МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту
Завідувач кафедри ПМ та МСС
_____ к.ф.м.н., доцент Коплик І. В.

«___» _____ 20__р.

КОМПЛЕКСНА КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»

спеціальність 113 «Прикладна математика»

освітньо-професійна програма «Прикладна математика»

тема роботи **«РЕАЛІЗАЦІЯ АЛГОРИТМУ НАВЧАННЯ З ПІДКРІПЛЕННЯМ В УМОВАХ СТОХАСТИЧНОГО СЕРЕДОВИЩА»**

Виконавець

Студент факультету ЕлІТ

Яковлев М.М. _____

Науковий керівник

к.ф-м.н.., доцент

Князь І.О. _____

Суми - 2020

РЕФЕРАТ

Назва документа: звіт з переддипломної практики на тему «РЕАЛІЗАЦІЯ АЛГОРИТМУ НАВЧАННЯ З ПІДКРІПЛЕННЯМ В УМОВАХ СТОХАСТИЧНОГО СЕРЕДОВИЩА».

Ключові слова: МОДЕЛЮВАННЯ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, МАШИННЕ НАВЧАННЯ, Q-НАВЧАННЯ

Короткий зміст документа та основні висновки: цей документ є звітом з переддипломної практики на тему «РЕАЛІЗАЦІЯ АЛГОРИТМУ НАВЧАННЯ З ПІДКРІПЛЕННЯМ В УМОВАХ СТОХАСТИЧНОГО СЕРЕДОВИЩА». Завдання практики – виконати аналітичний огляд літератури щодо моделювання систем на основі машинного навчання, розглянути алгоритми машинного навчання; знайти схожі реалізації даного проекту та визначити їх недоліки. Вступ містить короткі відомості про сутність сфери машинного навчання, актуальність даної сфери в житті сучасної людини. У кінці документа описані отримані на даному етапі висновки. У додатках містяться код програми.

Кількість сторінок: 27.

Кількість рисунків: 12.

Кількість використаних джерел: 17.

Кількість додатків: 1.

ЗМІСТ

ВСТУП

У наші дні технології розвиваються дуже швидко, та наше життя все більше залежить від них. Особливо швидко розвивається галузь машинного навчання. Застосовуються ці знання в багатьох сферах: оцінка страхових ризиків, бухгалтерський облік, сільське господарство, наукові дослідження тощо.

Для нашого дослідження ми обрали застосування машинного навчання в допоміжних систем керування автомобілем. Як казав Г. Штайгер, президент підрозділу Bosch Chassis System Control, «Системи допомоги водію – це обов'язковий крок на шляху до глобальної цілі знизити рівень смертності на дорогах до «нуля»».

Зараз вже існує багато рішень цієї задачі. Одна з останніх розробок в цій області – технологія Мобілай (Mobileye), яка використовується в машинах BMW, General Motors тощо. Ця система відстежує траєкторію руху автомобіля і попереджує водія, якщо він занадто близько наблизиться до іншого автомобілю. При цьому автомобілю доступно автоматичне гальмування в критичних ситуаціях. Інший приклад - розробка компанії Tesla Autopilot 3.0. Ці системи є досить громіздкими та потребують дороге обладнання. Більшість з них заснована на нейронних мережах. На автомобілі розміщуються камери, та машинне навчання використовується для розпізнавання образів та реакції автомобіля на оточення. Наша ідея полягає в тому, що можна використовувати алгоритми навчання з підкріпленням, а саме Q-навчання, для розв'язку цієї задачі. Використовування даних з таблиць дає змогу навченому автомобілю з дуже високою точністю обирати правильну дію в певній ситуації.

Мета роботи – показати можливість застосування алгоритму Q-навчання для навігації автомобіля в стохастичному середовищі. Для цього необхідно побудувати комп'ютерну модель навігації автомобіля в стохастичному середовищі за допомогою алгоритму Q-навчання. Навігація агенту повинна відбуватися за допомогою сенсорів, які вимірюють лінійну відстань до перешкод.

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Огляд літератури

Машинне навчання - це додаток штучного інтелекту (AI), який надає системам можливість автоматично навчатися та вдосконалюватись із досвіду без явного внесення команд людиною. Машинне навчання орієнтоване на розробку комп’ютерних програм, які можуть отримати доступ до даних та використовувати їх для навчання. Процес навчання починається із спостережень або даних, таких, наприклад, як безпосередній досвід чи інструкція, щоб шукати шаблони даних та приймати кращі рішення в майбутньому на основі наданих нами прикладів. Основна мета - дозволити комп'ютерам навчатися автоматично, без втручання або допомоги людини, та відповідно коригувати дії. Використовуючи класичні алгоритми машинного навчання, текст розглядається як послідовність ключових слів. Натомість підхід, заснований на семантичному аналізі, імітує здатність людини розуміти значення тексту.

Розглянемо методи машинного навчання. Алгоритми машинного навчання часто класифікуються як навчання з учителем (керовані) та без вчителя (не керовані).

- *Керовані алгоритми машинного навчання.* Можуть застосовувати те, що було вивчено раніше, до нових даних, використовуючи приклади для прогнозування майбутніх подій. Починаючи з аналізу відомого навчального набору даних, алгоритм навчання виробляє певну функцію для прогнозування вихідних значень. Система здатна забезпечити цілі для будь-якого нового входу після достатньої підготовки. Алгоритм навчання може також порівнювати його результат з правильним, призначеним результатом та знаходити помилки, щоб відповідно модифікувати модель.
- *Некеровані алгоритми машинного навчання.* Використовуються, коли інформація, що використовується для тренування, не є ні

класифікованою, ні маркованою. Без нагляду навчання вивчає як системи можуть зробити висновок про функцію опису прихованої структури з зазначених даних. Система не знаходить правильного виводу, але вона досліджує дані і може робити висновки з них для опису прихованих структур з не маркованих даних.

- *Напівкеровані алгоритми машинного навчання.* Знаходяться десь між контрольованим та непідконтрольним навчанням, оскільки вони використовують як марковані, так і немарковані дані для навчання - як правило, це невелика кількість мічених даних та велика кількість не мічених даних. Системи, що використовують цей метод, здатні значно підвищити точність навчання. Зазвичай навчання під наглядом вибирають тоді, коли отримані мічені дані потребують кваліфікованих та відповідних ресурсів для того, щоб їх навчати / вчитися з них. В іншому випадку для придбання маркованих даних зазвичай не потрібні додаткові ресурси.
- *Алгоритми машинного навчання з підкріпленням*. Це метод навчання, який взаємодіє зі своїм оточенням, виробляючи дії та виявляючи помилки чи винагороду. Пошук проб і помилок та відстрочення винагороди є найбільш релевантними характеристиками підкріплення. Цей метод дозволяє машинним та програмним агентам автоматично визначати ідеальну поведінку в конкретному контексті, щоб максимально підвищити його ефективність. Простий зворотний зв'язок з винагородою необхідний, щоб агент дізнався, яка дія найкраща; це відомо як сигнал посилення.

Для нашої роботи ми використовуємо алгоритм машинного навчання з підкріпленням. Найбільш повною та актуальною роботою на тему навчання з підкріпленням на даний момент є книга Річарда Сатона та Ендрю Барто «Reinforcement Learning: An Introduction»

В цій книзі описана основна проблематика навчання з підкріпленням, історія розвитку даної галузі та сучасне використання технологій з даної сфери. У розділі 6.5 під назвою «Q-Learning: Off-Policy TD Control» детально описується алгоритм Q-навчання та порівняння його з іншими алгоритмами.

Щодо нейронних мереж, зазначимо їх переваги та недоліки і пояснимо саме чому був обраний алгоритм Q-навчання.

Переваги нейронних мереж:

- Стійкість до шумів вхідних даних (нейронні мережі здатні коректно функціонувати, навіть якщо на вході дані зашумленими);
- Адаптація к змінам;
- Відмовостійкість (нейронні мережі здатні нормально функціонувати навіть при досить серйозних пошкодженнях);
- Надвисока швидкість.

Недоліки нейронних мереж:

- Прийняття рішень в декілька етапів;
- Обчислювальні завдання;
- Відповідь завжди приблизна.

Для нашої задачі дуже важливим параметром є саме точність отриманого результату. Хоча нейронні мережі можуть видавати досить точні значення, але все ж присутня вірогідність отримати з великою похибкою, що в деяких сферах є неприпустимим. А алгоритм Q-навчання зберігає точні значення, отримані в процесі навчання. Агент, навчений за допомогою Q-навчання не приймає випадкових рішень, а користується даними з таблиці.

Зараз активно розповсюджується метод DQN. Основна ідея полягає в заміні формули розрахунку Q значень на нейронну мережу. На жаль, цей метод за своєю архітектурою працює тільки з дискретними діями, тому застосування його для нашої задачі є не дуже доцільним. У DQN на вхід нейронної мережі подається поточний state (поточна ситуація), а на виході нейронна мережа прогнозує число Q. А так як на виході мережі перераховані відразу всі можливі дії (кожен зі своїм

передбаченим Q), то виходить що нейронна мережа в DQN реалізує класичну функцію Q (s, a) з Q-learning. Видає Q для state і action (тому позначення Q (s, a) як функції від s і a). Ми просто шукаємо звичайним argmax по масиву серед виходів мережі осередок з максимальним числом Q і робимо дію, яке відповідає індексу цього осередку. Причому можна завжди вибирати дію з максимальним Q, тоді така політика буде називатися детерменістской. А можна вибирати дію як випадкове з доступних, але пропорційно їх Q-значень (тобто дії з високим Q будуть вибиратися частіше, ніж з низьким). Така політика називається стохастична. У стохастичного вибору плюс в тому, що автоматично реалізується пошук та дослідження світу (Exploration), так як кожен раз вибираються різні дії, іноді не здаються найоптимальнішими, але можуть в майбутньому привести до великої нагороди.

Серед наукових робіт за схожою тематикою була знайдена робота І.І. Чернявського «Применение машинного обучения для создания управляющих автоматов на примере игры «ROBOCODE»» Завдання, яке вирішується в даній роботі, полягає в розробці методу автоматичної побудови керуючих автоматів. Це завдання розглядається на прикладі побудови танка для гри «Robocode». Гра являє собою змагання роботів-танків на прямокутному полі. Танк складається з корпусу, радара і гармати.

Для вирішення даної задачі автор використав комбінований метод, в якому використовувалися як Q-навчання, так і нейронні мережі. Стан автомата описується нейронною мережу, яка обчислює Q-функцію.

1.2 Механізм функціонування навчання з підкріпленням.

Навчання з підкріпленням (англ. Reinforcement learning) – сфера машинного навчання, яка вивчає які дії програмні агенти повинні вживати у навколишньому середовищі з метою максимізації винагороди.

Базова модель навчання з підкріпленням складається з:

- множини станів середовища **S**;
- множини доступних дій **A**;

- правил переходу між станами;
- правил скалярної безпосередньої винагороди переходу;
- правил, які описують, що спостерігає агент.

Правила часто бувають стохастичними. Спостереження зазвичай передбачає скалярну, негайну винагороду, пов'язану з останнім переходом. У багатьох роботах агент передбачає спостереження за поточним оточенням (повна спостережливість). Якщо ні, то агент має часткову спостережливість. Іноді набір доступних для агенту дій обмежується (нульовий баланс неможливо зменшити. Наприклад, якщо поточне значення агенту дорівнює 3, а перехід стану зменшує значення на 4, перехід не буде дозволений).

1.3 Принцип роботи Q-навчання

Одним з найважливіших проривів у навчанні підкріплення було розробка алгоритму керування методом управління ТД, відомого як Q-навчання (Watkins, 1989).

Формула Беллмана для розрахунку Q значень виглядає так:

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

S_t – стан агенту,

a_t – дія агенту,

α – фактор навчання,

r_t – нагорода,

γ - фактор дисконтування.

Фактор навчання визначає, якою мірою нова інформація перевизначатиме стару. Фактор дисконтування визначає важливість майбутніх винагород.

Загальна схема роботи Q-навчання:

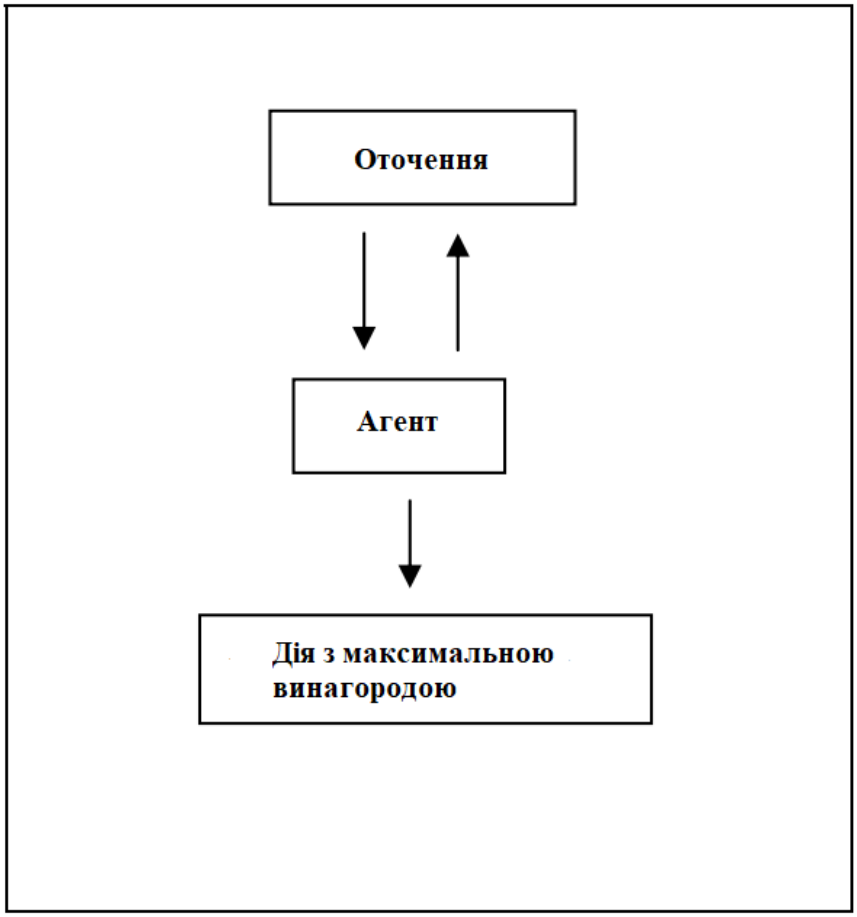

Рис. 1.1. Схема взаємодії частин системи Q-начання

Але для нашої задачі необхідно модифікувати формулу розрахунку. В стандартних задачах з використанням Q-навчання використовується функція максимуму, оскільки найважливішим показником для агенту є пошук шляхів здобуття найбільшої винагороди. Для нашої задачі – уникнення значень з дуже низькою винагородою. Для агенту найважливішою задачею є уникнення перешкод, тому функцію необхідно замінити на мінімум.

Остаточний вигляд формули для розрахунку Q значень:

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \min_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Умовний вигляд значень в Q таблиці виглядає так:

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Рис 1.2 Таблиця Q-навчання. Верхня частина до тренування, нижня після тренування

2. ПРАКТИЧНА РЕАЛІЗАЦІЯ

2.1 Побудова моделі

Для розв’язку даної задачі:

- Була обрана мова програмування Python 3;
- Будуємо модель клітинного автомату;
- В якості алгоритму навчання обираємо Q-learning;
- Для навігації у просторі на тілі агенту розміщуємо 5 сенсорів для виміру лінійної відстані до об’єктів.

В грі присутні 3 типи об’єктів: дороги, перешкоди та агент. Агент є однією клітинкою в моделі «клітинний автомат», на якому знаходяться сенсори. Для зручності побудови клітинного автомату кут між сенсорами 45°. Область видимості об’єктів позначається сірим кольором на рис. 1.

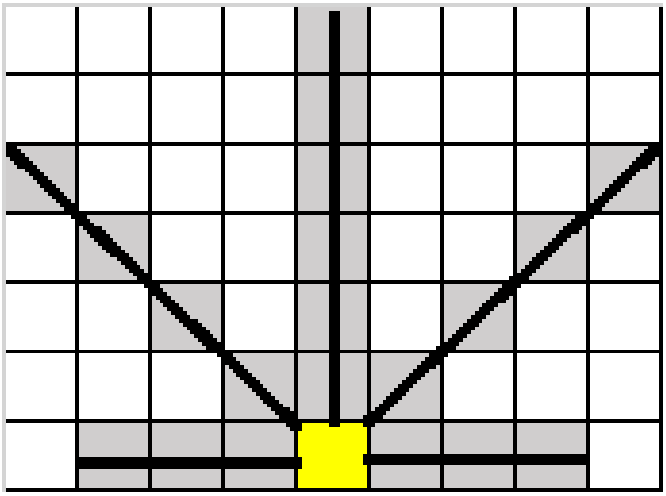

Рис.2.1. Схематичне зображення моделі агенту та сенсорних систем

Обмеження, накладені на агента:

- Може рухатися в 3 напрямках: ліворуч, праворуч та прямо;
- За одну ітерацію переміщується на 1 клітинку.

2.2 Реалізація алгоритму Q-learning

Стан агенту визначається набором значень сенсорів. Набір значень є строго упорядкованим, довжини записуються від лівого сенсора за годинниковою стрілкою. Дія агенту визначається літерами: l – ліворуч, r – праворуч, f – прямо.

Програмно Q-таблиця реалізується у вигляді словника(хеш-таблиці), у якій в якості ключа виступає набір **стан+дія**. Значення Q-таблиці зберігається у вигляді (4 2 1 4 2 ‘r’) 1.25 . Якщо дія агенту не призвела до зіткнення з перешкодою, даний крок отримує винагороду 10, якщо призвела, то -100.

Як зазначалося вище, реалізація відбувається на мові програмування Python. Були використані бібліотеки random (для здобуття випадкових значень) та matplotlib (для побудови графіків). Бібліотеки з готовими реалізаціями машинного навчання не використовувалися.

2.3 Процес навчання

Процес навчання можна розділити на 4 основних етапи:

1. Завантаження Q-таблиці, якщо агент попередньо натренований. Якщо ні, то створення порожньої Q-таблиці.
2. Генерація оточення. Відбувається за допомогою функції random() з зазначенням співвідношення кількості доріг до кількості перешкод.
3. Запуск симуляції руху агенту по середовищу. Для першого этапу симуляції використовуються випадкові кроки. Кожні N симуляцій генерується нове оточення для моделювання більшої кількості ситуацій.
4. Остання симуляція. Відбувається без випадкових кроків. В кінці симуляції створюється анімація руху агенту. Таблиця Q зберігається у текстовий файл.

Успішність дай агенту оцінюються кількістю кроків до зіткнення з перешкодою. На основі ціх данних можно побудувати стовпчикову діаграму для спостереження за процесом навчання.

2.4 Моделювання з різними параметрами

Для всіх моделей використовувалися такі параметри:

В формулі розрахунку Q-значень: $\alpha = 0.1$, $\gamma = 0.4$

Оточення має розміри 40x40

Кількість ітерацій в моделюванні – 1 000 000

Обмеження на кількість кроків агенту: 1000

Процес навчання розділимо на 3 етапи:

- Дослідження. Eps = 0.9;
- Основне навчання. Eps = 0.4;
- Фінальне навчання. Eps = 0.02.

Результатом моделювання є графік навчання та анімація руху агенту.

Для відображення даних на стовпчиковій діаграмі ділимо данні на етапи, описані вище.

Довжини сенсорів записуємо від лівого до правого.

Стовпчикові діаграми, отримані після моделювання з різними параметрами:

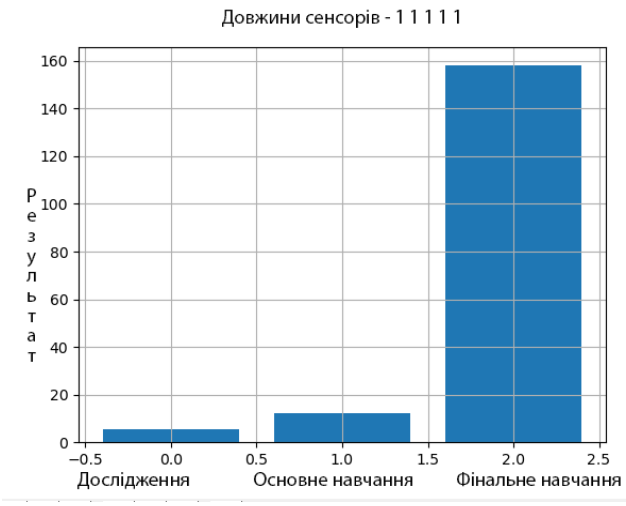

Рис 2.2 Результат з довжинами сенсорів 1 1 1 1 1

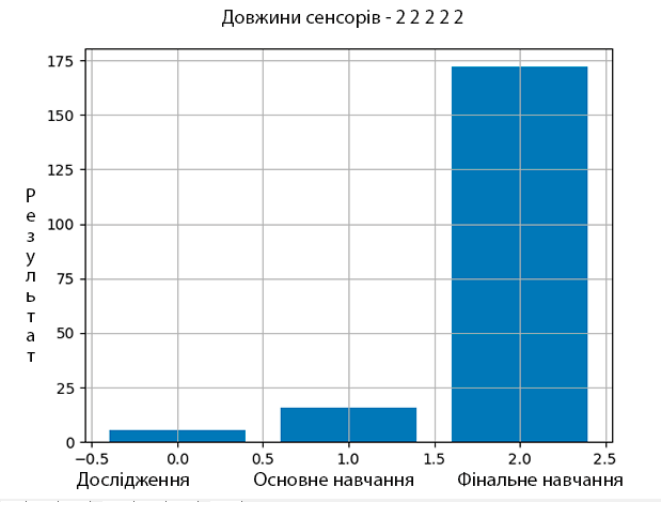

Рис 2.3 Результат з довжинами сенсорів 2 2 2 2 2

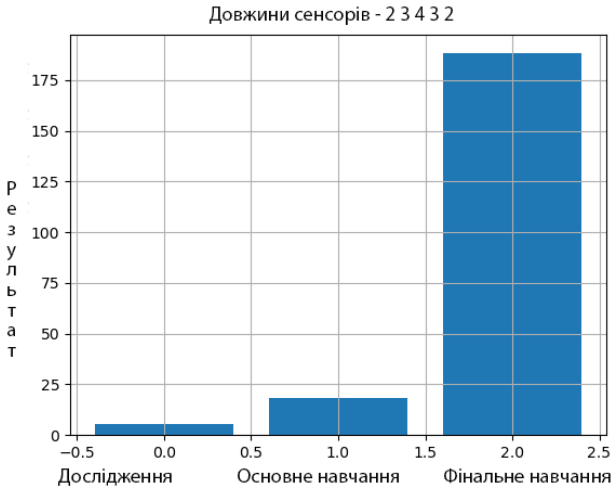

Рис 2.4 Результат з довжинами сенсорів 2 3 4 3 2

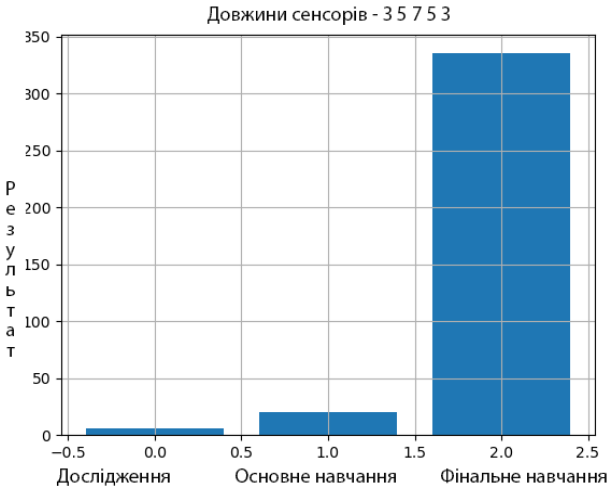

Рис 2.5 Результат з довжинами сенсорів 3 5 7 5 3

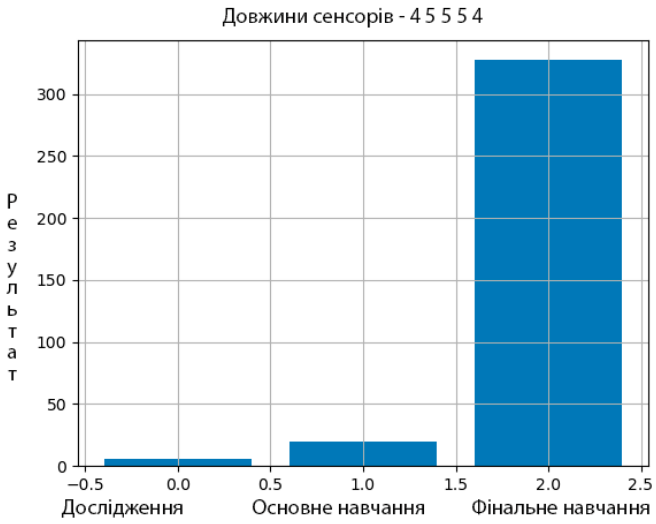

Рис 2.6 Результат з довжинами сенсорів 4 5 5 5 4

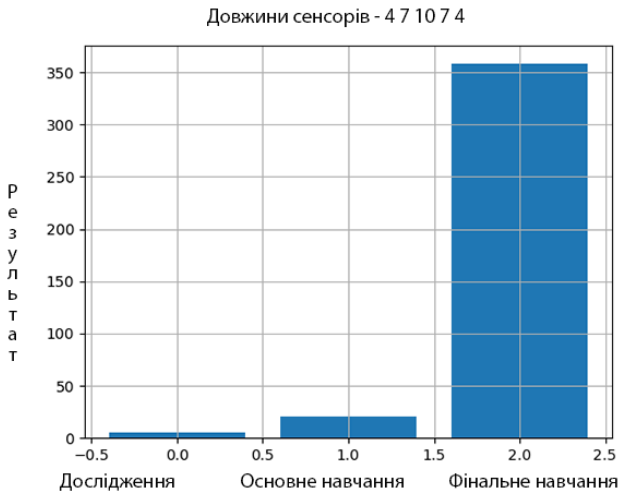

Рис 2.7 Результат з довжинами сенсорів 4 7 10 7 4

З діаграм очевидно, що зі збільшенням кількості можливих комбінацій покращується результат.

Порахувати кількість можливих комбінацій можно за формулою:

$$\prod_{i=1}^{n}(l_i + 1)$$

Де l_i – довжина і-го сенсору.

Кадри з анімацій, створених в результаті моделювань для значень сенсорів (4 5 7 5 4):

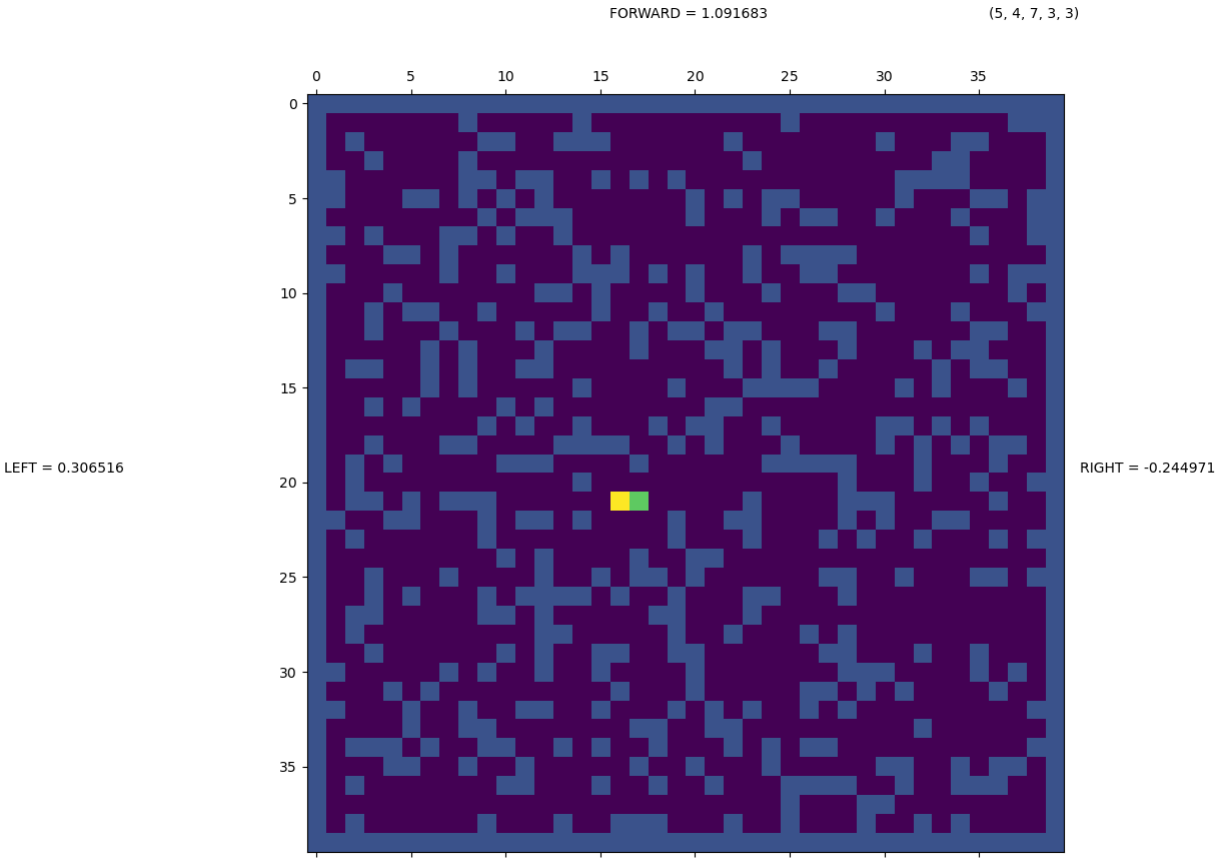

Рис 2.8 Кадр з анімації моделювання руху агенту

Перешкоди позначаються синім кольором, дорога – фіолетовим, агент – жовтим, а зеленим кольором зображено розташування агенту на попередньому кроці. Позначення того, де агент був на попередньому кроці необхідно для того, щоб визначити в яку сторону він рухається. Показники сенсорів, Left, Right та Forward зазначені в напрямку руху агенту.

Значення Left, Forward та Right показують значення Q-таблиці для відповідних дій агенту. Праворуч від значення Forward знаходяться показники сенсорів.

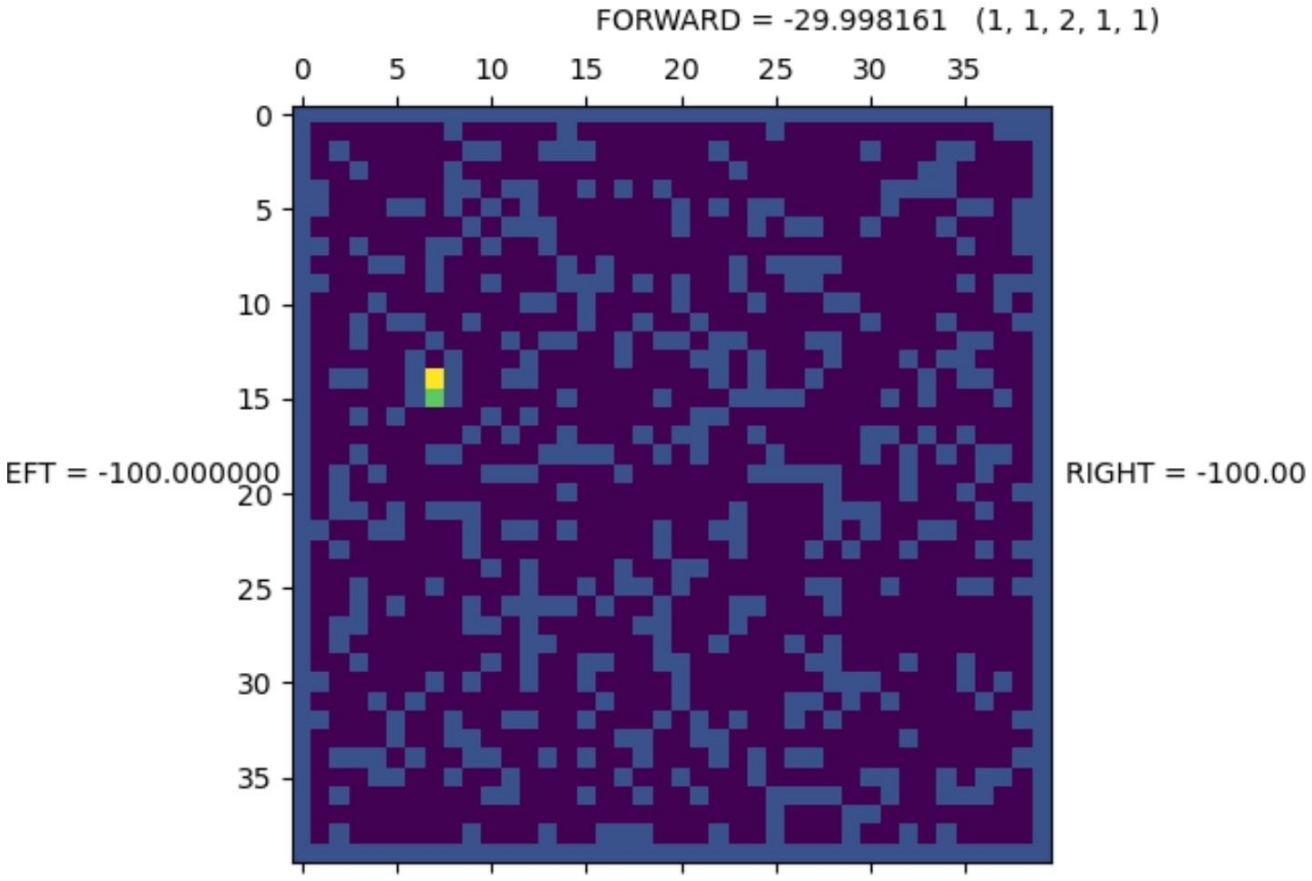

Рис 2.9. Кадр з анімації, на якому агент потрапив до пастки

Для отримання цього кадру використовувався недостатньо навчений агент, який потрапив до пастки. Після даного моделювання значення в Q-таблиці будуть перераховані, і наступного разу вірогідність потрапити в таку ж пастку буде меншою.

Побудуємо залежність кількості комбінацій від отриманого результату. Для побудови даної залежності використовувались сенсорні системи з рівними значеннями довжин всіх сенсорів (1 1 1 1 1, 2 2 2 2 2 тощо). З попередніх діаграм немає сенсу брати дуже великі значення довжин сенсорів. Обмежимося набором (9 9 9 9 9). Для оцінки результатів використовуємо етап фінального навчання, оскільки на цьому етапі найменша кількість випадкових кроків та агент пройшов 75% навчання.

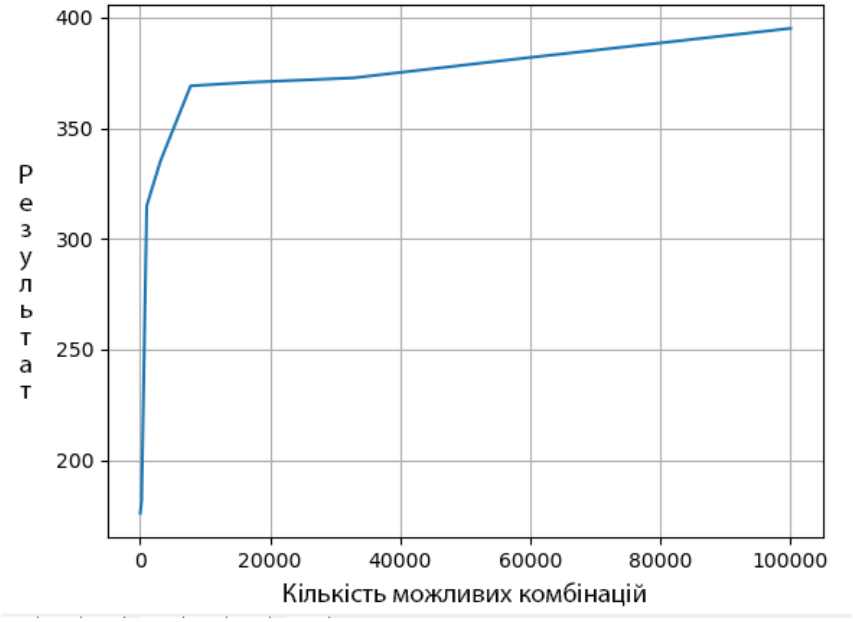

Рис 2.8 Графік залежності результату від кількості можливих комбінацій

Графік схожий на гілку параболи. Зі збільшенням кількості можливих комбінацій збільшується отриманий результат, але також збільшується швидкість пошуку дії з Q-таблиці та кількість пам’яті, необхідної для роботи програми.

ВИСНОВКИ

У роботі показано, що навчання агенту у неоднорідному стохастичному середовище можливо у рамках модифікованого метода Q-Learning.

Оптимально стратегія (рух на максимальній відстані від перепон) досягається за умови модифікації класичного метода Q-Learning, а саме заміні в формулі Беллмана максимуму на мінімум. Таким чином, агент отримує більш точну інформацію про відстані до перешкод.

Основним фактором, який впливає на результативність дій агенту, є кількість можливих комбінацій сенсорних систем.

Оптимальна кількість та розташування сенсорних датчиків у рамках моделі клітинного автомату – 5 датчиків з кутом 45° між ними. Таке розташування сенсорних систем дає змогу збирати достатнью кілкість інформації з оточення для успішної навігації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Reinforcement Learning: An Introduction» Richard S. Sutton and Andrew G. Barto c 2014, 2015
2. Matiisen, Tambet (December 19, 2015). "Demystifying Deep Reinforcement Learning"
3. Baird, Leemon (1995). "Residual algorithms: Reinforcement learning with function approximation"
4. Bozinovski, S. (1982). "A self learning system using secondary reinforcement"
5. И.И.Чернявский «Применение машинного обучения для создания управляющих автоматов на примере игры «ROBOCODE»»
6. Thomas Bäck & Hans-Paul Schwefel (Spring 1993) "An overview of evolutionary algorithms for parameter optimization"
7. Klyubin, A., Polani, D., and Nehaniv, C. (2008). Keep your options open: an information-based driving principle for sensorimotor systems.
8. Francois-Lavet, Vincent; et al. (2018). "An Introduction to Deep Reinforcement Learning".
9. Mnih, Volodymyr; et al. (2015). "Human-level control through deep reinforcement learning"
10. Kaplan, F. and Oudeyer, P. (2004). Maximizing learning progress: an internal reward system for development. Embodied artificial intelligence
11. Sutton & Barto 1998, §6. Temporal-Difference Learning
12. https://expertsystem.com/machine-learning-definition/
13. https://itc.ua/blogs/avtopilot-tesla-vyhodit-na-novyj-uroven-nachalos-testirovanie-proaktivnoj-funkczii-reagirovaniya-na-signaly-svetoforov-i-dorozhnye-znaki/
14. Williams, Ronald J. (1987). "A class of gradient-estimating algorithms for reinforcement learning in neural networks".

15. Tokic, Michel; Palm, Günther (2011), "Value-Difference Based Exploration: Adaptive Control Between Epsilon-Greedy and Softmax"

16. Kulkarni, Tejas D.; Narasimhan, Karthik R.; Saeedi, Ardavan; Tenenbaum, Joshua B. (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation"

17. Klyubin, A., Polani, D., and Nehaniv, C. (2008). Keep your options open: an information-based driving principle for sensorimotor systems. PloS ONE, 3(12):e4018. https://dx.doi.org/10.1371%2Fjournal.pone.0004018

ДОДАТОК А

Основні класи програми з коментарями:

Реалізація Q-learning:

```
class Q:
    def __init__(self, state = {}):
        self.gamma = 0.4
        self.alpha = 0.1
        self.state = state

    def get_wp(self, plr):
        self.plr = plr

    def run_model(self, silent=1):
        self.plr.curr_state = tuple(self.plr.get_features())
        r = self.plr.reward
        if self.plr.prev_state not in self.state:
            self.state[self.plr.prev_state] = 0
        if r == -100:
            nvec = 0
        else:
            nvec = []
            for i in self.plr.actions:
                cstate = self.plr.curr_state  + tuple(self.plr.actions[i])
                if cstate not in self.state:
                    self.state[cstate] = 0
                nvec.append(self.state[cstate])
            #print(self.plr.prev_state, r, nvec)
            nvec = min(nvec)
        self.state[self.plr.prev_state] = self.state[self.plr.prev_state] + self.alpha * (-
self.state[self.plr.prev_state]
                + r + self.gamma * nvec)
```

Клас гри:

```
class W:
    def __init__(self, map, QModel, eps, x = 1, y = 1, randomDest = False):
        self.P=P(self,x,y, eps)
        self.map = map
        self.n = 0
        self.m = 0
        self.fillSize()
        if randomDest:
```

```
            self.doRandomDest()
        self.QM = QModel
        self.QM.get_wp(self.P)

    def fillSize(self):
        self.n = len(map)
        self.m = len(map[0])

    def doRandomDest(self):
        flag = True
        while flag:
            x = random.randint(0, self.n - 1)
            y = random.randint(0, self.m - 1)
            if map[x][y] == 0:
                flag = False
                self.P.x = x
                self.P.y = y
            #print(self.P.x, self.P.y)

    def step(self):
        self.P.move()

    def is_finished(self):
        px, py = self.P.getxy()
        end_bool = 0
        if map[px][py] == 1:
            end_bool = 1
        if map[px][py] == 2:
            end_bool = 2
        return end_bool

    def get_reward(self, end_bool, iteration):
        if end_bool == 0:
            self.P.reward = 10
        if end_bool == 1:
            self.P.reward = -100

    def play(self, anim = False):
        end_bool = self.is_finished()
        iter = 0
        ANIM = []
        try:
            self.P.sensorController.collectData(self.P)
```

```
        except IndexError:
            print("ERROR: ", self.P.getxy())
        while end_bool == 0:
            # block for animation
            if anim:
                ANIM.append([self.P.x, self.P.y])
                name1 = tuple(self.P.get_features())
                for i in self.P.actions:
                    namea = name1 + tuple(self.P.actions[i])
                    if namea not in self.QM.state:
                        self.QM.state[namea] = 0
                    ANIM[iter].append(self.QM.state[namea])
                ANIM[iter].append(name1)
            #exception
            if iter > 1000:
                break
            #main proces
            self.step()
            end_bool = self.is_finished()
            self.get_reward(end_bool, iter)
            self.QM.run_model()
            iter = iter + 1
        if anim:
            return ANIM
        return iter
```

Клас сутності (опис рухомого об’єкту):

```
class un:
    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.actions = {"forward": 'f',
                        "left": 'l',
                        "right": 'r'}
    def getxy(self):
        return self.x, self.y
```

Клас агенту:

```
class P(un):
    def __init__(self,W, x, y, eps):
        self.sensorController = sensorsController()
        self.W = W
        self.dx = -1
```

```
        self.dy = 0
        self.eps = eps
        self.movmnt = 'f'
        un.__init__(self, x, y)
        self.prev_state = tuple(self.get_features()) + (self.dx, self.dy)
        self.curr_state = tuple(self.get_features()) + (self.dx, self.dy)

    def setTarget(self):
        for i in range(0, self.W.n):
            for j in range(0, self.W.m):
                if self.W.map[i][j] == 2:
                    self.targetX = i
                    self.targetY = j


    def get_dxdy(self):
        return self.dx, self.dy
    def get_features(self):
        features = []
        features.append(self.sensorController.leftSensor.distance)
        features.append(self.sensorController.leftMiddleSensor.distance)
        features.append(self.sensorController.middleSensor.distance)
        features.append(self.sensorController.rightMiddleSensor.distance)
        features.append(self.sensorController.rightSensor.distance)
        return features
    def strtg(self):
        randomnum = random.random()
        if  randomnum < self.eps:
            a = []
            for i in self.actions:
                a.append(self.actions[i])
            act = random.choice(a)
        else:
            name1 = tuple(self.get_features())
            best = ['f', float('-inf')]
            for i in self.actions:
                namea = name1 + tuple(self.actions[i])
                if namea not in self.W.QM.state:
                    self.W.QM.state[namea] = 0
                if best[1] < self.W.QM.state[namea]:
                    best = [self.actions[i], self.W.QM.state[namea]]
            act = best[0]
        return act
```

```
    def getXYFromMovmnt(self):
        if self.movmnt == 'l':
            tempdx = -self.dy
            self.dy = self.dx
            self.dx = tempdx
        if self.movmnt == 'r':
            tempdx = self.dy
            self.dy = -self.dx
            self.dx = tempdx
    def move(self):
        self.movmnt = self.strtg()
        self.getXYFromMovmnt()
        self.prev_state = tuple(self.get_features()) + tuple(self.movmnt)
        a = self.x + self.dx
        b = self.y + self.dy
        expr = ((0 <= a < self.W.n) and (0 <= b < self.W.m))
        if expr:
            self.x = a
            self.y = b
            #try:
            self.sensorController.collectData(self)
```

Загальний опис сенсору:

```
class sensor():
    def __init__(self, lenght):
        self.lenght = lenght
        self.distance = 1
```

Контролер, який відповідає за оновлення даних сенсорів:

```
class sensorsController():
    def __init__(self):
        self.leftSensor = sensor(leftSensorLenght)
        self.leftMiddleSensor = sensor(leftMiddleSensorLenght)
        self.middleSensor = sensor(middleSensorLenght)
        self.rightMiddleSensor = sensor(rightMiddleSensorLenght)
        self.rightSensor = sensor(rightSensorLenght)
    def collectData(self, player : P):
        self.leftSensor.distance = 1
        self.leftMiddleSensor.distance = 1
        self.middleSensor.distance = 1
        self.rightMiddleSensor.distance = 1
        self.rightSensor.distance = 1
        x, y = player.getxy()
```

```
            dx, dy = player.get_dxdy()

            if x == 0 or x == player.W.n - 1 or y == 0 or y == player.W.m - 1:
                self.leftSensor.distance = 0
                self.leftMiddleSensor.distance = 0
                self.middleSensor.distance = 0
                self.rightMiddleSensor.distance = 0
                self.rightSensor.distance = 0
                #self.backSensorLenght = 0
                return 0
            i = 1
            while i <= leftSensorLenght and map[x - dy*i][y + dx*i] != 1:
                self.leftSensor.distance += 1
                i += 1
            i = 1
            while i <= leftMiddleSensorLenght and map[x + (dx - dy) * i][y + (dx + dy) *
i] != 1:
                self.leftMiddleSensor.distance += 1
                i += 1
            i = 1
            while i <= middleSensorLenght and map[x + dx * i][y + dy * i] != 1:
                self.middleSensor.distance += 1
                i += 1
            i = 1
            while i <= rightMiddleSensorLenght and map[x + (dx + dy) * i][y + (-dx +
dy) * i] != 1:
                self.rightMiddleSensor.distance += 1
                i += 1
            i = 1
            while i <= rightSensorLenght and map[x + dy * i][y - dx * i] != 1:
                self.rightSensor.distance += 1
                i += 1
            i = 1
```

Генератор мапи:

```
class MapGenerator:
    def generateMap(self,map1, eps):
        n = len(map1)
        m = len(map1[0])
        for i in range(1, n-1):
            for j in range(1, m-1):
                randomnum = random.random()
                if randomnum < eps:
```

```
                map1[i][j] = 0
            else:
                map1[i][j] = 1
    #MapGenerator.printMap(map1)
def printMap(self, map1):
    for i in range(0, n):
        for j in range(0, m):
            print(map1[i][j], end=' '),
        print()
```