Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ГРИ НА UNITY/C# З ІМПЛЕМЕНТАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
с.в. Борозенний С.О.
(прізвище та ініціали)
_____
(підпис)
“__”_____ 2022 р.

Виконав студент
Погодічев І.Д.
(прізвище та ініціали)
“__”_____ 2022 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____ О. П. Жежерун (підпис)
“___”_____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Погодічеву Івану Дмитровичу факультету інформатики 3-го курсу

ТЕМА Розробка гри на UNITY/C# з імплементацією штучного інтелекту

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі “__”_____ 2022 р. Борозенний О.С. _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Розробка гри на UNITY/C# з імплементацією штучного інтелекту.

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	14.01.2022	
2.	Аналіз матеріалів за темою	14.03.2022	
3.	Розробка та програмування алгоритмів	14.05.2022	
4.	Написання текстової частини до курсової роботи	23.05.2022	
5.	Коригування виконаної роботи	06.06.2022	
6.	Захист курсової роботи	13.06.2022	

Погодічев І.Д. _____

Борозенний О.С. _____

“___” _____ 2022 р.

Зміст

Анотація

Робота присвячена розробці інструментів для тих, хто хоче створювати унікальні світи у свої проєктах та логіку пересування великої кількості сутностей у просторі. Задля наглядного відображення цих інстрементів, робота виконана у вигляді гри, на нагальну тему.

Ключові слова: процедурна генерація, генерація світу, рух товпи, пошук шляху, Unity, Navigation, NavMesh.

Вступ

Постановка завдання

Мета моєї роботи полягає в тому, щоб показати як створюються відеоігри, які прийоми та технології використовуються сьогодні. Усі люди так чи інакше грають в комп’ютерні, мобільні або консольні ігри. Для прикладу буде створено гру на дуже актуальну тему. Кожного разу генерується унікальне місто, гравець грає за умовний БПЛА і повинен уражати ворогів. Корисно почитати буде кожному, як розробникам відеоігор, так і користувачам.

Основна частина роботи складається з 3 розділів.

Перший розділ містить інформацію про історію та розвиток штучного інтелекту у відеоіграх, особливо про процедурну генерацію та пошук шляху для группи NPC.

Другий розділ присвячений процесу генерації світу. А саме алгоритму, який я використав та його імплементації.

У третьому розділі розглядаються деталі алгоритму пошуку шляху для великої кількості NPC (non-player characters) - ботів, який виркористовується у практичній частині та симуляція їх поведінки.

Основна частина

Розділ 1: Історія та приклади використання штучного інтелекту у відеоіграх: процедурна генерація та симуляція натовпу

AI – artificial intelligence – штучний інтелект – у наш час не потребує представлення. Всі знають що це та яка це розповсюджена у наш час річ. Ми бачимо роботу штучного інтелекту у смартфонах, автомобілях і побутовій техніці. Відеігри не є вийнятком, а напвпаки – один з найяскравіших представників використання штучного інтелекту, адже ми можемо побачити його роботу очима та навіть повзаємодіяти з ним.

Отже, які є різновиди застосування штучного інтелекту у відеоіграх? Розглянемо найбільш популярні:

1. NPC – non-player characters – неігрові персонажі – найбільш розповсюджений приклад AI, оскільки у наш час складно уявити гру без них. Їхня поведінка завжди визначається алгоритмами штучного інтелекту. Дуже часто behaviour trees - дерева прийняття рішень визначають поведінку NPC.
2. Data mining – збір даних – надважливий і розповсюджений вид штучного інтелекту у будь-якій сфері інформаційних технологій. Data mining дозволяє компаніям розробникам краще зрозуміти поведінку користувача і зрозуміти як покращити їхній продукт.
3. PCG – procedural content generation – процедурна генерація контенту – створення будь-якого контенту або даних комп’ютером самостійно або при взаємодії з розробником, ігровим дизайнером або гравцем. Це один з варіантів штучного інтелекту, який я використав у своїй роботі. Надалі я розповім детальніше що це таке.

4. Pathfinding – пошук шляху – якщо коротко, то пошук оптимального, коротшого або вигіднішого шляху з однієї точки до іншої. Власне, це і є друга частина моєї роботи, про яку я теж розповім детальніше.

1.1 Що таке процедурна генерація контенту

Процедурна генерація – це створення будь-якого контенту або даних комп'ютером самойстійно або при взаємодії з розробником, гейм дизайнером або гравцем [1]. Мета використання процедурної генерації може бути різною: спростити або пришвидшити процес розробки, урізноманітнити контент або адаптувати контент персонально для кожного гравця.

Здебільшого, цей підхід використовується у відеоіграх або сучасній анімації для того щоб створювати унікальний контент. Наприклад, предмети декорацій, діалоги, унікальні рівні, дизайн персонажів або, найчастіше, 2D та 3D світи. Яскравими прикладами процедурної генерації світу або мапи можуть бути такі відеоігри, як: Civilization, Don't Starve, Factorio, всім відомий Minecraft, та інші.

1.2 Pathfinding та симуляція натовпу.

Пошук шляху (*з англійської Pathfinding*) це пошук шляху з однієї точки вашого світу до іншої. Ігровий світ і є найважливішою частиною пошуку шляху. Ігровий штучний інтелект повинен вміти точно прораховувати свій маршрут в заленості від різних перешкод які можуть трапитися йому на шляху. Для того щоб зрозуміти, для чого потрібен Pathfinding саме в моїй роботі, треба розглянути Crowd Simulation [6].

Crowd Simulation або симуляція натовпу – це симуляція великої кількості людей, тварин або інших ігрових сутностей, кожна з яких взаємодіє з іншою та навколишнім середовищем. Кожну сутність прийнято називати агентом (*з англійської Agent*). Кожний агент повинен вміти

взаємодіяти зі світом, в якому він знаходиться, та іншшими агентами. Але найголовніша його задача, вміти вправно пересуватися від однієї своєї цілі до іншої. Саме тому пошук шляху є невід'ємною частиною симуляції натовпу і його необхідно було розглянути. Також у моїй роботі присутній Fleeing Algorithm [6] – алгоритм втечі, який теж буде розглянуто.

Симуляція натовпу має дуже багато застосувань та користі. Самостійний та гарно продуманий алгоритм симуляції натовпу зекономить багато часу на розробці та ресурсів коомп'ютерів користувачів, а отже грошей. Також не рідкість коли він використовується у сучасному кіно, адже з новітніми технологіями комп'ютерної графіки ніхто не помітить різницю між, наприклад, намальованим натовпом на пішохідному переході або справжнім, в той же час зекономить багато грошей на акторах. Зустріти його можна дуже багато де і у відеоіграх, наприклад, всім відома GTA або симуляція трафіку у Need For Speed.

Розділ 2: Procedural Content Generation

Розглянемо процедурну генерацію контенту з технічної точки зору. Є дуже багато різних підходів та методів того, як все ж таки працює процедурна генерація. Наприклад за допомогою генератора випадкових чисел, як у Civilization, або Perlin Noise (*з англійської шум Перліна*), як у Minecraft. Я у своїй роботі використав не дуже популярний, але дуже цікавий метод L-System – систему Лінденмайєра [5], яка перекликається багатьма курсами, які присутні у нашому університеті.

2.1 Система Лінденмайєра

Система Лінденмайєра – математичне пояснення росту рослин, запроваджене біологом Лінденмайєром у 1968 році. Оскільки Л-Система описує ріст, то вона повинна мати «коріння» і правила по яким будуються наступні ітерації росту – **axiom** and **rules** – аксіома та правила. Також потрібно ввести поняття глибини алгоритму – **depth** – це кількість ітерацій яку пройде алгоритм.

Приклад:

Axiom: B

Rules: B → F[-B]+B, F → FF

Depth = 4, тоді результуюча таблиця буде наступна:

Depth	**Resulting String – Результуюча стрічка**
0	B
1	F[-B]+B
2	FF[-F[-B]+B]+ F[-B]+B
3	FFFF[-FF[-F[-B]+B]+ F[-B]+B]+ FF[-F[-B]+B]+ F[-B]+B

Table 1 Одночасні заміни рядків у L-системі

Довжина стрічки росте дуже стрімко, але що взагалі значить стрічка та як воно допоможе нам побудувати місто для гри?

2.2 L-System Procedural Fractal Generation

Ми впевнилися, що наш алгоритм рекурсивний - кожна наступна ітерація побудована на результаті попередньої. Якщо ми спроектуємо це на мапу і скажемо що одні символи означають малювання лінії вперед, а інші поворот в одну чи іншу сторону на певний кут, то ми відкриємо для себе іншу сторону цього алгоритму – fractal structures. Якщо просто, то фрактали це такі структури форма яких подібна одна на одну. Отже, якщо ввести визначення кожному символу, наприклад, згідно таблиці, яку запровадив сучасний дослідник Гері Вільям Флейк [6], на основі старого концепту математика Сеймура Пейперта.

Cmd.	Turtle Action
F	(Draw Forward) Move the turtle forward by a fixed length, drawing a line from the old position to the new position.
G	(Go Forward) Move the turtle forward by a fixed length, but do not draw a line.
+	Turn the turtle to the right by a fixed angle. If an integer precedes the "+" symbol, then the turtle effectively makes that number of right-hand turns.
−	Turn the turtle to the left by a fixed angle. If an integer precedes the "−" symbol, then the turtle effectively makes that number of left-hand turns.
[	Save the turtle's current position and angle for later use onto a stack of saved states.
]	Remove the last saved state from the stack and use it to restore the turtle's last saved position and angle.
|	Move the turtle forward by a length computed from the execution depth, drawing a line from the old position to the new.

Table 2 Графічні правила [6]

Базуючись на цих правилах, можна зробити графічну візуалізацію Таблиці 1. Єдине, що нам потрібно, це ввести кут повороту для + і -. Нехай він буде 45°. Отримуємо наступне відображення:

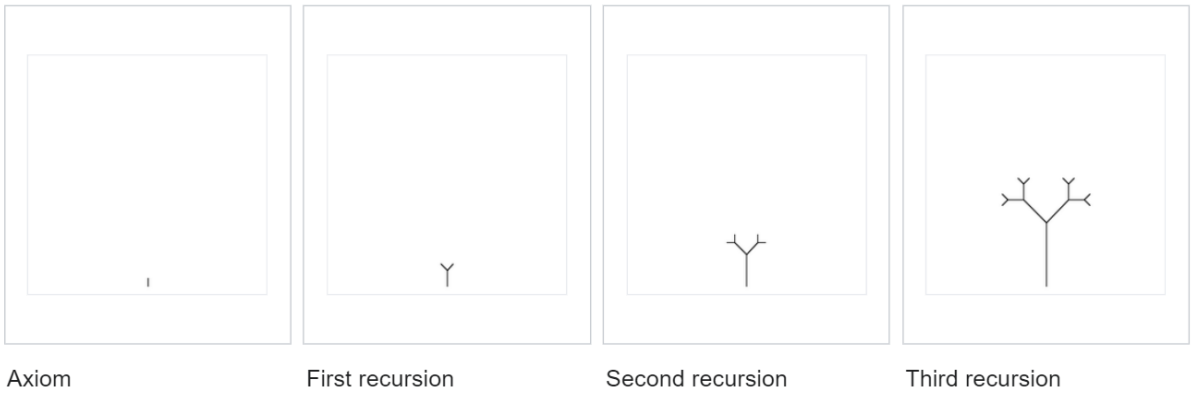

Рисунок 1 4 Стадії генерації Л-Системи

Таким чином, задаючи різні параметри можна отримувати генерації різних фракталів, наведемо ще декілька прикладів:

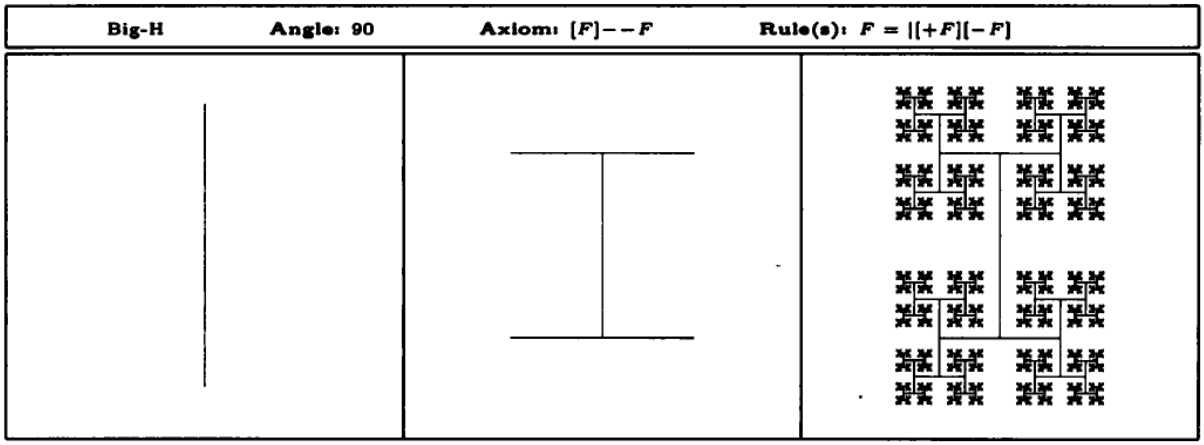

Рисунок 2 Приклад генерації L-System 1 [6]

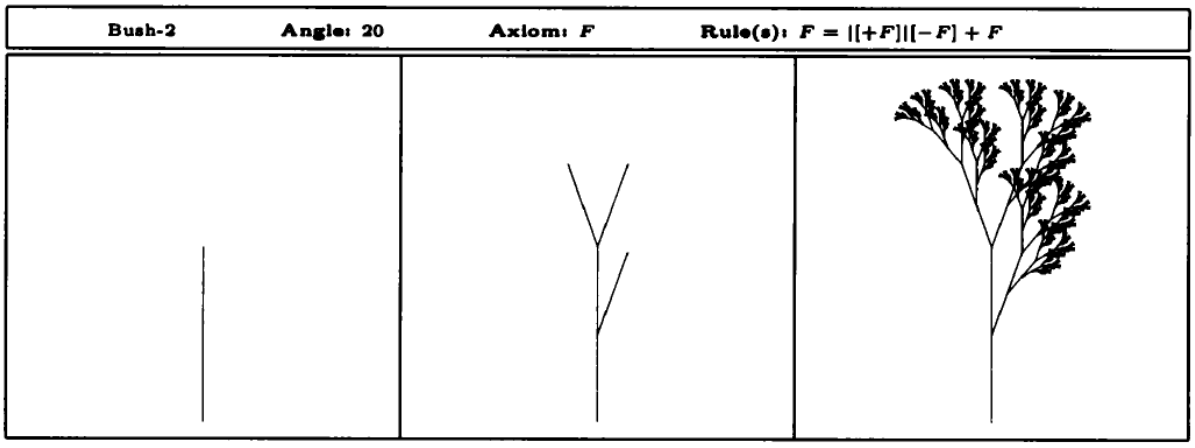

Рисунок 3 Приклад генерації L-System 2 [6]

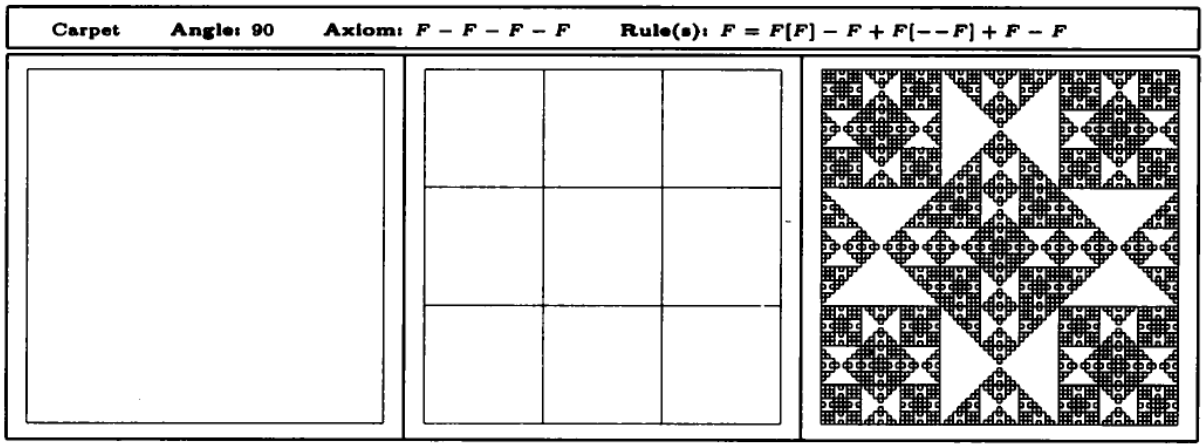

Рисунок 4 Приклад генерації L-System 3 [6]

Як ми бачимо, у цього алгоритму дуже багато можливостей. У першому прикладі ми отримали фрактал схожий на місто, у другому на польову рослину, а в третьому взагалі на килим, тільки змінюючи вхідні дані.

Єдина проблема цього рішення для мого проєкту це занадто помітна симетричність, як для міста. Можна запровадити просте рішення – шанс того, що в процесі виконання алгоритму ми пропустимо точку. Для цього ми просто вводимо ще одну константу у наші вхідні дані. Відповідно алгоритм видасть щось подібне до цього:

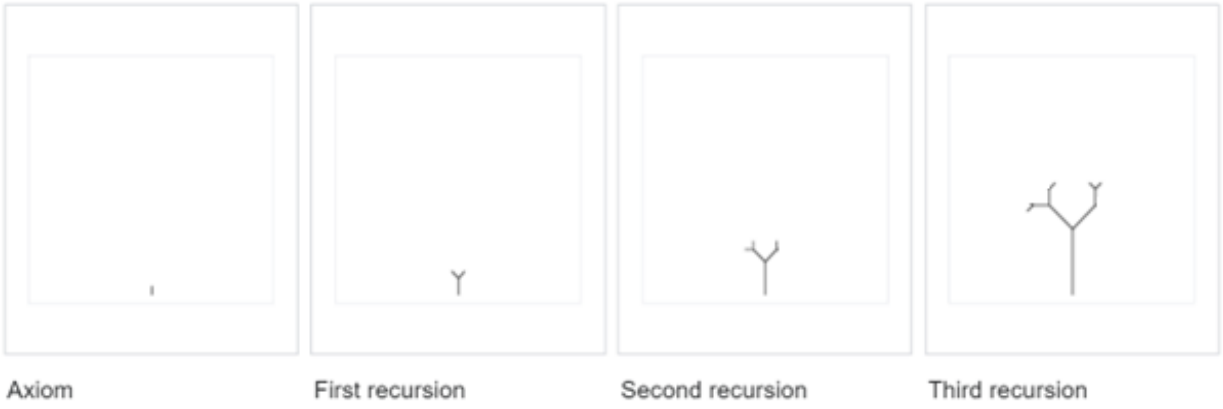

Рисунок 5 Приклад удосконаленого алгоритму

2.3 Результат Імплементації Алгоритму

Отже, ми знаємо як працює алгоритм. Наступною задачою є трансформацію згенерованого фрактала у місто. По-перше, нам потрібно замінити лінії, на дороги. По-друге, додати вздовж доріг якісь будівлі. А також заповнити пусті місця рослинністю для різноміння нашого світу. Імплементувавши це все, ми кожного разу отримаємо унікальне місто з

різним проектуванням. Для наглядного прикладу я взяв вхідні дані, як на рисунку 2, результат вийшов наступний:

Рисунок 6 Приклад результат алгоритму 1

Рисунок 7 Приклад результату алгоритму 2

Рисунок 8 Приклад результату алгоритму 3

Рисунок 9 Приклад результату алгоритму 4

Розділ 3: Pathfinding and Crowd Simulation

3.1 Unity Navigation System

Як я вже зазначав у першому розділі, мета алгоритму пошуку шляху полягає у знаходженні найоптимальнішого шляху між двома точками. Існує дуже багато різних алгоритмів пошуку шляху: всім відомий A*, Minimum Spanning Tree та інші. Але оскільки я роблю проєкт на ігровому двигуні Unity, то він нам пропонує засоби та технології, які можуть полегшити розробку такого алгоритму – Navigation System [2]. Навігаційна система дозволяє розробнику додавати поведінку «розумного» руху його ігровим сутностям по ігровому світу.

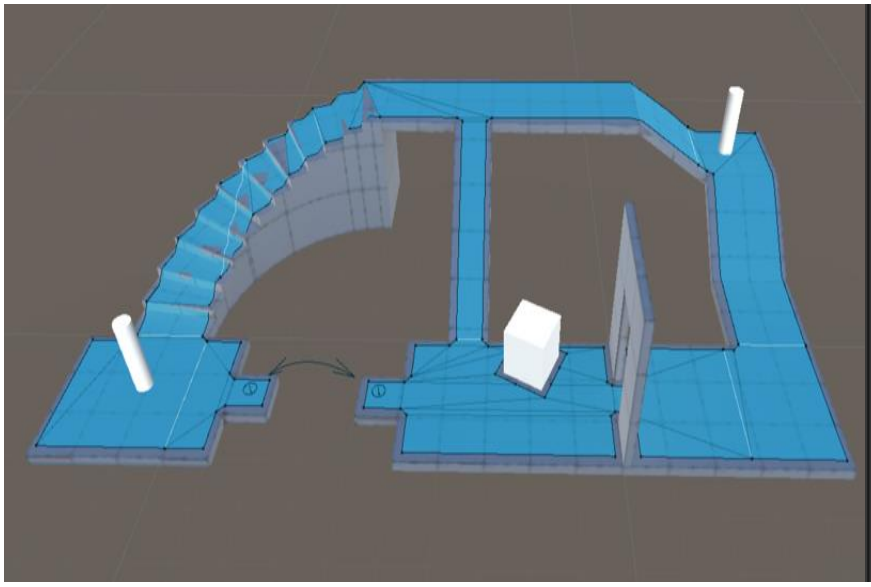

Рисунок 10 Unity Navigation System

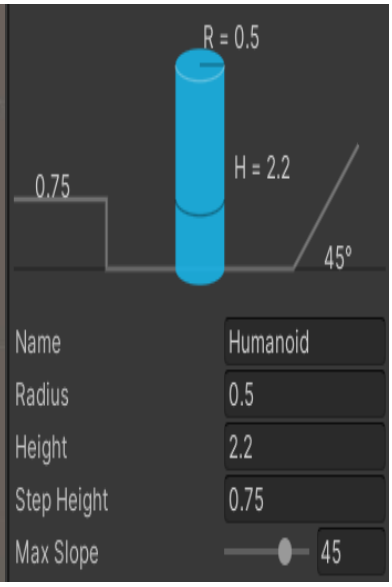

Рисунок 11 Agent Settings

На рисунку 10 зображено те як працює система навігації в Unity. Потрібно обрати що є поверхнею по якій можна ходити, бігати, або стрибати – NavMesh [3], а також обрати поверхню яка є перешкодою. Також потрібно зазначити параметри нашого агента – висоту, його радіус, кут на який він може підійматися та дальність стрибка, що зображено на рисунку 11. Як результат ми отримуємо навігаційну поверхню для нашого агента.

Синім зображена поверхня, по якій може ходити агент, сірим – яка є для нього перешкодою.

Таким чином у нашому місті ми можемо обрати дороги і землю як поверхню, придатну для пересування, а будинки, дерева та інші агенти, як перешкоди.

3.2 Crowd Movement

Наразі ми знаємо як створити навігаційну поверхню для нашого міста, але як заставити наших агентів, в даному випадку ворогів та їхню техніку, пересуватися по ній та робити це кожного разу по різним маршрутам, справляючи враження реальних військових, які йдуть по своїм справам. Для цього ми можемо запровадити Waypoint System – Систему маршрутних точок.

Waypoint System – це якась структура даних з точок які знаходяться на прохідній поверхні. Завдяки достатній кількості точок, можна отримати дуже багато унікальних маршрутів так, щоб у кожного агента був свій. Як працює алгоритм:

1. Навмання обираємо точку з системи маршрутних точок
2. Агент досягає точки
3. Преходимо до першого пункту :)

Так ми отримаємо досить просту, але робочу систему контролю «натопву», який веде себе доволі органічно і реалістично.

3.3 Fleeing Algorithm

Отже, як я і казав, в моїй роботі також знадобиться алгоритм втечі (з *англійської Fleeing Algorithm*) [6]. Потрібен він нам тому що вороги які не потрапили у радіус вибуху, але потрапили у певний трохи більший радіус ударної хвилі будуть тікати від самого вибуху. Насправді його реалізація не

дуже складна, лише потребує трошки розрахунків. Для зручності наведу схему роботи:

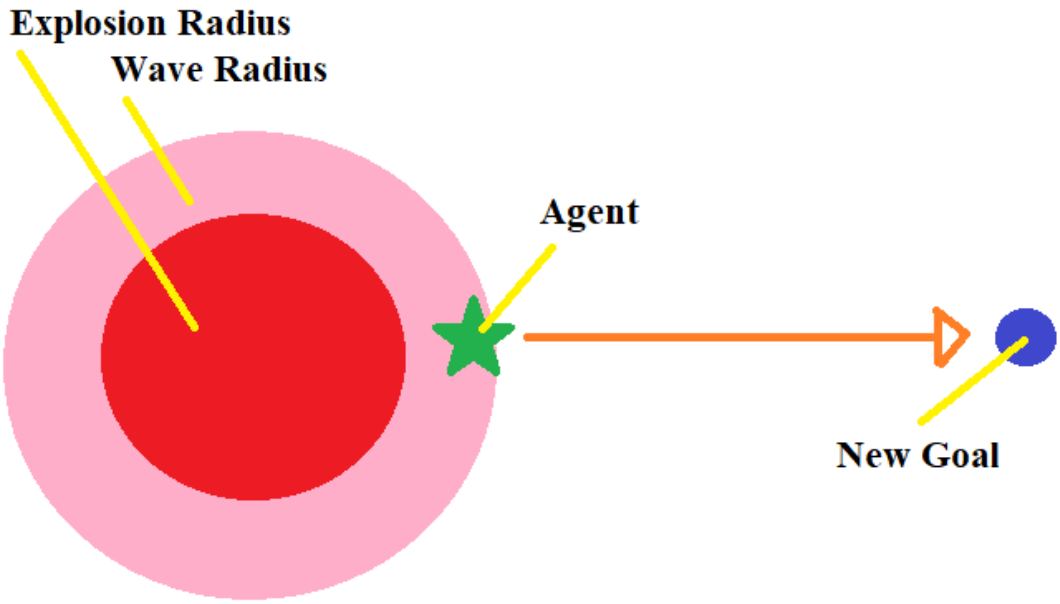

Рисунок 12 Схема роботи Fleeing Algorithm

Якщо агент попадає у більший радіус від вибуха, то треба взяти вектор протилежний вектору до точки вибуха і помножити на відстань яку повинен пробігти втікач – ця точка і стане його теперішньою ціллю на нашому навігаціному покритті. Після того як він її досягне, агент може спокійно повертатися до свого патрулювання по системі його маршрутних точок.

Висновок

Алгоритми, які були розглянуті та реалізовані у моїй роботі, стануть у нагоді як розробникам відеоігор, так і користувачам. Розробники можливо дізнаються щось нове, а користувачі дізнаються як працюють їхні улюблені продукти.

Для того щоб побачити в деталях роботу розглянутих алгоритмів, можна подивитися практичну частину цієї курсової. Робота була написана на ігровому рушії Unity та мові C#.

Список термінів та скорочень

NPC – Non-Player Character – ігрові персонажі, яких не контролює гравець

AI - Artificial Intelligence – штучний інтелект

Data Mining – добування даних

PCG – Procedural Content Generation – процедурна генерація контенту

Pathfinding – пошук шляху з однієї точки до іншої

Crowd Simulation – симуляція натовпу

NavMesh – Navigation Mesh – навігаційна поверхня

Agent – агент, ігрова сутність, яка взаємодіє зі світом

Waypoint System – система маршрутних точок

Fleeing Algorithm – алгоритм втечі

Список використаних джерел

[1] http://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html

[2] https://docs.unity3d.com/Manual/Navigation.html

[3] https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

[4] https://docs.unity3d.com/Manual/com.unity.modules.ai.html

[5] The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation. Gary William Flake

[6] https://learn.unity.com/project/crowd-simulation?uv=2020.2&courseId=5dd851beedbc2a1bf7b72bed