Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка підсистеми ігрового штучного інтелекту

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
Доктор. тех. наук, доцент
Глибовець А.М.
(прізвище та ініціали)

_____
(підпис)
“____” _____ 2020 р.

Виконав студент 1 курсу
Велігурський О.С.
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

ЗМІСТ

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

доц., к.ф.-м.н.

_____С. С. Гороховський

(підпис)

“____”_____2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту___Велігурському Олександру

___1___курсу факультету інформатики

ТЕМА: Розробка підсистеми ігрового штучного інтелекту

Зміст ТЧ до курсової роботи:

Вступ

1. Штучний інтелект у ігровій індустрії
2. Прийняття рішень та скінченні автомати
3. Передбачення подій та системи на основі правил
4. Опис практичної частини роботи

Висновки

Список джерел

Додатки (за необхідністю)

Дата видачі “___” _____ 2019 р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка підсистеми штучного інтелекту

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2019 р.	
2.	Огляд літератури за темою роботи	лютий 2019 р.	
3.	Вивчення матеріалів для створення підсистеми	лютий 2019 р.	
4.	Розробка та реалізація алгоритму роботи підсистеми штучного інтелекту	березень - квітень 2020 р.	
5.	Написання пояснювальної роботи	квітень 2020 р.	
6.	Створення слайдів для доповіді та написання доповіді	травень 2020 р.	
7.	Надання роботи керівнику для перевірки	квітень2020 р.	
8.	Корегування роботи за результатами перевірки керівником	травень 2020 р.	
9.	Остаточне оформлення пояснювальної роботи та слайдів	травень 2020 р.	
10.	Подання роботи на кафедру для перевірки на плагіат	травень 2020 р.	
11.	Захист курсової роботи	травень 2020 р.	

Студент _____**Велігурський О.С.**_____

Керівник _____Глибовець А.М._____

“_____” _____ _____ р.

ВСТУП

Актуальність, наукове та практичне значення обраної теми

З розвитком ігрової індустрії, підсистеми ігрового штучного інтелекту поступово впроваджувалися у різні типи продуктів. Першим поштовхом до ігрового штучного інтелекту у 1951 році стала програма написана Крістофером Стречі для гри у шашки. Наявність навіть простих механізмів поведінки персонажів, керованих комп'ютером, дозволяла зімітувати поведінку проти реального суперника. З серйозним поштовхом в ігровій індустрії у 21 сторіччі, штучний інтелект набув нових технологій та можливостей: навігація у просторі, пошук оптимального шляху та інші.[1]

На даний момент ілюзія розумного супротивника використовується повсюди і є невід'ємною частиною ігрового процесу. Основною цілю ігрового штучного інтелекту є створення видимості інтелектуальної поведінки ігрових об'єктів. Ці системи, в першу чергу, існують для того, щоб надати гравцю цікавий ігровий процес та враження від гри. Головна задача ШІ - не виграти у людини, а гарно йому піддатися.

Мета та завдання курсової роботи

Мета: Створити універсальну підсистему ігрового штучного інтелекту, яка може бути використана для різного типу ігор категорії "Пазли".

Завдання: Розробити алгоритм, який спростить розробку системи для певного розділу ігрового програмного забезпечення. Реалізувати тестову версію гри "Пятнашки" для демонстрації.

Об’єкт дослідження

Розробка допоміжної бібліотеки ІШІ з використанням Unity Engine.

Предмет дослідження

Алгоритми ігрового штучного інтелекту, які вважаються основоположними у ігровій індустрії.

Практичне значення одержаних результатів

Розроблена підсистема може бути впроваджена для створення ігор типу PuzzleGames. Допоможе швидко впровадити базові функції ігрового штучного інтелекту.

Джерела дослідження

Під час написання курсової роботи було досліджено наукові статті про основні задачі ІШІ та особливості розробки подібного функціоналу. Було проаналізовано різні веб-ресурси, які містять інформацію про методики, які можна використати під час розробки. Було проведено аналіз алгоритмів та обрано, той який найкраще підходить для даної підсистеми.

Структура роботи

Курсова робота складається з анотації, вступу, чотирьох основних розділів та підрозділів, висновку, списку використаних джерел та переліку прийнятих скорочень.

Перший розділ курсової роботи присвячений поняттю ігрового штучного інтелекту. Вивчено приклади успішних проектів у розробці підсистем штучного інтелекту.

Другий розділ акцентує увагу на методах прийняття рішень та на системах на основі правил. Описано принципи роботи цих алгоритмів.

У третьому розділі розказується про здатність точно передбачати наступний хід супротивника. Вивчена математична модель кінцевих автоматів, яка дуже часто використовується для створення ігор зі штучним інтелектом.

Четвертий розділ є детальним описом практичної частини роботи. У розділі повно описано, як було обрано середовище розробки, мову програмування. Було проведено аналіз вибору алгоритму за яким буде створюватись застосунок. Міститься детальний опис використаних технологій та процесу розробки з кодом. Додатково надано інструкцію впровадження підсистеми та створено демо гру для демонстрації.

РОЗДІЛ 1

1.1 Штучний інтелект у ігровій індустрії

Основна мета ігрового штучного інтелекту - це зосередитись на тому, які дії повинен виконувати об'єкт, зважаючи на те, в якому стані він зараз знаходиться. Зазвичай це називають управлінням «інтелектуальними агентами», де агент може бути ігровим персонажем, транспортним засобом, ботом, а іноді і чимось більш абстрактним. Агент повинен вміти сприймати своє оточення, приймати рішення на основі стану світу та діяти відповідно до всієї цієї інформації.

Відчуття: агент отримує або знаходить інформацію про об’єкти навколо нього, які можуть впливати на його стан, наприклад: загрози навколо, предмети, події, стан інших агентів або дії гравця.

Мислення: агент приймає рішення, як діяти у тій чи іншій ситуації, залежно від стану інших об’єктів у ігровому світі.

Дії: агент виконує дії для реалізації попереднього рішення. Ситуація міняється через проведені дії, після цього цикл повторюється.

Як правило, основною частиною цього циклу є мислення. Але іграм не потрібна складна система для знаходження інформації, так як основна частина даних вже є невід'ємною частиною самої гри. Немає необхідності запускати алгоритми розпізнавання зображень, щоб визначити, чи є ворог попереду - гра вже знає про це і надає дані прямо в процесі прийняття рішень. [2]

Тренування алгоритму є зайвим, оскільки це методика машинного навчання. Абсолютно не має сенсу розробляти систему тренування, яка буде найскладнішою частиною роботи.

Хороший ІШІ повинен мати такі властивості:

1) Взаємодіє з ігровими системами.

2) Реагує на дії гравця.

3) Передбачуваний.

4) Дає гравцю трохи більше можливостей.

5) Агенти не повинні знаходити найкращий підхід проти людей, але бути реалістичними.

Одним з останніх успіхів отримала система для гри у покер. Перемога у цій грі надзвичайно сильно залежить від удачі та умінь реальної людини. Гравці повинні вміти блефувати та вміти розпізнавати поточний стан суперників.

У 2017 році була опублікована система DeepStack (університет Альберти), яка легко обіграє професіоналів, а також проект Libratus який в 20-денному марафоні виграв віртуальні $ 1,7 млн у чотирьох професійних покеристів в матчах один на один. У липні 2019 Libratus показали найдосконалішу ІШІ систему для покеру. Вона вміє успішно грати (в тому числі з використанням блефу) відразу з п'ятьма опонентами. Щоб стати непереможним, боту довелося розіграти трильйони покерних партій з п'ятьма своїми клонами, які вчили самі себе одночасно.[7]

РОЗДІЛ 2

2.1 Прийняття рішень

Прийняття рішень – це базовий метод, в якому система повинна впливати на об’єкти за допомогою штучного інтелекту. Цей вплив може бути реалізований декількома методами: «звернення об’єктів» або «за заданими параметрами ШІ».

Системи, у яких використовується метод звернення до ігрових об’єктів, підходять більше для простих ігор, де об’єкти звертаються до системи тільки тоді, коли знаходяться у стадії мислення. Також, цей підхід дозволить більш вигідно використовувати переваги багато-потокової архітектури.

При заданих параметрах системи, вона зазвичай ізольована у якості окремого елемента ігрової структури. Цей варіант використовує окремий потік або декілька потоків для обчислення найкращого рішення для заданих значень. При прийнятті рішення аналізується загальний стан гри і надсилається усім агентам. Такий підхід найчастіше застосовується у стратегіях.[3]

Приклад використання умовних операторів:

Гра в якій потрібно ухилятися від куль які прямують на гравця. У персонажа є можливість рухатися ліворуч або праворуч. Це легке завдання для ігрового інтелекту, потрібно лише вирішити в яку сторону рухати персонажа. Найпростіше рішення – розташовувати дійову особу поза траєкторією куль, якщо дистанція до кулі менша за задану. Простий алгоритм реалізації:

```
void Update()
{
    If one of bullets is straight forward to target & straight distance to bullet <
    value
        If bullets exist from left
            Move target right
        Else If bullets exist from right
            Move target left
        Else
            Standby
}
```

«Відчуття» : Гра знає де знаходиться гравець та кулі, знає їх швидкість та траєкторію.

«Мислення» : Персонаж може зробити крок праворуч або ліворуч, або не рухатися

«Дії» : Переміщення персонажа ліворуч, праворуч за заданою швидкістю або стояти на місці.

Тобто, це базовий набір правил, які реагують відносно поточного стану світу.

2.2 Скінченні автомати

Скінченні автомати – модель управління прийняттям рішень для систем із скінченною кількістю станів. Кінцеві автомати розділяються на детерміновані і недетерміновані.

$$M = (V, Q, q_0, F, \delta),$$

V - вхідний алфавіт (кінцева кількість вхідних символів), з якого формуються вхідні слова, які сприймаються кінцевим автоматом;

Q - безліч внутрішніх станів;

q0 - початковий стан

F - безліч заключних, або кінцевих станів

delta – значення функції переходів.

На впорядкованої парі (стан, вхідний символ або порожній) є безліч станів, але кількість скінченна, в які з даного стану можуть перейти до вхідного стану або порожнього ланцюгу (ε). Існують як мінімум два простих способи реалізації кінцевого автомата.

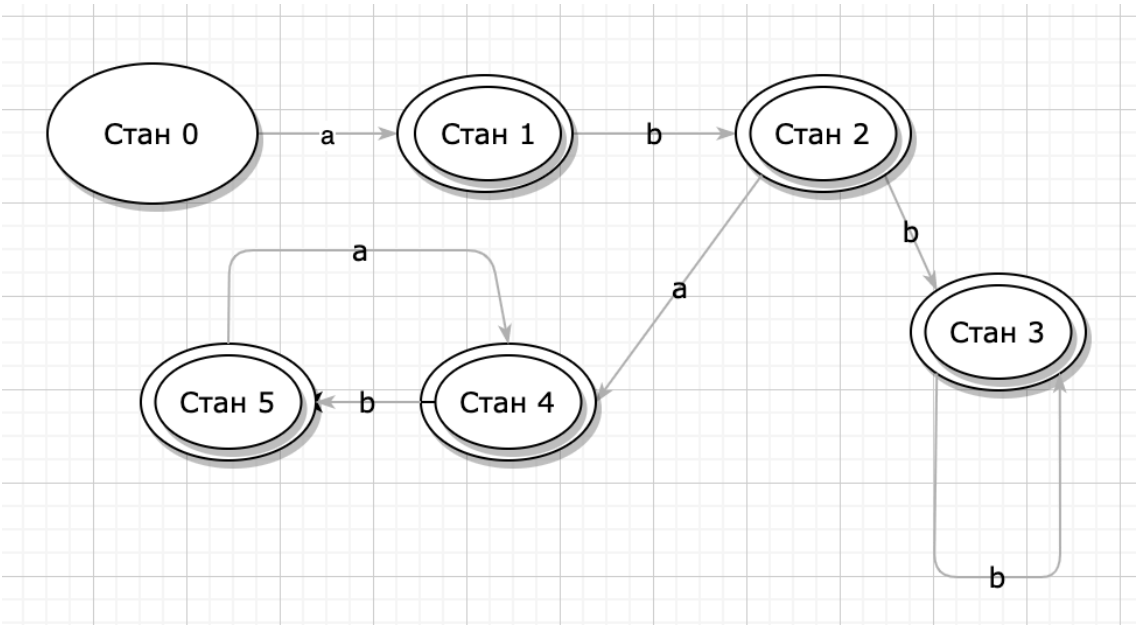
Перший спосіб: Діаграма станів (або граф переходів) - графічне представлення безлічі станів і функції переходів. Являє собою орієнтований граф. Якщо перехід зі стану q1 в q2 може бути здійснений по одному з декількох символів, то всі вони повинні бути над дугою діаграми.

Другий спосіб: Таблиця переходів - табличне представлення функції δ. Зазвичай в такій таблиці кожному рядку відповідає один стан і один допустимий вхідний символ. В осередку на перетині рядка і стовпця записується стан, в який повинен перейти автомат.

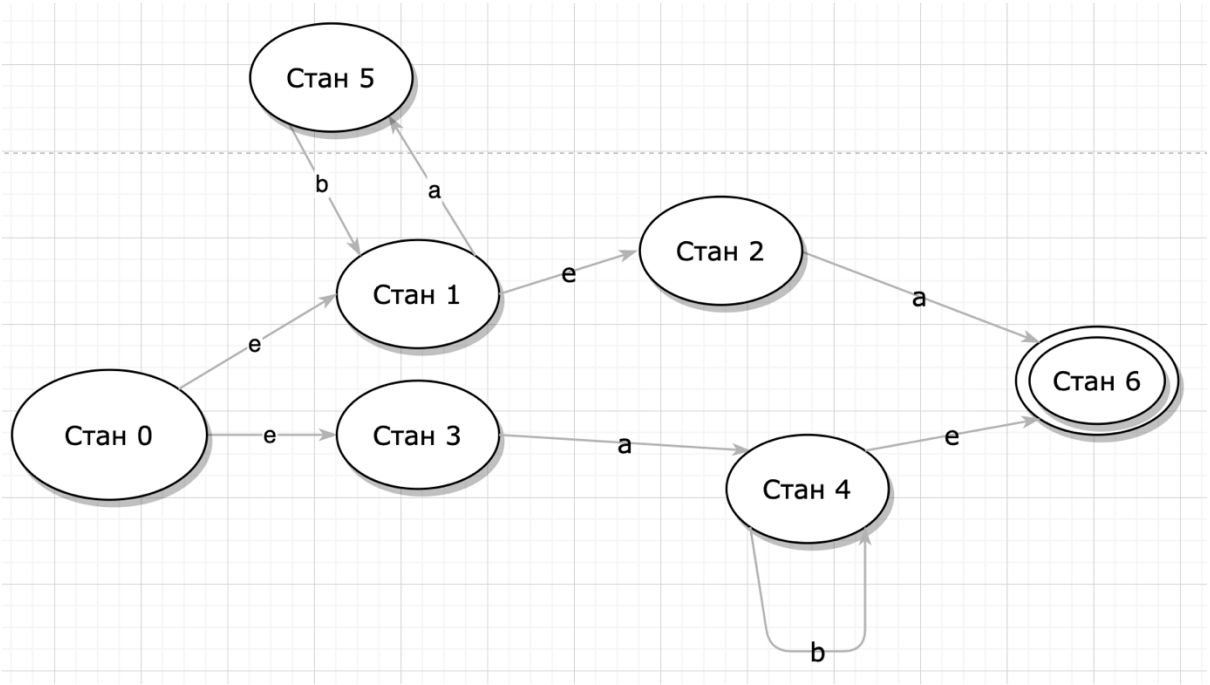
Детермінованим кінцевим автоматом називається такий автомат, в якому немає дуг з міткою ε (не містить жодного символу), і з будь-якого стану, по будь-якому символу можливий перехід не більше, ніж в один стан. [12]

Недетермінований кінцевий автомат є узагальненням детермінованого. Недетермінованість автоматів може досягатися двома способами: або можуть існувати переходи, помічені порожнім ланцюгом ε, або з одного стану можуть виходити кілька переходів, позначених одним і тим же символом. [12]

Детермінований скінченний автомат



Недетермінований скінченний автомат



Теорема стверджує, що для будь-якого кінцевого автомата може бути побудований еквівалентний йому детермінований кінцевий автомат (два

кінцевих автомата називають еквівалентними, якщо їх мови збігаються). [12]

Формалізація — це метод відображення певної області у вигляді формальної системи, коли форма виділяється як особливий предмет дослідження незалежно від змісту. Такий метод дає змогу виявити загальні структури, сформулювати на цій основі загальні закони і правила міркування.

Будь-яка формалізація за визначенням ігнорує деяку частину доступної інформації, тому формалізація — це тільки правильний спосіб з'єднання думок, сам же висновок може зовсім не відповідати дійсності. [13]

РОЗДІЛ 3

3.1 Передбачення подій

Здатність точно передбачати наступний хід супротивника вкрай важлива для адаптивної системи. Для вибору наступного дії можна використовувати різні методи, наприклад розпізнавання закономірностей минулих ходів, випадкові здогадки та інше.

Одним з найпростіших і найпопулярніших способів адаптації системи є відстеження рішень, які були прийняті до теперішнього стану світу. Основна ідея – це оцінювати на скільки успішні або неуспішні кроки були зроблені до цього і на основі цих даних роботи наступний крок.

Наприклад, під таку оцінку можуть потрапити успішні або неуспішні попадання куль, отримані або втрачені можливості. Можна зберігати додаткові відомості про ігровий світ, щоб утворити контекст для вибору підходящих рішень, наприклад рівень здоров'я, колишні дії, позицію персонажа та куль у світі. Також потрібно проводити огляд блоків колишніх дій у цілому, щоб вирішити чи потрібно змінювати тактику проти гравця.

Вивчаючи дії суперника можна зробити висновок та діяти згідно поставлених цілей. Але бувають ситуації коли неможливо передбачити дії гравця точно або взагалі. Тоді, під час рішення визначається функція корисності такого кроку. Якщо рішення може дати кілька можливих результатів з різними можливостями, то система має визначити цінність цих результатів і ймовірності виграшу, потім перемножити відповідні цінності, ймовірності і складність, щоб дати в підсумку математичне сподівання – дію, яка має давати найбільшу очікувану цінність.[5]

У деяких випадках спостерігається парадокс, коли кращий вибір може привести до гіршого рішення або до відмови прийняти рішення.

3.2 Системи на основі правил

Найпростішою формою штучного інтелекту є система на основі правил, де поведінка ігрових об’єктів визначається заздалегідь на основі заданих алгоритмів. Якщо такі об’єкти будуть мати схожу поведінку, але зі своїми унікальними функціями, то результат здаватиметься неявним та неочікуваним, але таку систему не можна цілком віднести до ігрового штучного інтелекту. [6]

Додамо до нашої гри з другого розділу деякі нові функції. Тепер наші кулі будуть трьох різних видів з своїми власними функціями, які різняться між собою. Чорні кулі – летять швидше за інші, зелені кулі – можуть змінити свою позицію з одного ряду до іншого, червоні – можуть ставати невидимими на деякий час. Тобто кожна куля має свій власний набір правил, за якими діє. Якби кулі, які з’являються, були лише одного типу, тоді їх поведінка була зрозумілою і гравець зміг би передбачити і продовжувати гру нескінченно. Але оскільки кулі заявляються випадковим чином і одразу різних типів, то їх рухи здаються складнішими і більш непередбачуваними.

З цього прикладу випливає, що правила не обов'язково повинні бути жорстко заданими. Такі змінні, як швидкість, зміна шляху, унікальні можливості дозволяють отримати більш різноманітну поведінку об'єктів навіть при використанні систем на основі правил. У більш складних і розумних системах в якості основи використовуються низки умовних правил. Системи на основі правил є фундаментальною основою ігрового штучного інтелекту.

РОЗДІЛ 4

4.1 Вибір засобів розробки підсистеми

В якості середовища розробки підсистеми ігрового штучного інтелекту було використано Unity3D та мову програмування C#. Для роботи підсистеми було використано такі бібліотеки: System; System.Collections; System.Collections.Generic; System.IO; System.Linq; UnityEngine.Events; UnityEngine.UI;

Unity3D – кросс-платформне середовище розробки застосунків. Unity дозволяє створювати додатки, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої та інше. Підтримується операційні системи: Microsoft Windows, MacOS, Linux.

Для перевірки роботи системи було прийнято рішення розробити тестову версію гри “П’ятнашки” - популярна головоломка, придумана в 1878 році Ноєм Чепмен. Являє собою набір однакових квадратних кісточок з нанесеними числами. Довжина сторони коробки в чотири рази більше довжини сторони кісточок для набору з 15 елементів, відповідно в коробці залишається незаповненим одне квадратне поле. Мета гри - переміщуючи кісточки по коробці, домогтися упорядкування їх за номерами, бажано зробивши якомога менше рухів.[8] Гра може мати початкові варіанти такі, що їх рішення не існує, тому необхідно було передбачити створення таких початкових станів, які б задовольняли умовам проходження.

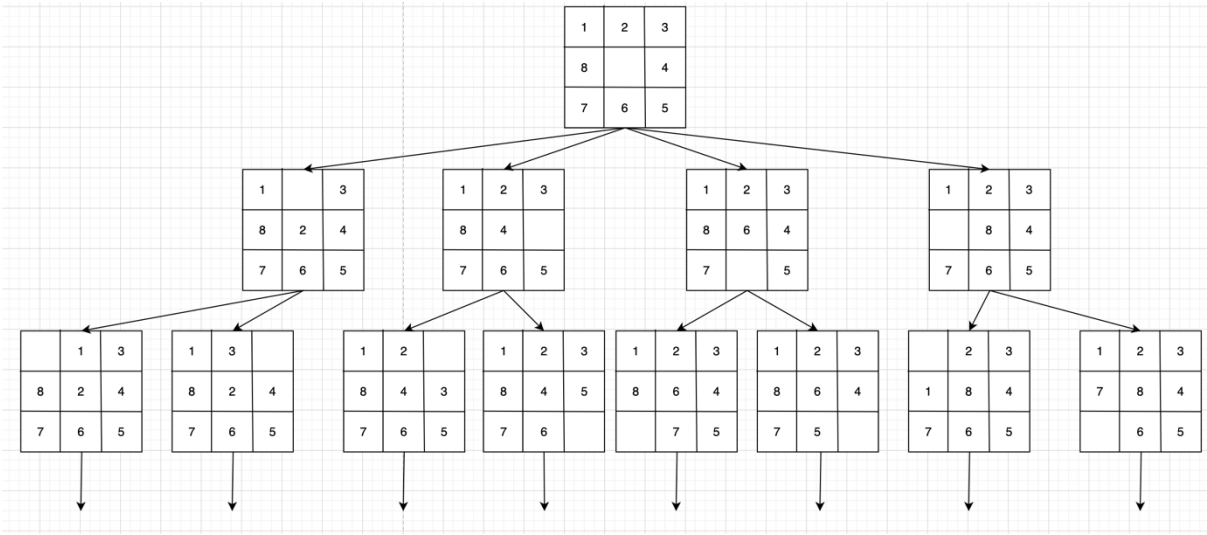
4.2 Алгоритм та структура програми (опис файлів)

Оскільки алгоритм ігрового штучного інтелекту повинен підтримувати різні види ігор, він не може бути створений спеціально для якоїсь одної гри. AI також повинен бути постійним процесом, який може

адаптуватися до поточного стану системи, а не до результатів попередніх станів. З цієї причини методи відкритого циклу часто є найбільш ефективними для вирішення такої задачі.

Популярним методом розробки ігрового штучного інтелекту є алгоритм пошуку по дереву Монте-Карло (MCTS). Разом із ним використовується метод UCT. AI моделює потенційні дії та впорядковує їх за певними ознаками, виходячи із середньої найвищої винагороди проходження кожного шляху. [9]

Як правило, для вирішення будь-якої задачі на основі графу може використовуватися будь-який алгоритм накладання маршрутів, але для того, щоб будь-яка гра такого типу могла бути вирішена за допомогою незмінного алгоритму, буде використано саме MCTS. Дошка 3 на 3 має 181440 варіантів різних рішень, що вже є нелегкою задачею.



UCT - популярний алгоритм, який допомагає з пошуком для методу Монте-Карло і не тільки. Якщо по простому, то цей алгоритм допомагає обрати не кращий рух на даний момент, а рух який призведе до кращого

результату у майбутньому. Підрахунок цих границь відбувається за формулою –

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$ де :

W(i) - кількість виграшних ситуацій для вузла, розглянутого після і-го переміщення

n(i) - кількість проходжень через вузол, що розглядається після і-го переміщення

N(i) - загальна кількість проходжень після і-го переміщення, виконаного початковим вузлом

c – константне значення, яке обирається емпіричним шляхом

(теоретично - c = √2)

Основна увага MCTS приділяється аналізу найбільш перспективних кроків, розширенню дерева пошуку на основі випадкової вибірки пошукового простору. Кожна ітерація алгоритму складеться з 4 кроків:

Selection: Починаємо з кореневого вузла та послідовно вибираємо дочірні вузли, поки не буде досягнутий листовий вузол. Початковий вузол - це поточний ігровий стан, а листовий вузол - це будь-який вузол, з якого ще не розпочато моделювання.

Expansion: якщо листовий вузол не завершить гру з якимось результатом, створюється ще один дочірній вузол. Дочірні вузли - це будь-які можливі рухи від теперішнього ігрового стану, визначені листовим вузлом.

Simluation: завершує роботу випадкового дочірнього вузла.

Backpropagation: оновлює інформацію у пройдених вузлах на основі даного стану.

Раунди пошуку повторюються до тих пір, поки залишається час або кількість ітерацій, які відведені на прохід. В кожному вузлі обирається хід, значення UCT якого – найвище. Пошук за Монте-Карло має значні переваги порівняно з іншими алгоритмами, які мінімізують простір пошуку. Зокрема, алгоритм не потребує явної оцінки ходу. Оскільки метод проходить лише по найбільш перспективних розгалуженнях, то дерево росте асиметрично, що поліпшує швидкість проходження. Більш того, метод може бути зупинений у будь-який час, що іноді дозволить видати дуже несподіваний крок.

Недоліком такого підходу є те, що в критичній позиції алгоритм може не врахувати можливі варіанти, через використання випадковості. По суті, пошук намагається обрізати послідовності, які є вже не актуальними. У деяких випадках це може призвести до дуже специфічного стану, який є суттєвим, але який ігнорується при обрізанні дерева. [11]

```
public class MonteCarloTreeSearch
{
   public Node SearchForCurrentState(State rootState, int iterations)
   {
      Node rootNode = new Node(0, null, rootState);
      for (int i = 0; i < iterations; i++)
      {
         Node newNode = rootNode;
         var state = new State(rootState.Size);
         newNode = SelectNextNode(newNode, state);
         newNode = ExpandChildNodes(newNode, state);

         Rollout(newNode,state);
         Backpropagate(newNode);
      }
```

```
        var selectedNode = rootNode.ChildNodes.OrderByDescending(c => c.Visits).FirstOrDefault();
        return selectedNode;
    }
    private static Node SelectNextNode(Node node, State state)
    {
        while (node.AvailableMoves.Count == 0 && node.ChildNodes.Any())
        {
            node = node.Select_UCT_Child();
            state.PerformAction(node.Move);
        }
        return node;
    }

    private static Node ExpandChildNodes(Node node, State state)
    {
        if (node.AvailableMoves.Any())
        {
            var m = node.AvailableMoves.OrderBy(x => Guid.NewGuid()).FirstOrDefault();
            state.PerformAction(m);
            node = node.AddChild(m, state);
        }
        return node;
    }

    private static void Rollout(Node node, State state)
    {
        int zero = node.State.FindEmptyIndex();
        int[] neighbours = Neighbours.Instance.GetNeighbors(zero);
        foreach (var value in neighbours)
        {
            var temp = neighbours;
            if (temp.Length == 0)
            {
```

```
            break;
        }
        state.PerformAction(state.Array.OrderBy(x => Guid.NewGuid()).FirstOrDefault());
        temp.Where(val => val != value).ToArray();
      }
    }
    private static void Backpropagate(Node node){
      while (node != null)
      {
        var result = 1;
        node.UpdateNode(result);
        node = node.ParentNode;
      }
    }
}
```

Клас «Node» повинен мати посилання на його батька (та / або дітей) для обходу дерева. Конструктор вузла повинен мати можливість отримати екземпляр шуканого стану. Присутні методи для додавання нової ноди до списку, оновлення ноди, підрахунок та вибір ноди верхньої межі.

```
public class Node
{
    public State State {get;}
    public Node ParentNode { get; set; }
    public int Move { get; set; }
    public int Wins { get; set; }
    public int Visits { get; set; }
    public List<Node> ChildNodes { get; set; }
    public List<int> AvailableMoves { get; set; }

    private const double ChoosenConstant = 2.0;
    public Node(int move, Node parent, State state)
    {
        ParentNode = parent;
        State = state;
        ChildNodes = new List<Node>();
        Wins = 0;
        Visits = 0;
```

```
        AvailableMoves = state.Array.ToList();
    }
```

Додавання нової ноди

```
    public Node AddChild(int move, State state)
    {
        var n = new Node(move, this, state);
        AvailableMoves.Remove(move);
        ChildNodes.Add(n);

        return n;
    }
```

Вибір ноди верхньої межі

```
    public NodeSelectUCTChild()
    {
        var sorted = ChildNodes.OrderByDescending(CalculateUCT);
        return sorted.FirstOrDefault();
    }
```

Підрахунок верхньої межі

```
    private double CalculateUCT(Node node)
    {
        var result = (double)node.Wins / node.Visits + Math.Sqrt(ChoosenConstant *
Math.Log(Visits) / node.Visits);

        return result;
    }
```

Оновлення відвідувань

```
    public void UpdateNode(int value)
    {
        Visits += 1;
        Wins += value;
    }
}
```

Клас стану системи. Унаслідується від базового стану. Має масив значень та розмір дошки. Включає у себе методи для роботи з грою, а саме: знаходження пустого індексу, роботу з масивом, перевірку на правильність кінцевого результату, дії при виборі даного стану.

```
public class State: IGameState
{
    public int[] Array
    {
        get;
    }
    public int Size{ get; set;}

    private int _emptyIndex;

    public int GetEmptyTileIndex()
    {
        return _emptyIndex;
    }

    public State(int size)
    {
        Size = size;
        Array = new int[Size * Size];
        for (int i = 0; i < Array.Length; ++i)
        {
            Array[i] = i;
        }

        _emptyIndex = Array.Length - 1;
    }

    public State(int[] arr)
    {
        Size = (int)System.Math.Sqrt(arr.Length);

        Array = new int[Size * Size];
        for (int i = 0; i < Array.Length; ++i)
        {
            Array[i] = arr[i];

            if (arr[i] == (Array.Length - 1))
            {
                _emptyIndex = i;
            }
        }
    }

    public State(State other)
    {
        Size = other.Size;
        _emptyIndex = other._emptyIndex;
        Array = new int[Size * Size];
        other.Array.CopyTo(Array, 0);
    }
```

```
    public static bool Equals(IGameState a, IGameState b)
    {
        for (int i = 0; i < a.Array.Length; i++)
        {
            if (a.Array[i] != b.Array[i])
            {
                return false;
            }
        }
        return true;
    }

    public int FindEmptyIndex()
    {
        for (int i = 0; i < Array.Length; i++)
        {
            if (Array[i] == Array.Length - 1)
            {
                return i;
            }
        }

        return Array.Length;
    }

    public void PerformAction(int index)
    {
        int tmp = Array[index];
        Array[index] = Array[_emptyIndex];
        Array[_emptyIndex] = tmp;
        _emptyIndex = index;
    }
}
```

Клас для генерації випадкових послідовностей та перемішування елементів масиву.

```
static class RandomExtensions
{

    public static void Shuffle<T>(this System.Random rng, T[] array)
    {
        int n = array.Length;
        while (n > 1)
        {
            int k = rng.Next(n--);
            T temp = array[n];
            array[n] = array[k];
            array[k] = temp;
```

```
        }
    }

    public void Randomize(int Size)
    {
        new System.Random(Size* Size + 1).Shuffle(Array);

        for (int i = 0; i < Array.Length; i++)
        {
            if (Array[i] == (Array.Length - 1)) _emptyTileIndex = i;
        }
    }

}
```

Клас Utils із допоміжними властивостями, наприклад:

1)Створення нового спрайту за допомогою текстури (може бути використано наприклад для пазлів)

2)Завантажити текстуру з папки проекта

3)Завантажити текстуру з файлу

4)Отримати впорядкований список цифр в залежності від розміру дошки

```
public class Utils
{
    public static Sprite LoadNewSprite
            (
            Texture2D SpriteTexture,
            int x,
            int y,
            int w,
            int h ,
            float PixelsPerUnit = 100.0f,
            SpriteMeshType spriteType = SpriteMeshType.Tight
            ){
        Sprite NewSprite = Sprite.Create(
SpriteTexture, new Rect(x, y, w, h), new Vector2(0,   0), PixelsPerUnit, 0, spriteType);
        return NewSprite;
    }
    public static Texture2D LoadTexture(string resourcePath)
    {
        Texture2D tex = Resources.Load<Texture2D>(resourcePath);
        return tex;
    }
```

```
    public static Texture2D LoadTextureFromFile(string FilePath)
    {
        Texture2D Tex2D;
        byte[] FileData;

        if (File.Exists(FilePath))
        {
            FileData = File.ReadAllBytes(FilePath);
            Tex2D = new Texture2D(128, 128);
            if (Tex2D.LoadImage(FileData))
            {
                return Tex2D;
            }
        }
        return null;
    }


    public static int[] GetTileNumbers(int rows, int cols,float tileWidth, float tileHeight)
    {
        int[] indexes = new int[rows * cols];

        float x, y;
        for (int i = 0; i < cols; i++)
        {
            y = (rows - 1 - i) * tileHeight;
            for (int j = 0; j < rows; j++)
            {
                x = j * tileWidth;
                int index = i * cols + j;
                indexes[index] = index;
            }
        }
        return indexes;
    }
}
```

Створення дошки для гри Пятнашки, а саме: створення об’єктів для формування гри, задання їм станів та відповідних нумерацій.

```
public class PuzzleBoard: MonoBehaviour
{
    private Dictionary<int, Vector3> PositionsList = new Dictionary<int, Vector3>();

    public float TileWidth{ get; }
    public float TileHeight { get;}

    public List<GameObject> Tiles;
```

```
    public PuzzleBoard(float w, float h, int boardSize,GameObject prefab, Transform parentToSpawnAt)
    {
        Tiles = new List<GameObject>();
        for (int i = 0; i < boardSize * boardSize; i++)
        {
            var tile = Instantiate(prefab, parentToSpawnAt, false);
            Tiles.Add(tile);
        }

        TileWidth = w;
        TileHeight = h;

        float x, y;
        float startx = -TileWidth * boardSize / 1.5f;
        float starty = TileHeight * boardSize / 1.5f - TileHeight;

        for (int i = 0; i < boardSize; i++)
        {
            y = -i * TileHeight;
            for (int j = 0; j < boardSize; j++)
            {
                x = j * TileWidth;
                int index = i * boardSize + j;
                PositionsList.Add(index, new Vector3(startx + x, y + starty, 0.0f));
                Tiles[index].name = index.ToString();
                Tiles[index].GetComponent<RectTransform>().sizeDelta =
new Vector2(TileWidth, TileHeight);
            }
        }
    }

    public void SetState(State state)
    {
        for (int i = 0; i < state.Array.Length; i++)
        {
            GameObject obj = Tiles[state.Array[i]];
            obj.transform.localPosition = PositionsList[i];
        }
    }

    public void SetNumbers(int[] values, Action<GameObject> OnItemClickAction)
    {
        for (int i = 0; i < Tiles.Count; i++)
        {
            var ItemText = Tiles[i].GetComponentInChildren<TextMeshProUGUI>();

            if(values[i] == -1)
            {
                ItemText.text = "";
```

```
            Tiles[i].GetComponentsInChildren<Image>()[1].color = new Color32(71, 71, 90, 255);
            }
            else
            {
               ItemText.text = (values[i]+1).ToString();

            }
            var ItemButton = Tiles[i].GetComponent<Button>();
            ItemButton.onClick.RemoveAllListeners();

            ItemButton.onClick.AddListener(() =>
            {
               OnItemClickAction?.Invoke(ItemButton.gameObject);
            });
         }
      }
   }
```

Методи для створення випадкової послідовності, яка буде точно вирішена

```
private void RandomState()
   {
      int[] arr = new int[PuzzleSize * PuzzleSize];
      for (int i = 0; i < PuzzleSize * PuzzleSize; i++)
      {
         arr[i] = i;
      }
      Current_State = new State(arr);
      Finish_State = new State(arr);
      isSolved = false;
      solving_Now = false;
      int depth = UnityEngine.Random.Range(50, 100);
      StartCoroutine(CreateRandomSolvableState(depth));
      randomizing_Now = true;
   }

   private IEnumerator CreateRandomSolvableState(int depth)
   {
      randomizing_Now = true;

      for (int i = 0; i < depth; ++i)
      {
         int zero = Current_State.FindEmptyIndex();
         int[] neighbours = Neighbours.Instance.GetNeighbors(zero);

         int index = UnityEngine.Random.Range(0, neighbours.Length);
         SwapTiles(neighbours[index], Current_State, false);
         yield return new WaitForEndOfFrame();
      }
```

```
        isSolved = false;
        randomizing_Now = false;
    }
```

Плавний рух клітинок та метод заміни клітинок місцями

```
private IEnumerator MoveOverSeconds(GameObject objectToMove, Vector3 end, float seconds)
    {
        float elapsedTime = 0;
        Vector3 startingPos = objectToMove.transform.localPosition;
        while (elapsedTime < seconds)
        {
            objectToMove.transform.localPosition = Vector3.Lerp(startingPos, end, (elapsedTime / seconds));
            elapsedTime += Time.deltaTime;
            yield return new WaitForEndOfFrame();
        }
        objectToMove.transform.localPosition = end;
    }

    private State SwapTiles(int first, State puzzle, bool useCoroutine)
    {
        int second = puzzle.GetEmptyTileIndex();

        Vector3 newPos = puzzleBoard.Tiles[puzzle.Array[second]].transform.localPosition;
        newPos.z = 1.0f;
        Vector3 newEmptyPos = puzzleBoard.Tiles[puzzle.Array[first]].gameObject.transform.localPosition;
        newEmptyPos.z = 0.0f;
        puzzleBoard.Tiles[puzzle.Array[second]].transform.localPosition = newEmptyPos;

        if (useCoroutine)
        {
            StartCoroutine(MoveOverSeconds(puzzleBoard.Tiles[puzzle.Array[first]], newPos, 0.2f));
        }
        else
        {
            puzzleBoard.Tiles[puzzle.Array[first]].transform.localPosition = newPos;
        }
        puzzle.SwapWithEmpty(first);
        return puzzle;
    }
```

Методи для обробки кліків

```
private void TileClickHandler(GameObject clickedTile)
    {
        int zero = Current_State.GetEmptyTileIndex();
        int[] neighbours = Neighbours.Instance.GetNeighbors(zero);
```

```
    for (int i = 0; i < neighbours.Length; ++i)
    {
        if (clickedTile.name == Current_State.Array[neighbours[i]].ToString())
        {
            SwapTiles(neighbours[i], Current_State, true);
            isSolved = false;
        }
    }
}

private void ShuffleHandler()
{
    RandomState();
}

private void SolveHandler()
{
    if (!isSolved && !solving_Now)
    {
        SolvePuzzle();
    }
}

private void SetButtonsHandlers()
{
    Shuffle_Button.onClick.AddListener(ShuffleHandler);
    Solve_Button.onClick.AddListener(SolveHandler);
}
```

Методи початку, завершення та вирішення гри

```
private IEnumerator Monte_Carlo(State start_state, State finish_state, Action<int> OnSolutionFoundAction)
{
    MonteCarloTreeSearch monteCarlo = new MonteCarloTreeSearch();

    while (!isSolved)
    {
        steps++;
        var newCurrentNode = monteCarlo.SearchForCurrentState(start_state, 1000);

        if (State.Equals(newCurrentNode.State, finish_state))
        {
            Debug.Log("States.Equals");
            Debug.Log(string.Join(", ", newCurrentNode.State.Array));
            Debug.Log(string.Join(", ", finish_state.Array));
            OnSolutionFoundAction?.Invoke(steps);
            break;
        }
```

```
            puzzleBoard.SetState(newCurrentNode.State);

            yield return new WaitForEndOfFrame();
        }
        puzzleBoard.SetState(finish_state);
    }

    private void OnSolutionFoundAction(int stepsCount)
    {
        _stepsCountText.text = $"Рахунок: {stepsCount}";
        Debug.Log("Solution found.." + "Total moves needed = " + stepsCount);

        isSolved = true;
        solving_Now = false;
    }

    private void SolvePuzzle()
    {
        solving_Now = true;
        StartCoroutine(Monte_Carlo(Current_State, Finish_State, OnSolutionFoundAction));
    }

    private IEnumerator ShowSolved()
    {
        WinGameObject.SetActive(true);
        Debug.Log("SOLVED => RELOAD");
        Solve_Button.interactable = true;
        yield return new WaitForSeconds(3);
        LoadAPuzzle();
        steps = 0;
    }
```

Методи початку роботи гри, а саме дії, які виконаються перед запуском, на початку запуску та дії які будуть виконуватися під час кожного кадру:

```
private void Awake()
    {
        SetButtonsHandlers();
        int array_size = PuzzleSize * PuzzleSize;
        int[] arr = new int[array_size];
        for (int i = 0; i < array_size; i++)
        {
            arr[i] = i;
        }
        Current_State = new State(arr);
        Finish_State = new State(arr);
        WinGameObject.SetActive(false);
        Neighbours.Instance.CreateGraph(PuzzleSize);
    }
```

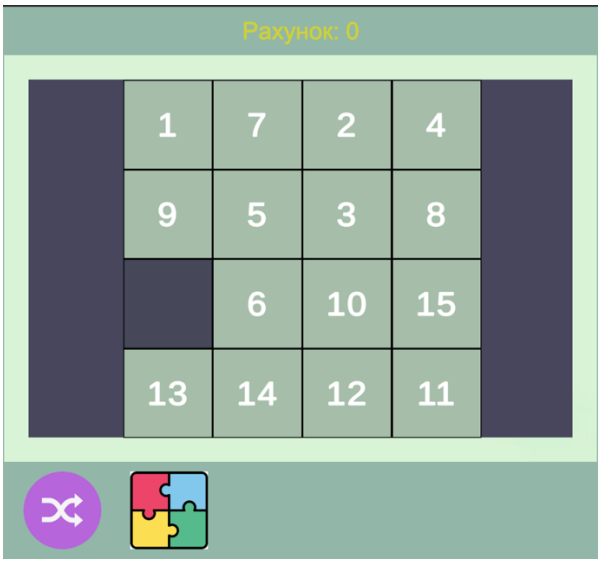
```
    private void Start()
    {
        if (loading_Puzzle_Now)
        {
            LoadAPuzzle();
        }
    }

    void Update()
    {
        if (!isSolved && !randomizing_Now)
        {
            isSolved = State.Equals(Current_State, Finish_State);
            if (isSolved)
            {
                StartCoroutine(ShowSolved());
            }
        }
    }

    private void LoadPuzzle()
    {
        if (puzzleBoard != null)
        {
            for (int i = 0; i < puzzleBoard.Tiles.Count; i++)
            {
                Destroy(puzzleBoard.Tiles[i].gameObject);
            }
            puzzleBoard.Tiles.Clear();
        }
        Tile_Size = ParentTransform.rect.height / PuzzleSize;

        puzzleBoard = new PuzzleBoard(Tile_Size, Tile_Size, PuzzleSize,
ParentTransform.rect.height, ItemPrefab,ParentTransform.transform);

        int[] values = Utils.GetTileNumbers(PuzzleSize, PuzzleSize, Tile_Size, Tile_Size);
        int empty_value = -1;
        values[values.Length - 1] = empty_value;
        puzzleBoard.SetNumbers(values, TileClickHandler);
        puzzleBoard.SetState(Current_State);
    }
```

4.3 Тестування програми та результати її виконання

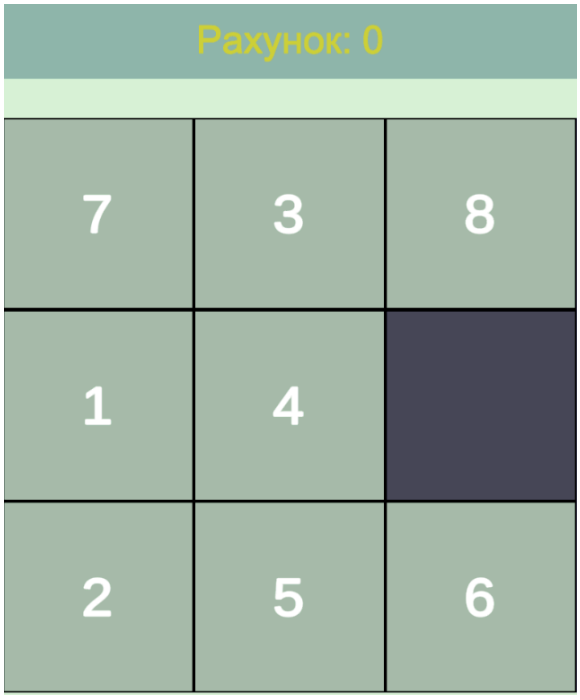
Гра має дуже простий інтерфейс, який дозволяє керувати можливостями застосунку. Також передбачена можливість самостійного проходження гри.

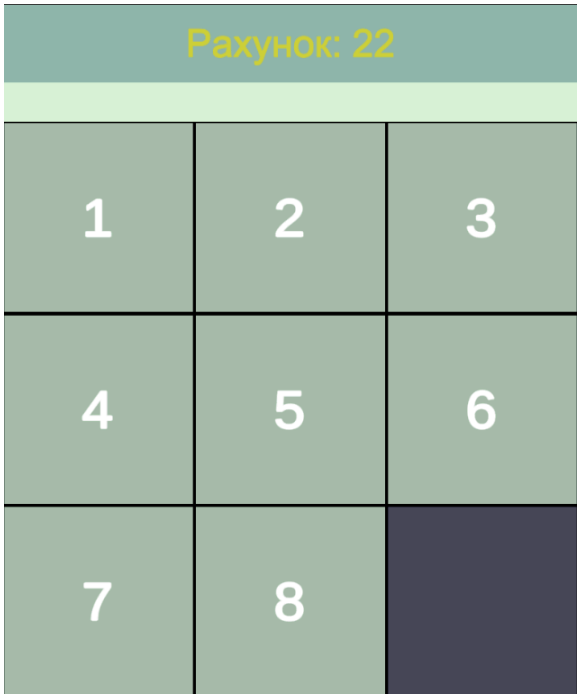
Інтерфейс складається з ігрової дошки, тексту з кількістю зроблених кроків, панелі виграшу та допоміжних кнопок:

1) Кнопка перемішування – випадковим чином розставляє клітинки на дошці
2) Кнопка розв’язку – проводить аналіз теперішнього стану дошки та шукає найкоротший шлях до вирішення. Після виконання розблоковує кнопки переміщення між ходами.

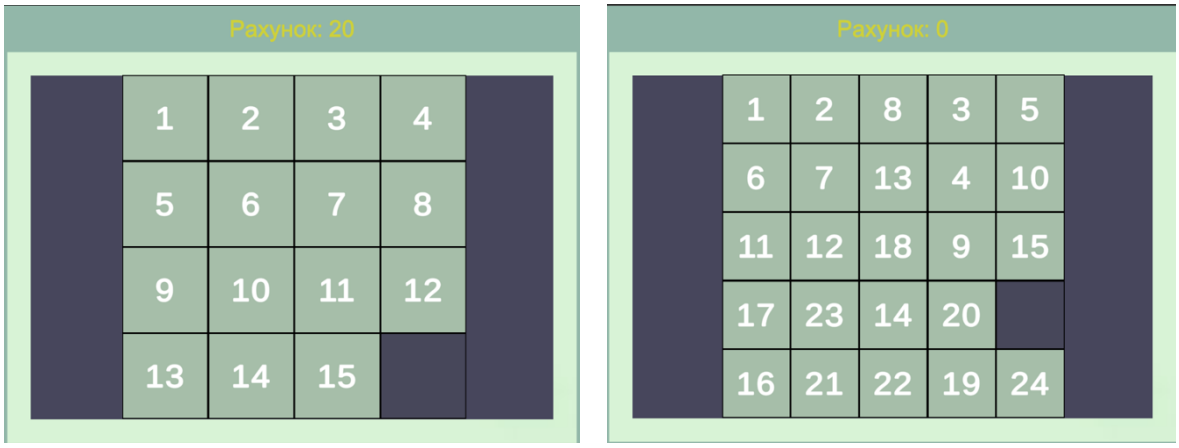


Стан до та після автоматичного вирішення гри

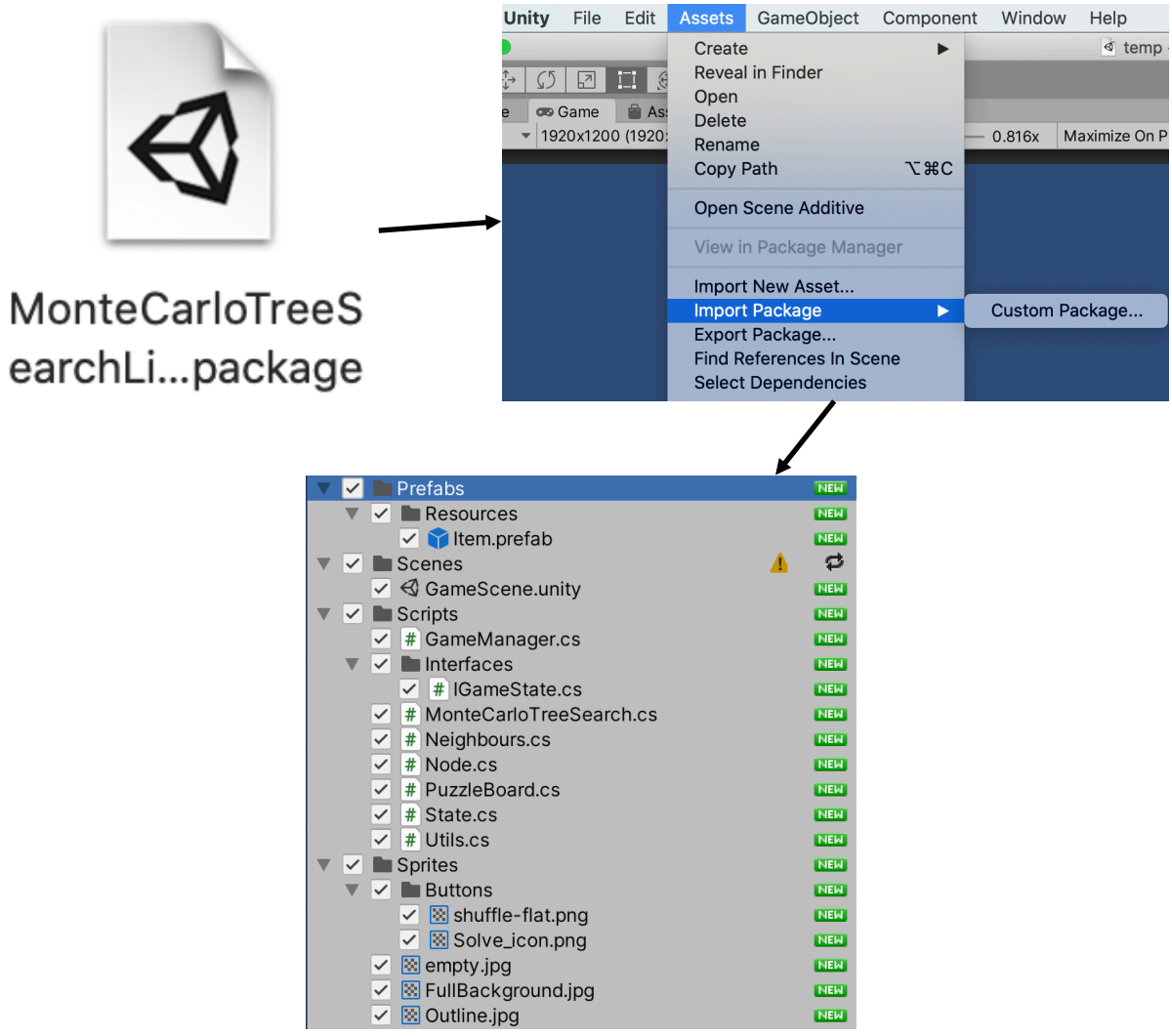




Розмірність дошки та кількість ітерацій для проходження можуть легко та швидко задаватися у налаштуваннях.



Створена підсистема штучного інтелекту дозволяє швидко впровадити методи для вирішення ігор типу “Пазлові ігри”. Для цього необхідно завантажити створену бібліотеку та імпортувати її у проект.



Наслідувати клас GameState від IGameState, перезаписати необхідні методи для створюваної гри, звернутися до класу MonteCarloTreeSearch та визвати метод SearchForCurrentState(State rootState, int iterations), де rootState – це початковий стан системи на даний момент, iterations – кількість кроків, під час яких буде проводитися пошук варіантів. При закінченні, алгоритм віддасть новий стан системи.

ВИСНОВКИ

Результатом курсової роботи стала підсистема ігрового штучного інтелекту, основною ціллю якої є спрощення розробки ігор підрозділу PuzzleGames. Було оглянуто базові методи створення ігрового штучного інтелекту та розглянуто вже працюючі системи штучного інтелекту.

У першому розділі було детально розглянуто аспекти ігрового штучного інтелекту. Відчуття, мислення, дії – три основні принципи роботи ІШІ.

У другому розділі була висунута інформація про системи прийняття рішень, роботу скінченних автоматів та формалізацію. Були виокремлені види скінчених автоматів та місця застосування цих правил.

Третій розділ розповідає про важливість передбачення подій у системах з використанням штучного інтелекту та про системи на основі правил. Це є невід'ємною часткою про розробці застосунків, де застосовується технологія розумної поведінки.

У четвертому розділі курсової роботи детально описано та обґрунтовано вибір використаних технологій та процес розробки підсистеми. Також було описано можливості та аспекти обраного алгоритму. Було надано інформацію про те, як впровадити бібліотеку у проект та приклад використання. Додатково було створено демо версію гри «Пятнашки», для тестування роботи системи. Гра має прості налаштування та засоби для їх редагування.

В перспективі, розроблена підсистема ІШІ спростить розробку ігор визначеної категорії та дозволить швидко впровадити базові методи для роботи з системою штучного інтелекту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. [Електронний ресурс] Wikipedia: Artificial intelligence https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games
2. [Електронний ресурс] Habr: Як створити ігровий штучний інтелект https://habr.com/ru/company/pixonic/blog/428892/
3. Вільямс І.Д., “ Штучний інтелект у іграх ”, (2007)
4. Ian Millington, “ Artificial intelligence in games”, -2d ed.
5. Володимир В’югін - “Математичні основи машинного навчання та прогнозування” (2014)
6. Воскобойніков Сергій, Стаття “Інтелектуальні системи в комп’ютерних іграх”
7. [Електронний ресурс] CurrentTime: Де ШІ вже отримав перемогу над людиною https://www.currenttime.tv/a/ai-games/30242784.html
8. [Електронний ресурс] Wikipedia: 15 puzzle https://en.wikipedia.org/wiki/15_puzzle
9. [Електронний ресурс] Wikipedia: general Game Playing https://en.wikipedia.org/wiki/General_game_playing
10. [Електронний ресурс] Wikipedia: Гра в 15 https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0_%D0%B2_15
11. [Електронний ресурс] Wikipedia: Monte Carlo tree search https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
12. [Електронний ресурс] Кінцеві автомати http://jak.bono.odessa.ua/articles/kincevij-avtomat.php
13. [Електронний ресурс] Wikipedia: Формалізаціяhttps://uk.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ІШІ – ігровий штучний інтелект

ШІ - штучний інтелект

MCTS – Monte-Carlo Tree Search

UCT – Upper Confidence bounds applied to trees