Комп’ютерний практикум №2

Моделі пошуку в стохастичному і невідомому середовищі

ПІБ: Кононов М. А.

Група: ЗПІ-зп01

Мета роботи: ознайомитись з алгоритмами пошуку в умовах невідомості та навчання з підкріпленнями; дослідити їх використання для інтелектуального агента в типовому середовищі.

***З*авдання:** обрати середовище моделювання та задачу, що містить агента, який може бути навчений методом «з підкріпленнями». В обраному середовищі вирішити задачу знаходження найкращої стратегії поведінки, реалізувавши один з методів. Виконати дослідження реалізованого методу.

Номер варіанту: 8

Завдання для варіанту:

- форма карти: кімнати з вузькими проходами-дверми;
- задача дослідження: вплив значення винагороди та вартості руху;
- алгоритм: Q-learning.

Середовище: коротко опишіть обране середовище, чи стохастичне воно. Формалізуйте середовище в термінах станів, можливих дій, переходів і винагород

Пакмен – це гра, де жовта куля з ротиком рухається по лабіринту та намагається з’їсти якомога більше харчових гранул (маленькі білі крапки), уникаючи привидів (інші агенти з очима). Якщо Пакмен з’їсть всю їжу в лабіринті, він виграє. Великі білі крапки – капсули, які дають Пакмену можливість з’їсти привида за обмежений час. Середовище є стохастичним, оскільки початкові положення, а також вибір дій рухомих об’єктів

здійснюється випадково: на одній і тій самій карті при тих самих параметрах можна отримати зовсім різні результати гри. Згодом ця випадковість може зменшуватись і прийняті рішення базуються на стані середовища.

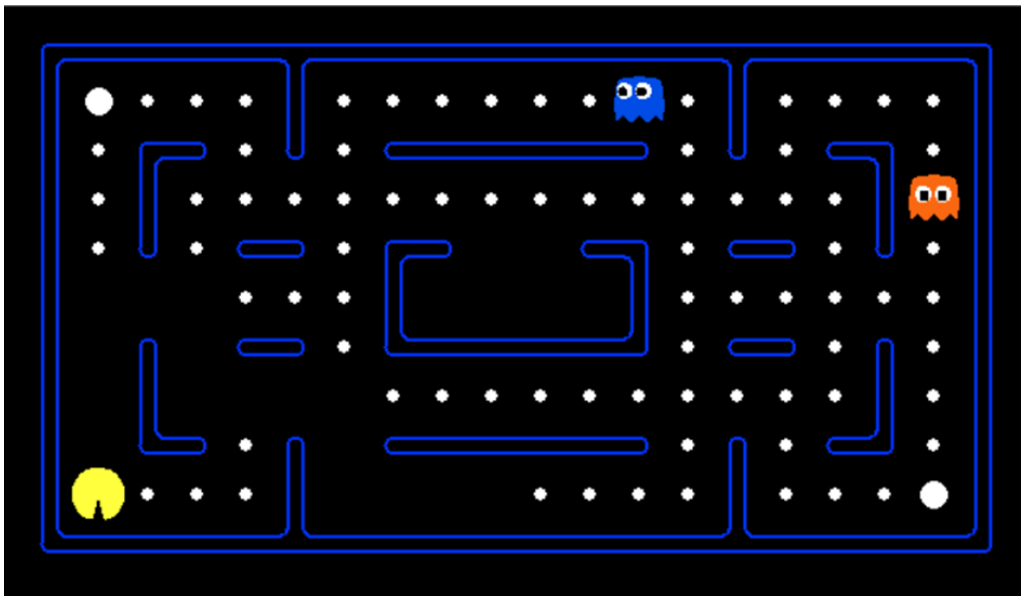

Рисунок 1. Зовнішній вигляд середовища

Метод вирішення задачі: охарактеризуйте метод з Вашого варіанту (Q-learning, Value iteration, або інший), офлайн чи онлайн тощо.

Q-learning було введено Крісом Воткінсом в 1989 році. Доведення збіжності було представлено Воткінсом та Даяном 1992 року. Цей метод використовується в штучному інтелекті при агентному підході, відноситься до експериментів виду навчання з підкріпленням. На основі отриманої від середовища винагороди агент формує функцію корисності Q, що в подальшому дає йому можливість вже не випадково обирати стратегію поведінки, а враховувати досвід попередньої взаємодії з середовищем. Однією з переваг Q-learning є те, що воно в змозі порівняти очікувану корисність доступних дій без формування моделі навколишнього середовища. Цей метод використовується для ситуацій, які можна представити у вигляді Марковського процесу прийняття рішень.

Оскільки Q-learning є ітеративним алгоритмом, воно неявно передбачає якісь початкові умови перед тим, як станеться перше уточнення. Високі початкові цінності, відомі також як «оптимістичні початкові умови», можуть заохочувати розвідування (англ. exploration): не важливо, яку дію обрано, правило уточнення призведе до того, що вона матиме нижчі цінності, ніж інші

альтернативи, підвищуючи таким чином імовірність їхнього обрання. Алгоритм складається з 5 основних кроків:

1. Initialization (Ініціалізація):

   for each s and a do Q[s, a] = RND // Ініціалізуємо функцію корисності Q від дії, a в ситуації s – як випадкову для будь-яких вхідних даних

2. Observe (Спостереження):

   s' = s // запам’ятати попередній стан

   a' = a // запам’ятати попередні дії

   s = FROM_SENSOR // отримати поточний стан від сенсора

   r = FROM_SENSOR // отримати винагороду за попередні дії

3. Update (Оновлення корисності):

   Q[s',a'] = Q[s',a'] + LF * (r + DF * MAX(Q,s) — Q[s',a'])

4. Decision (Вибір дії):

   a = ARGMAX(Q, s)

   TO_ACTIVATOR = a

5. Repeat: GO TO 2 // повторення: перехід на крок 2

Темп навчання (англ. learning rate), або розмір кроку (англ. step size), визначає, якою мірою отримана інформація перевизначить стару. Коефіцієнт 0 зробить так, що агент не навчатиметься нічому (використовуючи виключно апріорне знання), тоді як коефіцієнт 1 зробить так, що агент розглядатиме лише найновішу інформацію (ігноруючи попереднє знання для розвідування можливостей).

В найпростішому вигляді Q-learning зберігає дані в таблицях. Цей підхід має справу зі збільшенням кількостей станів та дій, оскільки правдоподібність відвідування агентом певного стану й виконання певної дії стає все меншою. Даний метод можливо поєднувати з наближенням функцій. Це уможливлює застосування даного алгоритму до більших задач, навіть коли простір станів є неперервним. Одним з рішень є використовувати як наближувач функцій (пристосовану для цього) штучну нейронну мережу. Наближення функцій може прискорювати навчання у скінченних задачах, оскільки цей алгоритм може узагальнювати попередній досвід до раніше небачених станів.

Реалізація методу: програмна реалізація

Для вирішення поставленої задачі ми використовували середовище Pacman з наступного практикуму, усі експерименти проводились у версії, завантаженій за наданим посиланням. У даній грі вартість руху залежить від того, чи містить клітинка, в якій знаходиться персонаж, їжу. Якщо, герой з'їв гранулу, то бали додаються. Але вони також віднімаються у випадку хаотичного переміщення в протилежні сторони, коли містер Пакмен не може вирішити, куди йому прямувати. Тому ми представили вартість руху у вигляді штрафних балів, які мають відніматись від винагороди. Чим менше цей параметр, тим в меншій мірі героя штрафують за «вагання», та тим більший вплив на середовище здійснює винагорода.

Для зручності тестування впливу цих двох параметрів в головний файл pacman.py ми додали певний код. Програма запускається засобами командного рядка, тому користувачу була надана можливість задавати розмір додаткової винагороди, а також кількість штрафних балів за «топтання на місці» (параметри reward та penalty). Їх можна вказати після головної команди запуску гри, наприклад, python pacman.py --reward 100 --penalty 10. Також було створено run_game.bat, який виглядає наступним чином:

```
echo off
cls
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic --reward 10 --penalty 100
pause
```

Перед запуском гри виконуються 50 ітерацій навчання (параметр -x командного рядка). Відправною точкою роботи моделі є клас ApproximateQAgent, методи якого посилаються на інші класи таким чином:

ApproximateQAgent → PacmanQAgent → QLearningAgent →

ReinforcementAgent → ValueEstimationAgent → Agent

Усі вони містять підпрограми __init__ (ініціалізація) та getAction, в якому прописані основні дії. Їх має містити будь-який клас для реалізації алгоритму впливу агента на середовище.

Клас Agent описує всі типи дійових осіб (головного героя та його ворогів) і знаходиться в game.py.

ValueEstimationAgent є абстрактним агентом, який встановлює Q-значення (стан та дія) для середовища. ValueIterationAgent та QLearningAgent успадковують властивості від нього. ValueIterationAgent розглядає модель середовища у вигляді марковського процесу прийняття рішень, який, зазвичай, перед виконанням будь-якої дії оцінює Q-дані. На відміну від нього QLearningAgent обробляє ці параметри під час роботи в середовищі.

ReinforcementAgent – це ValueEstimationAgent, оцінюючий Q-значення в першу чергу на основі досвіду, а не стану середовища, яке викликає метод observeTransition (state,action,nextState,deltaReward), який в свою чергу здійснює оновлення стану підпрограмою update(state, action, nextState, deltaReward). Він також спирається на QLearningAgent, який отримує дані про середовище, відповідно до них встановлює нові максимально допустимі значення та дії.

PacmanQAgent визначає головні параметри роботи алгоритму: швидкість навчання (alpha – learning rate), коефіцієнт розвідування (epsilon – exploration rate), коефіцієнт знецінювання (gamma – discount factor), кількість навчальних «заходів». Його методи також викликаються агентом ApproximateQAgent при ініціалізації та завершенні гри, який в свою чергу виконує власні дії.

Також є можливість використання кількох типів «екстракторів», кожен з яких повідомляє агенту різну інформацію про навколишній світ. Наприклад, SimpleExtractor надає дані про базові рефлекси головного героя: чи може гранула бути з’їденою, наскільки далеко наступна їжа, чи є зустріч з привидом неминучою, чи знаходиться привид на один крок від нього. Використання інших екстракторів призводило до некоректної роботи програми або до протилежних результатів гри. Крім того, є функція closestFood, яка повідомляє координати найближчої їжі. Дії будь-якого екстрактора мають бути описані в методі getFeatures, який у якості параметрів отримує self, стан середовища (state) та дію (action). У даній версії гри прийнято так, що всі алгоритми

управління персонажем розміщуються в файлі submission.py. Ми вирішили не відходити від даної концепції та розмістили всі класи в цьому одному файлі.

Результати застосування розробленого методу: результати роботи програми, отримані показники корисності тощо. Бажано візуально відобразити динаміку навчання, наприклад, у вигляді залежності корисності від кількості епізодів.

Оцінка результатів: прокоментувати отримані результати, зокрема оптимальність роботи, наявність стратегії дослідження та її використання, доцільність застосування методу в середовищі тощо

Реалізовані методи були протестовані ручним способом, їх результати показані на рисунках 2-9.

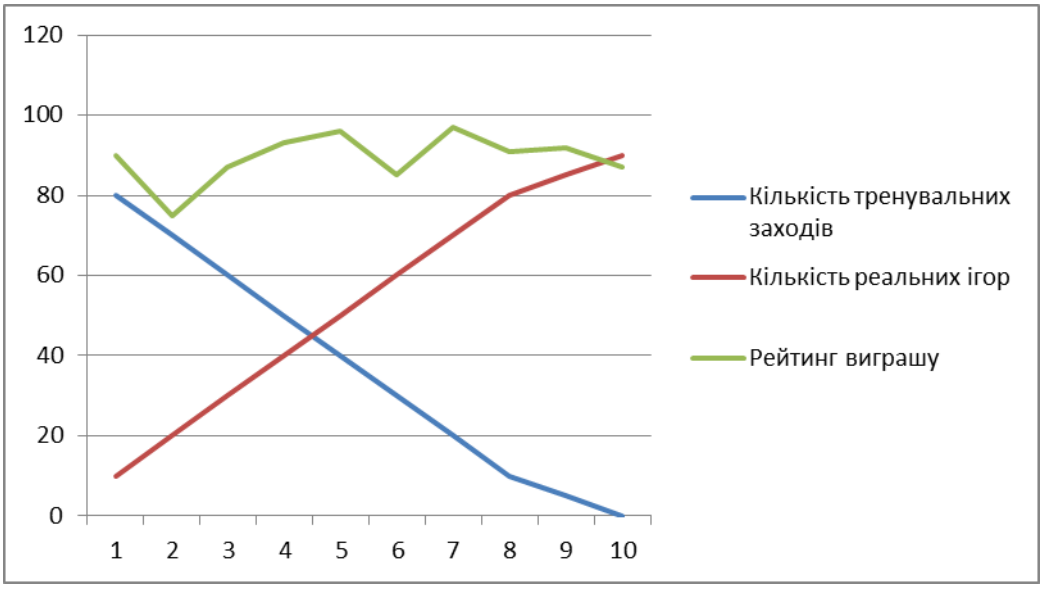

Рисунок 2. Залежність рейтингу виграшу від кількості тренувальних заходів

Програма сконструйована таким чином, що зменшення кількості тренувальних заходів призводить до збільшення кількості реальних тестів. Дивлячись на цей графік, можна стверджувати, що при сталих параметрах винагороди рейтинг виграшу не залежить від кількості тренувальних заходів.

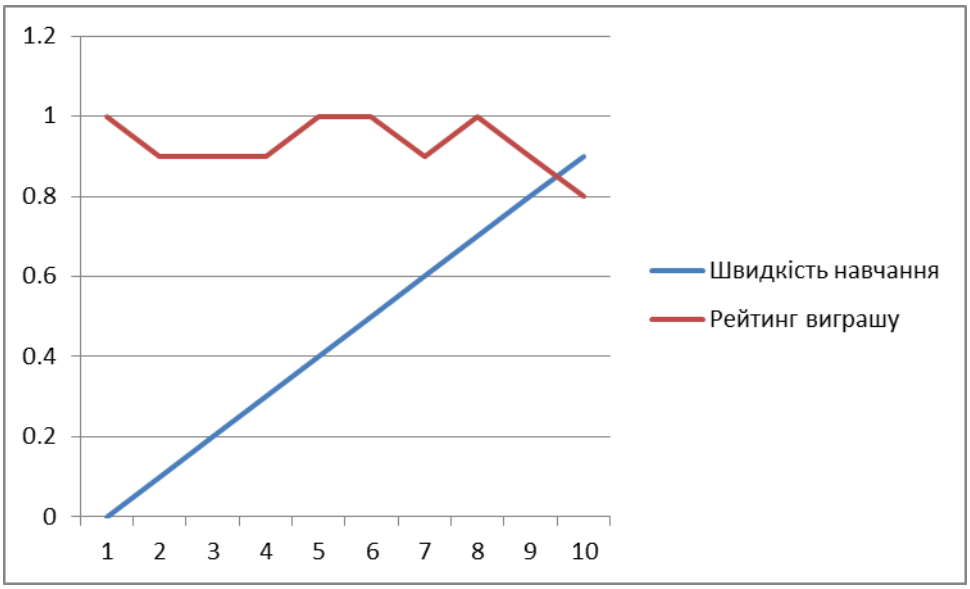

Рисунок 3. Залежність рейтингу виграшу від швидкості навчання (alpha)

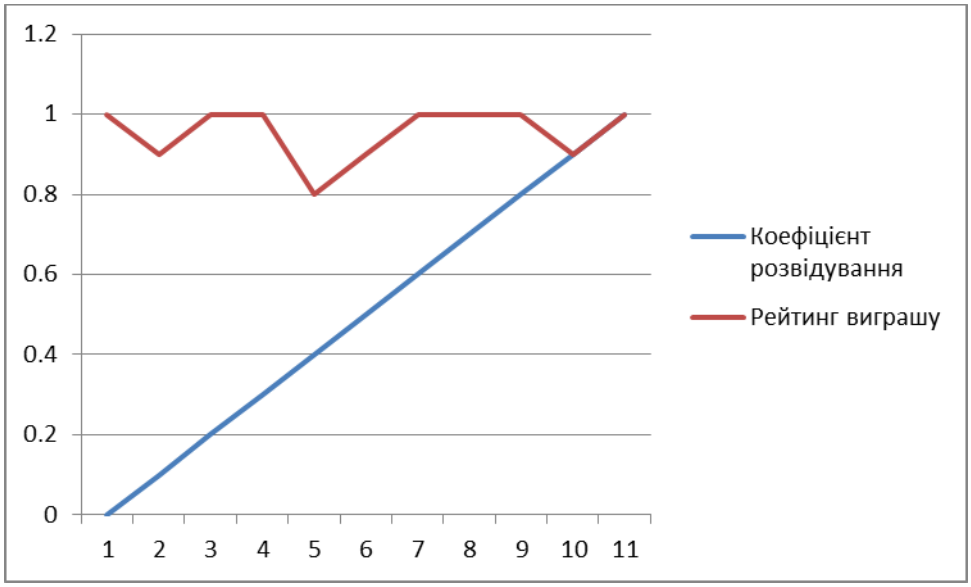

Рисунок 4. Залежність рейтингу виграшу від коефіцієнта розвідування (epsilon)

Така сама тенденція була отримана і при дослідженні коефіцієнта знецінювання (gamma). Було помічено, що результативність реалізованого рішення знаходиться на високому рівні: у найгіршому випадку 80% ігор завершуються перемогою головного героя. Ми робимо висновок, що дані параметри суттєво не впливають на коефіцієнт виграшу, або, можливо, зовсім не враховуються реалізованим алгоритмом. Нам слід перевірити програмний код, яким чином дані параметри беруть участь в роботі моделі та, можливо, виправити його. Можна стверджувати, що відсутність врахування заданих параметрів не сприяє вагомому негативному впливу на результати роботи.

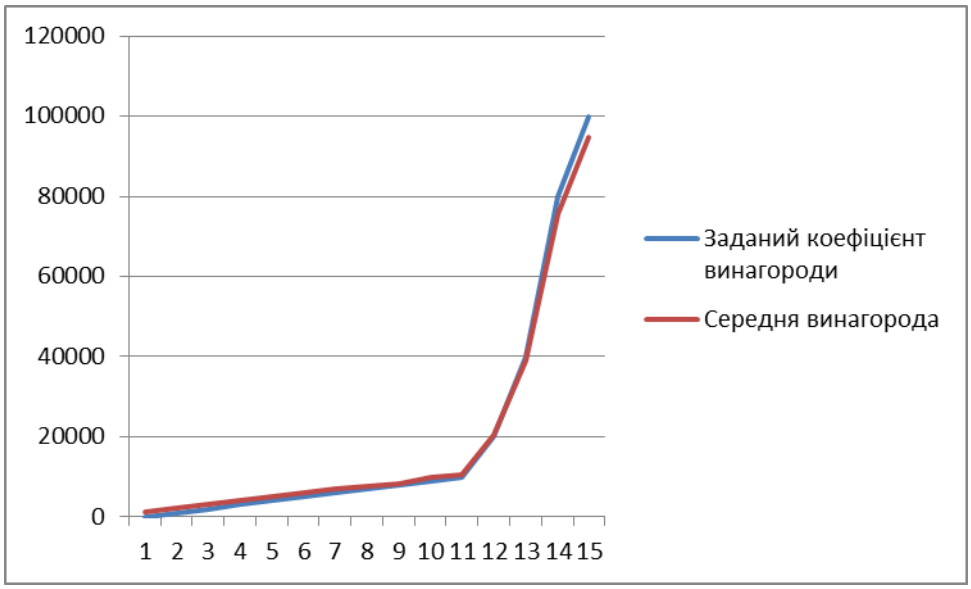

Рисунок 5. Залежність середньої винагороди від заданого коефіцієнта

З цього графіка видно, що отримана середня винагорода зростає разом із коефіцієнтом, який користувач задає в командному рядку. Це свідчить про правильність реалізації даної функціональної можливості програми.

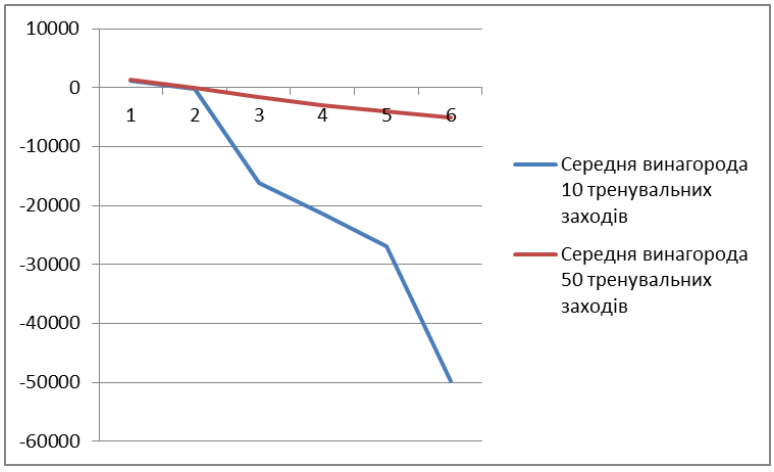

Рисунок 6. Вплив зменшення вартості руху на середню винагороду

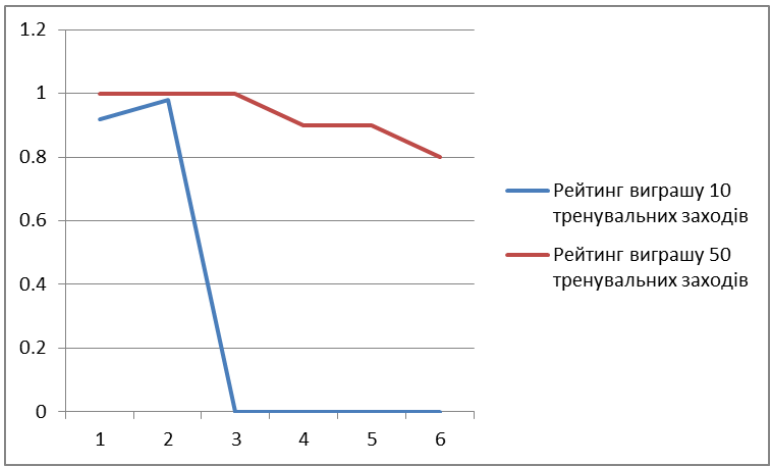

Рисунок 7. Вплив зменшення вартості руху на рейтинг виграшу

Було помічено, що зменшення вартості руху (збільшення кількості штрафних балів за циклічне топтання на місці) по-різному впливає на середню винагороду та процент виграшів. Після 50 тренувальних заходів процент виграшу, а, отже, й середня винагорода, значно більші. Якщо при тому самому значенні кількості штрафних балів було виконано 10 навчальних ітерацій, а 50 з них була реальними іграми, то вони у 100% випадків завершуються поразкою головного героя. Це говорить про важливість тренувань на певному етапі при експериментах із зниженням вартості руху.

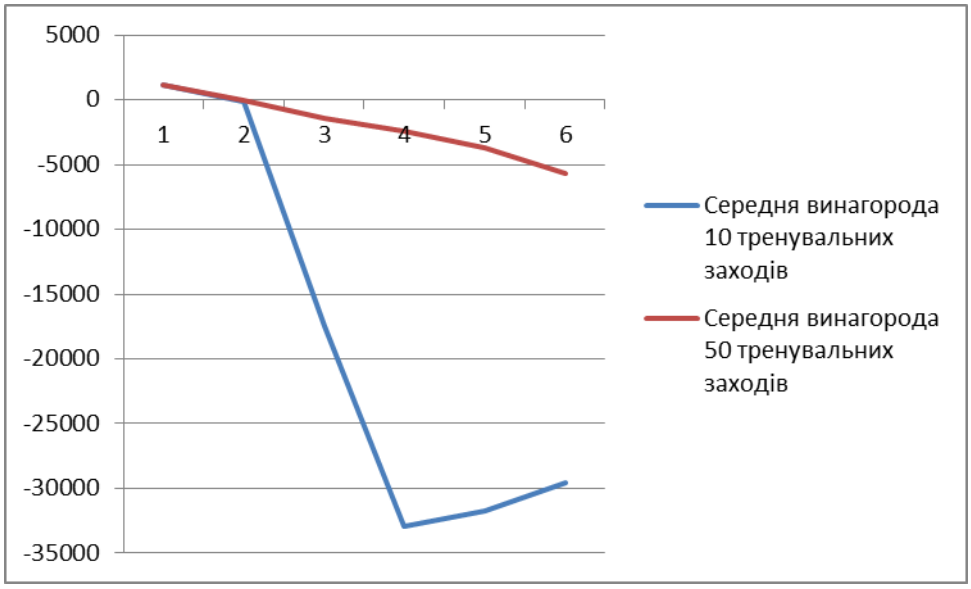

Рисунок 8. Вплив зростаючої різниці між штрафними балами та винагородою на середню винагороду

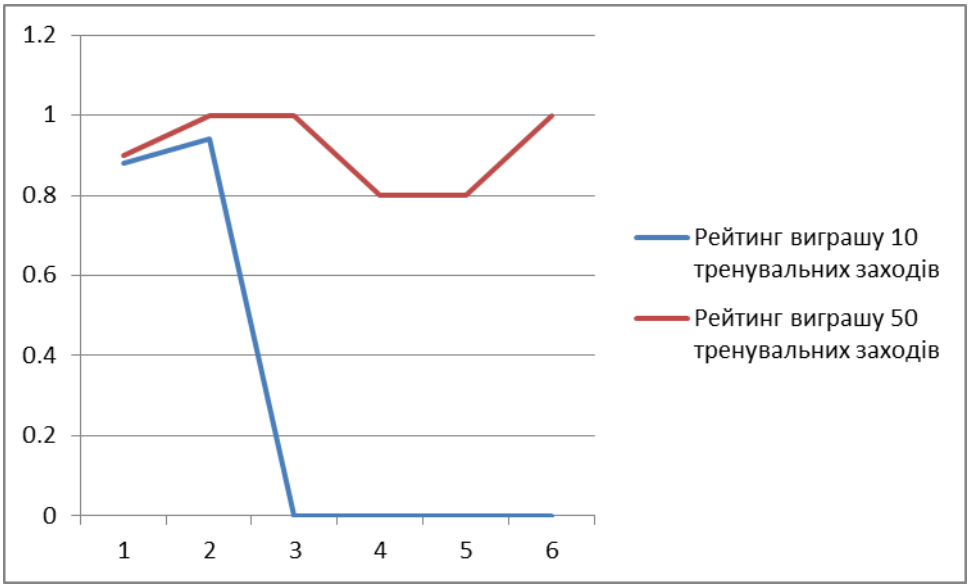

Рисунок 9. Вплив зростаючої різниці між штрафними балами та винагородою на рейтинг виграшу

На цих двох графіках ми бачимо підтвердження роботи попередньої тенденції: стрімке зменшення вартості руху при 10 тренувальних заходах призводить до повної відсутності виграних матчів. Проведення відносно великої кількості тренувань перед реальними іграми сприяє зберіганню кількості виграшів на високому рівні.

Задача дослідження впливу параметра алгоритму: результати серії експериментів з різними значеннями параметру, бажано у вигляді таблиці, графіку чи діаграми. Зробити висновки щодо впливу зміни параметрів.

Для отримання вищенаведених графіків ми побудували декілька таблиць, які виглядають наступним чином:

Кількість тренувальних заходів	Кількість реальних ігор	Рейтинг виграшу
80	10	0.9
70	20	0.75
60	30	0.87
50	40	0.93
40	50	0.96
30	60	0.85
20	70	0.97
10	80	0.91
5	85	0.92
0	90	0.87

Заданий коефіцієнт винагороди	Середня винагорода	Рейтинг виграшу
0	1312	0.92
10	2300	0.96
20	3250	0.92
30	4230	0.98
40	5053	0.94
50	5944	0.9
60	6862	0.92
70	7551	0.88
80	8242	0.84
90	9738	0.92
100	10379	0.86
200	20526	0.96
400	38923	0.9
800	75483	0.88
1000	94601	0.9

Заданий коефіцієнт винагороди	Середня винагорода 10 тренувальних заходів	Середня винагорода 50 тренувальних заходів	Рейтинг виграшу 10 тренувальних заходів	Рейтинг виграшу 50 тренувальних заходів
0	1248	1325	0.92	1
10	-130	-31.3	0.98	1
20	-16088	-1518	0	1
30	-21277	-2987	0	0.9
40	-26899	-3959	0	0.9
50	-49831	-5123	0	0.8

Різниця між штрафними балами та винагородою	Середня винагорода 10 тренувальних заходів	Середня винагорода 50 тренувальних заходів	Рейтинг виграшу 10 тренувальних зах	Рейтинг виграшу 50 тренувальних заходів
0	1157	1196	0.88	0.9
10	-94	-84	0.94	1
20	-17447	-1430	0	1
30	-32937	-2389	0	0.8
40	-31700	-3715	0	0.8
50	-29585	-5665	0	1

Рисунок 10. Таблиці для побудови графіків

Отже, обране нами середовище у вигляді гри Pacman моделює кімнату з вузькими проходами-дверми, а також надає широкі можливості для експериментів над алгоритмами машинного навчання. Нами була протестована одна з найпопулярніших моделей для автоматизації поведінки рухомих агентів в залежності від стану навколишнього середовища. У теоретичній частині даної роботи сказано, що результати Q-learning залежать від даних про навколишнє середовище. Це підтверджують проведені нами експерименти, зокрема те, що зменшення значення винагороди, а також вартості руху, призводить до майже повного унеможливлення отримати перемогу в грі. Ми бачили деяку залежність між кількістю тренувань та збільшенням кількості штрафних балів за хаотичне вагання: після 50 навчальних заходів при зменшенні вартості руху на 50 одиниць ми в більшості випадків спостерігали виграш. Але при суттєвому збільшенні параметра «penalty», наприклад, до 100 одиниць при тій самій кількості навчальних ітерацій ми бачимо, що в усіх випадках містер Пакмен програє. Це підтверджує коректність створеного коду: зменшення винагороди, а, отже, й вартості руху, призводить до зменшення ймовірності виграшу.

Алгоритм Q-learning має регулюватись деякими параметрами. При тестуванні ми не побачили їх впливу на результати гри. Тому ми вважаємо, що на поточному стані розробки дана програма має ряд недоліків, точну наявність яких слід встановити та виправити.