if (direction == 2 && CellIsAvailable (currentRow + 1,
currentColumn)) {
				// South
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn].southWall);
				DestroyWallIfItExists (mazeCells [currentRow + 1,
currentColumn].northWall);
				currentRow++;
			} else if (direction == 3 && CellIsAvailable (currentRow,
currentColumn + 1)) {
				// east
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn].eastWall);
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn + 1].westWall);
				currentColumn++;
			} else if (direction == 4 && CellIsAvailable (currentRow,
currentColumn - 1)) {
				// west
				DestroyWallIfItExists (mazeCells [currentRow,
currentColumn].westWall);
```

```
                        DestroyWallIfItExists (mazeCells [currentRow,
currentColumn - 1].eastWall);
                        currentColumn--;
                  }

                  mazeCells [currentRow, currentColumn].visited = true;
            }
      }

      private void Hunt() {
            courseComplete = true; // Set it to this, and see if we can prove
otherwise below!

            for (int r = 0; r < mazeRows; r++) {
                  for (int c = 0; c < mazeColumns; c++) {
                        if (!mazeCells [r, c].visited &&
CellHasAnAdjacentVisitedCell(r,c)) {
                              courseComplete = false; // Yep, we found something
so definitely do another Kill cycle.
                              currentRow = r;
                              currentColumn = c;
                              DestroyAdjacentWall (currentRow, currentColumn);
                              mazeCells [currentRow, currentColumn].visited =
true;
                              return; // Exit the function
                        }
                  }
            }
      }


      private bool RouteStillAvailable(int row, int column) {
            int availableRoutes = 0;

            if (row > 0 && !mazeCells[row-1,column].visited) {
                  availableRoutes++;
            }

            if (row < mazeRows - 1 && !mazeCells [row + 1, column].visited) {
                  availableRoutes++;
            }

            if (column > 0 && !mazeCells[row,column-1].visited) {
                  availableRoutes++;
            }

            if (column < mazeColumns-1 && !mazeCells[row,column+1].visited) {
                  availableRoutes++;
            }

            return availableRoutes > 0;
      }

      private bool CellIsAvailable(int row, int column) {
            if (row >= 0 && row < mazeRows && column >= 0 && column <
mazeColumns && !mazeCells [row, column].visited) {
                  return true;
            } else {
                  return false;
            }
      }

      private void DestroyWallIfItExists(GameObject wall) {
            if (wall != null) {
```

```
                GameObject.Destroy (wall);
            }
        }

        private bool CellHasAnAdjacentVisitedCell(int row, int column) {
            int visitedCells = 0;

            // Look 1 row up (north) if we're on row 1 or greater
            if (row > 0 && mazeCells [row - 1, column].visited) {
                visitedCells++;
            }

            // Look one row down (south) if we're the second-to-last row (or
less)
            if (row < (mazeRows-2) && mazeCells [row + 1, column].visited) {
                visitedCells++;
            }

            // Look one row left (west) if we're column 1 or greater
            if (column > 0 && mazeCells [row, column - 1].visited) {
                visitedCells++;
            }

            // Look one row right (east) if we're the second-to-last column (or
less)
            if (column < (mazeColumns-2) && mazeCells [row, column + 1].visited)
{
                visitedCells++;
            }

            // return true if there are any adjacent visited cells to this one
            return visitedCells > 0;
        }

        private void DestroyAdjacentWall(int row, int column) {
            bool wallDestroyed = false;

            while (!wallDestroyed) {
                // int direction = Random.Range (1, 5);
                int direction = ProceduralNumberGenerator.GetNextNumber ();

                if (direction == 1 && row > 0 && mazeCells [row - 1,
column].visited) {
                    DestroyWallIfItExists (mazeCells [row,
column].northWall);
                    DestroyWallIfItExists (mazeCells [row - 1,
column].southWall);
                    wallDestroyed = true;
                } else if (direction == 2 && row < (mazeRows-2) && mazeCells
[row + 1, column].visited) {
                    DestroyWallIfItExists (mazeCells [row,
column].southWall);
                    DestroyWallIfItExists (mazeCells [row + 1,
column].northWall);
                    wallDestroyed = true;
                } else if (direction == 3 && column > 0 && mazeCells [row,
column-1].visited) {
                    DestroyWallIfItExists (mazeCells [row,
column].westWall);
                    DestroyWallIfItExists (mazeCells [row, column-
1].eastWall);
                    wallDestroyed = true;
                } else if (direction == 4 && column < (mazeColumns-2) &&
mazeCells [row, column+1].visited) {
```

```
                        DestroyWallIfItExists (mazeCells [row,
column].eastWall);
                        DestroyWallIfItExists (mazeCells [row,
column+1].westWall);
                        wallDestroyed = true;
                    }
                }

        }

}
```

MazeLoader.script

```
using UnityEngine;
using System.Collections;

public class MazeLoader : MonoBehaviour {
	public int mazeRows, mazeColumns;
	public GameObject wall;
	public float size = 2f;

	private MazeCell[,] mazeCells;

	// Use this for initialization
	void Start () {
		InitializeMaze ();

		MazeAlgorithm ma = new HuntAndKillMazeAlgorithm (mazeCells);
		ma.CreateMaze ();
	}

	// Update is called once per frame
	void Update () {
	}

	private void InitializeMaze() {

		mazeCells = new MazeCell[mazeRows,mazeColumns];

		for (int r = 0; r < mazeRows; r++) {
			for (int c = 0; c < mazeColumns; c++) {
				mazeCells [r, c] = new MazeCell ();

				// For now, use the same wall object for the floor!
				mazeCells [r, c] .floor = Instantiate (wall, new Vector3
(r*size, -(size/2f), c*size), Quaternion.identity) as GameObject;
				mazeCells [r, c] .floor.name = "Floor " + r + "," + c;
				mazeCells [r, c] .floor.transform.Rotate (Vector3.right,
90f);

				if (c == 0) {
					mazeCells[r,c].westWall = Instantiate (wall, new
Vector3 (r*size, 0, (c*size) - (size/2f)), Quaternion.identity) as GameObject;
					mazeCells [r, c].westWall.name = "West Wall " + r
+ "," + c;
				}

				mazeCells [r, c].eastWall = Instantiate (wall, new
Vector3 (r*size, 0, (c*size) + (size/2f)), Quaternion.identity) as GameObject;
				mazeCells [r, c].eastWall.name = "East Wall " + r + ","
+ c;

				if (r == 0) {
					mazeCells [r, c].northWall = Instantiate (wall,
new Vector3 ((r*size) - (size/2f), 0, c*size), Quaternion.identity) as
GameObject;
					mazeCells [r, c].northWall.name = "North Wall " +
r + "," + c;
					mazeCells [r, c].northWall.transform.Rotate
(Vector3.up * 90f);
				}

				mazeCells[r,c].southWall = Instantiate (wall, new
Vector3 ((r*size) + (size/2f), 0, c*size), Quaternion.identity) as GameObject;
				mazeCells [r, c].southWall.name = "South Wall " + r +
```

```
"," + c;
                        mazeCells [r, c].southWall.transform.Rotate (Vector3.up
* 90f);
                }
            }
        }
}
```

Cell.script

```
using UnityEngine;

public class Cell : MonoBehaviour
{
    public GameObject WallLeft;
    public GameObject WallBottom;
}
```

Agent_navigation.script

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

public class Agent_navigation : MonoBehaviour
{

    public List<Vector3> waypoints;

    public int waypointIndex = 0;

    public int speed;

    public Graph_generation graphGeneration;

    public Transform target;


    public void Start()
    {
        waypoints = graphGeneration.wayPoints;
        graphGeneration.resetWaypoints();
    }

    public void Update()
    {
        waypoints = graphGeneration.wayPoints;
        move();
    }

    public void move()
    {
        if ((target.position - transform.position).magnitude > 1f &&
waypoints.Count == 0)
        {
            graphGeneration.resetWaypoints();
        }

        else if (waypoints.Count == 0)
        {
            moveFinalApproach();
        }

        else
        {
            followPath();
        }
    }


    public void followPath()
    {

        if (waypointIndex >= waypoints.Count)
        {
            return;
        }

        Vector3 movement = ((waypoints[waypointIndex] -
transform.position).normalized / 100) * speed;
        transform.position = transform.position + movement;
```

```
        if ((waypoints[waypointIndex] - transform.position).magnitude < .3f)
        {
            waypointIndex++;
            graphGeneration.resetWaypoints();
        }
    }


    public void moveFinalApproach()
    {
          Vector3 movement = ((target.position - transform.position).normalized
/ 100) * speed;
          transform.position = transform.position + movement;
    }

}
```

Graph_generation.script

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

public class Graph_generation : MonoBehaviour
{

    public GameObject tilePrefab;

    public Transform agent;

    public int width;

    public int height;


    public Node1[] nodes;

    public Transform pole;

    public Node1 previousStartNode;

    public List<Vector3> wayPoints;

    public Material[] materialTile;

    public int[] wallGrid;

    public GameObject wallPrefab;

    public int endPointPreviousPosition;
    public int startPointPreviousPosition;



    public void Awake()
    {
        wallGrid = new int[]
        {
            1,0,0,0,0,0,1,
            0,0,0,1,1,0,1,
            0,0,1,0,0,0,1,
            0,0,1,0,1,1,1,
            0,0,1,0,0,0,1,
            1,1,1,1,1,0,1,
            0,0,0,0,0,0,1
        };


        spawTiles();
        populateTileNeighbours();

        //wayPoints = new List<Vector3>();

        //findWayPoints(findEndNode(nodes[getObjectTilePos(agent)],
nodes[getObjectTilePos(pole)]));
    }

    public void Update()
    {
        //resetWaypoints();
    }
```

```
    public void resetWaypoints()
    {

        int endPointCurrentPos = getObjectTilePos(pole);

        int startPointCurrentPos = getObjectTilePos(agent);

        if (endPointCurrentPos != endPointPreviousPosition ||
startPointPreviousPosition != startPointCurrentPos)
        {
            Debug.Log("Reset waypoints");

            foreach (Node1 node in nodes)
            {
                node.visited = false;
                node.parent = null;
            }

            endPointPreviousPosition = endPointCurrentPos;

            startPointPreviousPosition = startPointCurrentPos;

            agent.GetComponent<Agent_navigation>().waypointIndex = 0;


            wayPoints = new List<Vector3>();

            findWayPoints(findEndNode(nodes[startPointCurrentPos],
nodes[endPointCurrentPos]));
        }


    }

    public void spawTiles()
    {
        nodes = new Node1[width * height];

        for (int z = 0, i = 0; z < height; z++)
        {
            for (int x = 0; x < width; x++, i++)
            {

                nodes[i] = new Node1(i, new Vector3(x * 10, 1f, z * 10));
                nodes[i].neighbours = new List<Node1>();



                // Spawn wall
                if (wallGrid[i] == 1)
                {
                    nodes[i].wall = true;
                    Instantiate(wallPrefab, new Vector3(x * 10, 5f, z * 10),
Quaternion.identity);
                }

                // Spawn tile
                else
                {
                    GameObject go = Instantiate(tilePrefab, new Vector3(x * 10,
0.005f, z * 10), Quaternion.identity);
                    // Change tile color to make a checkers style grid
                    go.GetComponent<MeshRenderer>().material = materialTile[i %
```

```
2];
                }

            }
        }

    }

    public void populateTileNeighbours()
    {
        for (int z = 0, i = 0; z < height; z++)
        {
            for (int x = 0; x < width; x++, i++)
            {

                //Debug.Log(nodes[i].ID);
                // Horizontal neighbours

                // Horizontal bounds
                if (nodes[i].location.x != 0 && nodes[i].location.x != (width-
1)*10)
                {
                    if (nodes[i - 1].wall == false)
                    {
                        nodes[i].neighbours.Add(nodes[i - 1]);
                    }

                    if (nodes[i + 1].wall == false)
                    {
                        nodes[i].neighbours.Add(nodes[i + 1]);
                    }
                }

                // In-between borders horizontal neighbours
                else
                {
                    if (nodes[i].location.x == 0)
                    {
                        if (nodes[i + 1].wall == false)
                        {
                            nodes[i].neighbours.Add(nodes[i + 1]);
                        }
                    }

                    else if(nodes[i].location.x == (width - 1)*10)
                    {
                        if (nodes[i - 1].wall == false)
                        {
                            nodes[i].neighbours.Add(nodes[i - 1]);
                        }
                    }
                }

                //Vertical neighbours
                if (nodes[i].location.z != 0 && nodes[i].location.z != (height -
1) * 10)
                {

                    if (nodes[i - width].wall == false)
                    {
                        nodes[i].neighbours.Add(nodes[i - width]);
                    }

                    if (nodes[i + width].wall == false)
```

```
                {
                    nodes[i].neighbours.Add(nodes[i + width]);
                }

            }

            // In-between borders horizontal neighbours
            else
            {
                if (nodes[i].location.z == 0)
                {
                    if (nodes[i + height].wall == false)
                    {
                        nodes[i].neighbours.Add(nodes[i + height]);
                    }
                }

                else if (nodes[i].location.z == (height - 1) * 10)
                {
                    if (nodes[i - height].wall == false)
                    {
                        nodes[i].neighbours.Add(nodes[i - height]);
                    }

                }
            }



        }
    }
}

public Node1 findEndNode(Node1 startNode, Node1 endNode)
{
    Queue<Node1> nodeQueue = new Queue<Node1>();

    endNode.visited = true;

    nodeQueue.Enqueue(endNode);

    while (nodeQueue.Count != 0)
    {
            Node1 currentNode = nodeQueue.Dequeue();

            foreach(Node1 neighbour in currentNode.neighbours)
            {

                if (neighbour.visited == false)
                {
                    // Add current node to neighbour as its parent
                    neighbour.parent = currentNode;


                    // Check if neighbour is the start node
                    if (neighbour == startNode)
                    {
                        return neighbour.parent;
                    }

                    // Mark neighbour as visited
                    neighbour.visited = true;
```

```
                            // Add neighbour to queue
                            nodeQueue.Enqueue(neighbour);
                        }
                    }
            }


        // If once all nodes have been visited the end node can't be found,
return null.
        return null;

    }

    public void findWayPoints(Node1 node)
    {
        Debug.Log("Find waypoints Running");

        if(node == null)
        {
            return;
        }

        if (node.parent == null)
        {
            wayPoints.Add(node.location);
            return;
        }

        else
        {
            wayPoints.Add(node.location);
            findWayPoints(node.parent);
        }

        agent.GetComponent<Agent_navigation>().target = pole;

    }

    public int getObjectTilePos(Transform someObject)
    {
        int x = (int)Mathf.Round(someObject.position.x/10);
        //Debug.Log("x=" + x);
        int z = (int)Mathf.Round(someObject.position.z/10);
        //Debug.Log("z=" + z);

        return x + z * width;

    }


    public void OnDrawGizmos()
    {
        foreach(Vector3 waypoint in wayPoints)
        {
            Gizmos.color = Color.red;
            Gizmos.DrawSphere(waypoint, 0.5f);
        }

    }


}
```

```
[System.Serializable]
public class Node1
{
    public int ID;

    public Vector3 location;

    public Node1 parent;

    public bool wall;

    public List<Node1> neighbours;

    public bool visited;

    public Node1(int _ID, Vector3 _location)
    {
        ID = _ID;
        location = _location;
        visited = false;
    }
}
```

Agent.script

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

public class Agent : MonoBehaviour {
      public float moveSpeed;

      public void FollowPath(List<Vector3> path) {
            StopCoroutine("FollowPathCo");
            StartCoroutine("FollowPathCo", path);
      }

      private IEnumerator FollowPathCo(List<Vector3> path) {
            foreach (var pos in path) {
                  Vector3 checkPoint = pos;
                  checkPoint.y = transform.position.y;

                  while (transform.position != checkPoint) {
                        transform.position =
Vector3.MoveTowards(transform.position, checkPoint, moveSpeed);
                        yield return null;
                  }
            }
      }
}
```

MouseInputHandler.script

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

public class MouseInputHandler : MonoBehaviour {
	private LayerMask tileLayer;

	[SerializeField]
	private Agent agent;

	[SerializeField]
	PathfindingManager pathfindingManager;

	private void Awake() {
		tileLayer = LayerMask.GetMask("Tile");
	}

	private void Update() {
		if (Input.GetMouseButtonDown(0) || Input.GetMouseButtonDown(1)) {
			Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
			RaycastHit result;

			if (Physics.Raycast(ray, out result, Mathf.Infinity,
tileLayer)) {
				if (Input.GetMouseButtonDown(0)) {
					Tile tile = result.collider.GetComponent<Tile>();
					if (tile != null) tile.ToggleObstacle();
				}
				if (Input.GetMouseButtonDown(1)) {
					Vector3 destPosition =
result.collider.transform.position;

					Node deptNode =
pathfindingManager.PositionToNode(agent.transform.position);
					Node destNode =
pathfindingManager.PositionToNode(destPosition);


	agent.FollowPath(pathfindingManager.FindPath(deptNode, destNode));
				}
			}
		}
	}
}
```

MyMathf.class

```
public static class MyMathf {
	public static int Sign(int num) {
		if (num == 0) return 0;
		return num > 0 ? 1 : -1;
	}

	public static bool IsInt(float num) {
		return num == (int)num;
	}
}
```

Node.script

```
using System;

public class Node : IComparable<Node> {
	public Node(int x, int y, Tile tile) {
		this.x = x;
		this.y = y;
		this.tile = tile;
	}

	// Let PathfindingManager check line of sight?
	public static Node[,] nodes;
	public int x;
	public int y;

	public int toCost;
	public int fromCost;

	public Tile tile;
	public Node from;

	public int lastHeapIndex;

	public int cost {
		get {
			return toCost + fromCost;
		}
	}

	public bool isBlocked {
		get {
			return tile.isBlocked;
		}
	}

	// :O
	public int CompareTo(Node other) {
		if (other == null) return -1;
		int result = cost.CompareTo(other.cost);
		if (result == 0) result = fromCost.CompareTo(other.fromCost);
		return -result;
	}

	public Point4 GetContainingPoints() {
		Point4 points = new Point4();
		points.point0 = new Point(x, y);
		points.point1 = new Point(x + 1, y);
		points.point2 = new Point(x, y + 1);
		points.point3 = new Point(x + 1, y + 1);
		return points;
	}

	public bool IsLineOfSight(Node end) {
		bool inSight;

		if (x == end.x) {
			int dy = MyMathf.Sign(end.y - y);
			if (dy == 0) return !tile.isBlocked;

			inSight = true;
			for (int sy = y; sy != end.y + dy; sy += dy) {
				if (nodes[x, sy].tile.isBlocked) {
					inSight = false;
```

```
                        break;
                    }
                }
                return inSight;
            }
            else if (y == end.y) {
                int dx = MyMathf.Sign(end.x - x);
                //if (dx == 0) return !tile.isBlocked;

                inSight = true;
                for (int sx = x; sx != end.x + dx; sx += dx) {
                    if (nodes[sx, y].tile.isBlocked) {
                        inSight = false;
                        break;
                    }
                }
                return inSight;
            }

            Point4 startPoints = GetContainingPoints();
            Point4 endPoints = end.GetContainingPoints();

            inSight = true;
            for (int i = 0; i < 4; i++) {
                if (!startPoints[i].IsLineOfSight(endPoints[i])) {
                    inSight = false;
                    break;
                }
            }
            return inSight;
        }
    }
```

NodeHeap.script

```
using System;

/*
 * Specific data structure for A* pathfinding
 * Basically binary heap but has Contains() method with O(1)
 * Could have used another HashSet for Contains()
 */
public class NodeHeap {
	private Node[] items;
	private int tail = 0;
	private int capacity;

	public int Count {
		get {
			return tail;
		}
	}

	public NodeHeap(int capacity = 4) {
		this.capacity = capacity;
		items = new Node[capacity];
	}

	private void Grow() {
		int newCapacity = capacity * 2;
		Node[] newItems = new Node[newCapacity];
		Array.Copy(items, newItems, capacity);
		items = newItems;
		capacity = newCapacity;
	}

	public void Add(Node item) {
		if (Count == capacity) Grow();

		item.lastHeapIndex = tail;
		items[tail++] = item;
		SortUp(item);
	}

	public Node Pop() {
		if (Count == 0) throw new InvalidOperationException("NodeHeap is
empty");

		Node firstItem = items[0];
		items[0] = items[--tail];
		items[0].lastHeapIndex = 0;
		SortDown(items[0]);
		return firstItem;
	}

	public void UpdateItem(Node item) {
		SortUp(item);
	}

	public bool Contains(Node item) {
		if (item.lastHeapIndex >= tail) return false;
		return items[item.lastHeapIndex] == item;
	}

	void SortDown(Node item) {
		while (true) {
			int childIndexLeft = item.lastHeapIndex * 2 + 1;
```

```
                int childIndexRight = item.lastHeapIndex * 2 + 2;
                int swapIndex = 0;

                if (childIndexLeft < tail) {
                      swapIndex = childIndexLeft;

                      if (childIndexRight < tail) {
                            if
(items[childIndexLeft].CompareTo(items[childIndexRight]) < 0) {
                                  swapIndex = childIndexRight;
                            }
                      }

                      if (item.CompareTo(items[swapIndex]) < 0) {
                            Swap(item, items[swapIndex]);
                      }
                      else {
                            return;
                      }

                }
                else {
                      return;
                }

          }
    }

    private void SortUp(Node item) {
          int parentIndex = (item.lastHeapIndex - 1) / 2;

          while (true) {
                Node parentItem = items[parentIndex];
                if (item.CompareTo(parentItem) > 0) {
                      Swap(item, parentItem);
                }
                else {
                      break;
                }

                parentIndex = (item.lastHeapIndex - 1) / 2;
          }
    }

    private void Swap(Node a, Node b) {
          items[a.lastHeapIndex] = b;
          items[b.lastHeapIndex] = a;
          int itemAIndex = a.lastHeapIndex;
          a.lastHeapIndex = b.lastHeapIndex;
          b.lastHeapIndex = itemAIndex;
    }
}
```

PathfindingManager.script

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

public class PathfindingManager : MonoBehaviour {
	[SerializeField]
	private Transform tileHolder;
	private Node[,] nodes;

	private int width;
	private int height;
	private Vector3 min;
	private Vector3 max;
	private float gapX;
	private float gapZ;

	/*
	 * Tiles should be shape of
	 *               1
	 *    0
	 */
	private void Awake() {
		List<Tile> tiles = new List<Tile>();
		for (int i = 0; i < tileHolder.childCount; i++) {
			tiles.Add(tileHolder.GetChild(i).GetComponent<Tile>());
		}

		min = tiles[0].transform.position;
		max = tiles[1].transform.position;

		width = (int)(max.x - min.x) + 1;
		height = (int)(max.z - min.z) + 1;

		gapX = (max.x - min.x) / width;
		gapZ = (max.z - min.z) / height;

		nodes = new Node[width, height];
		foreach (var tile in tiles) {
			int x = (int)(tile.transform.position.x - min.x);
			int y = (int)(tile.transform.position.z - min.z);

			nodes[x, y] = new Node(x, y, tile);
		}
		Node.nodes = nodes;
		Point.nodes = nodes;
	}

	private List<Node> GetAdjacents(Node node) {
		List<Node> adjacents = new List<Node>();

		for (int dx = -1; dx <= 1; dx++) {
			for (int dy = -1; dy <= 1; dy++) {
				if (dx == 0 && dy == 0) continue;

				int x = node.x + dx;
				int y = node.y + dy;

				if (x >= 0 && x < width && y >= 0 && y < height) {
					if (nodes[x, y].isBlocked) continue;

					if (dx == 0 || dy == 0) {
```

```
                                    adjacents.Add(nodes[x, y]);
                              }
                              else {
                                    if (!nodes[node.x, y].isBlocked && !nodes[x,
node.y].isBlocked)
                                          adjacents.Add(nodes[x, y]);
                              }
                        }
                  }
            }
            return adjacents;
      }

      public bool CheckPathExist(Node deptNode, Node destNode) {
            return FindPath(deptNode, deptNode).Count > 0;
      }

      public List<Vector3> FindPath(Node deptNode, Node destNode) {
            NodeHeap openHeap = new NodeHeap();
            HashSet<Node> closedSet = new HashSet<Node>();

            openHeap.Add(deptNode);
            deptNode.fromCost = deptNode.toCost = 0;
            deptNode.from = destNode.from = null;

            while (openHeap.Count > 0) {
                  Node nodeToVisit = openHeap.Pop();
                  closedSet.Add(nodeToVisit);

                  if (nodeToVisit == destNode) {
                        break;
                  }

                  foreach (var adjacent in GetAdjacents(nodeToVisit)) {
                        if (closedSet.Contains(adjacent)) continue;

                        Node fromNode = null;
                        if (nodeToVisit.from != null &&
nodeToVisit.from.IsLineOfSight(adjacent))
                              fromNode = nodeToVisit.from;
                        else
                              fromNode = nodeToVisit;

                        int toAdjacentCost = fromNode.toCost +
PredictDistanceCost(fromNode, adjacent);
                        if (!openHeap.Contains(adjacent) || toAdjacentCost <
adjacent.toCost) {
                              adjacent.toCost = toAdjacentCost;
                              adjacent.fromCost = PredictDistanceCost(adjacent,
destNode);
                              adjacent.from = fromNode;

                              if (openHeap.Contains(adjacent))
openHeap.UpdateItem(adjacent);
                              else openHeap.Add(adjacent);
                        }
                  }
            }

            List<Node> path = new List<Node>();
            if (destNode.from != null) {
                  Node currentNode = destNode;

                  while (currentNode != null) {
```

```
                        path.Add(currentNode);
                        currentNode = currentNode.from;
                  }
                  path.Reverse();
            }

            // PS
            // SmoothingPath(path);

            List<Vector3> positionPath = new List<Vector3>();

            // Visit departure node looks wired when path is changed
            // Skip first(departure) node
            for (int i = 1; i < path.Count; i++) {
                  positionPath.Add(path[i].tile.transform.position);
            }
            return positionPath;
      }

      private void SmoothingPath(List<Node> path) {
            for (int i = 1; i < path.Count - 1; i++) {
                  if (path[i - 1].IsLineOfSight(path[i + 1])) {
                        path.RemoveAt(i--);
                  }
            }
      }

      private int PredictDistanceCost(Node dept, Node dest) {
            int dx = Mathf.Abs(dept.x - dest.x);
            int dy = Mathf.Abs(dept.y - dest.y);

            return Mathf.Abs(dx - dy) * 10 + (dx > dy ? dy : dx) * 14;
      }

      public Node PositionToNode(Vector3 position) {
            int x = Mathf.Clamp((int)((position.x - min.x) / gapX), 0, width);
            int y = Mathf.Clamp((int)((position.z - min.z) / gapZ), 0, height);
            return nodes[x, y];
      }
}
```

Point.script

```
using System;
using UnityEngine;

public struct Point {
	public int x;
	public int y;
	public static Node[,] nodes;

	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}

	public bool IsLineOfSight(Point end) {
		int dfx = end.x - x;
		int dfy = end.y - y;

		bool inSight = true;
		if (Mathf.Abs(dfx) > Mathf.Abs(dfy)) {
			int dx = MyMathf.Sign(dfx);
			float prevY = y;

			for (int sx = x + dx; sx != end.x + dx; sx += dx) {
				float sy = y + (dfy * (sx - x)) / (float)dfx;

				int nx = dx > 0 ? sx - dx : sx;
				int ny = Mathf.Min((int)sy, (int)prevY);
				if (nodes[nx, ny].isBlocked) {
					inSight = false;
					break;
				}

				// Line is contained in two nodes; check another
node(upper bound)
				if ((int)prevY != (int)sy && !MyMathf.IsInt(prevY) &&
!MyMathf.IsInt(sy)) {
					ny = Mathf.Max((int)sy, (int)prevY);
					if (nodes[nx, ny].isBlocked) {
						inSight = false;
						break;
					}
				}
				prevY = sy;
			}
		}
		else {
			int dy = MyMathf.Sign(dfy);
			float prevX = x;

			for (int sy = y + dy; sy != end.y + dy; sy += dy) {
				float sx = x + (dfx * (sy - y)) / (float)dfy;

				int ny = dy > 0 ? sy - dy : sy;
				int nx = Mathf.Min((int)sx, (int)prevX);
				if (nodes[nx, ny].isBlocked) {
					inSight = false;
					break;
				}

				if ((int)prevX != (int)sx && !MyMathf.IsInt(prevX) &&
!MyMathf.IsInt(sx)) {
					nx = Mathf.Max((int)sx, (int)prevX);
```

```
                              if (nodes[nx, ny].isBlocked) {
                                    inSight = false;
                                    break;
                              }
                        }
                        prevX = sx;
                  }
            }
            return inSight;
      }
}

/*
 * Point[4] array.
 * Point array with size 4 is used a lot, so I made
 * value-type Point array. (for less GC call)
 */
public struct Point4 {
      public Point point0;
      public Point point1;
      public Point point2;
      public Point point3;

      public Point this[int idx] {
            get {
                  switch (idx) {
                        case 0: return point0;
                        case 1: return point1;
                        case 2: return point2;
                        case 3: return point3;
                  }
                  throw new InvalidOperationException("Point index out of range"
+ idx);
            }
            set {
                  switch (idx) {
                        case 0:
                              point0 = value;
                              break;
                        case 1:
                              point1 = value;
                              break;
                        case 2:
                              point2 = value;
                              break;
                        case 3:
                              point3 = value;
                              break;
                        default:
                              throw new InvalidOperationException("Point index
out of range" + idx);
                  }
            }
      }
}
```

Tile.script

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

public class Tile : MonoBehaviour {
      public bool isBlocked;

      private GameObject obstacle;

      private void OnValidate() {
            ToggleObstacle(true);
      }

      public void ToggleObstacle(bool updateOnly = false) {
            if (!updateOnly) isBlocked = !isBlocked;
            transform.Find("Obstacle").gameObject.SetActive(isBlocked);
      }
}
```