**Комп’ютерний практикум №3**

**Марківські процеси прийняття рішень. Навчання з підкріпленням.**

**ПІБ: Мєшков Андрій Ігорович, Ткач Владислав Анатолійович**

**Група: ІП-15**

**Мета роботи:** ознайомитись з методами пошуку в умовах невідомості та навчання з підкріпленням в моделях на основі станів; 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: 23**

**Завдання для варіанту:** 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 Value iteration. Виконати дослідження реалізованого методу - вплив коефіцієнта швидкості навчання α.

**Середовище:**

* **Стохастичність**: Лабіринт може мати стохастичність, якщо є можливість деякої випадковості у рухах агента
* **Стани**: Стани представлені координатами клітинок сітки, де кожна клітина може бути порожньою, містити стіну, агента, бомбу або золото. Агент може займати будь-яку клітину, яка не є стіною.
* **Дії**: Агент може рухатися ВГОРУ, ВНИЗ, ВЛІВО або ВПРАВО, якщо він не натрапить на стіну, в іншому випадку він залишається в тій самій клітині.
* **Переходи**: Переходи між станами залежать від вибраної дії. Якщо дія приводить агента до клітини зі стіною, він залишається в тій самій клітині. В іншому випадку він переміщується в сусідню клітину відповідно до вибраної дії.
* **Винагороди**: Винагороди попередньо визначені для певних станів:
  + -1 за звичайні порожні клітини.
  + 100 за клітину зі золотом.
  + -100 за клітину з бомбою.

Це середовище налаштоване для задач навчання з підсиленням, зокрема для ітерації вартості.

**Метод вирішення задачі:** Метод вирішення задачі Value Iteration є одним із найпоширеніших методів для навчання з підсиленням. Основна ідея полягає в тому, щоб ітеративно оновлювати оцінки значень кожного стану у середовищі, використовуючи принцип оптимальності Беллмана.

Ось характеристики методу Value Iteration:

- Офлайн стратегія: Value Iteration є офлайн стратегією, оскільки вона не вимагає взаємодії з середовищем під час навчання. Агент використовує інформацію про середовище, щоб оцінити оптимальні значення станів, не потребуючи активного взаємодії з ним.

- Дослідження середовища: Дослідження середовища необхідне для Value Iteration, але воно відбувається в офлайн режимі. Під час дослідження агент аналізує властивості середовища, такі як можливі переходи між станами і винагороди, для того щоб оцінити оптимальні значення станів.

- Реалізація дослідження середовища: У Value Iteration дослідження середовища зазвичай реалізується через ітерації алгоритму, де агент оновлює свої оцінки значень станів на кожному кроці. Цей процес включає аналіз доступних дій в кожному стані, переходів між станами, а також винагород, отриманих за ці переходи. Після дослідження агент використовує оновлені оцінки для вибору оптимальних дій у майбутньому.

**Реалізація методу:**

def iterate\_values(env, v, gamma, theta, p\_stoch):

converged = False

k = 0

sp = p\_stoch

p = {'UP': [sp + (1 - sp) / 4, (1 - sp) / 4, (1 - sp) / 4, (1 - sp) / 4],

'DOWN': [(1 - sp) / 4, sp + (1 - sp) / 4, (1 - sp) / 4, (1 - sp) / 4],

'LEFT': [(1 - sp) / 4, (1 - sp) / 4, sp + (1 - sp) / 4, (1 - sp) / 4],

'RIGHT': [(1 - sp) / 4, (1 - sp) / 4, (1 - sp) / 4, sp + (1 - sp) / 4]}

*while* not converged:

DELTA = 0

*for* i *in* range(0, env.height):

*for* j *in* range(0, env.width):

k += 1

*if* env.check\_state((i,j)) or (i,j) in env.walls:

v[i,j] = 0

*else*:

old\_v = v[i,j]

new\_v = []

*for* action *in* env.actions:

new\_v\_p = []

*for* idx, action\_p *in* enumerate(env.actions):

((n\_i, n\_j), reward) = env.P.get(((i,j), action\_p))

new\_v\_p.append(p.get(action)[idx] \* (reward + (gamma \* v[n\_i, n\_j])))

new\_v.append(sum(new\_v\_p))

v[i,j] = max(new\_v)

DELTA = max(DELTA, np.abs(old\_v - v[i,j]))

converged = True *if* DELTA < theta *else* False

print(k, 'iterations of state space')

*return* v

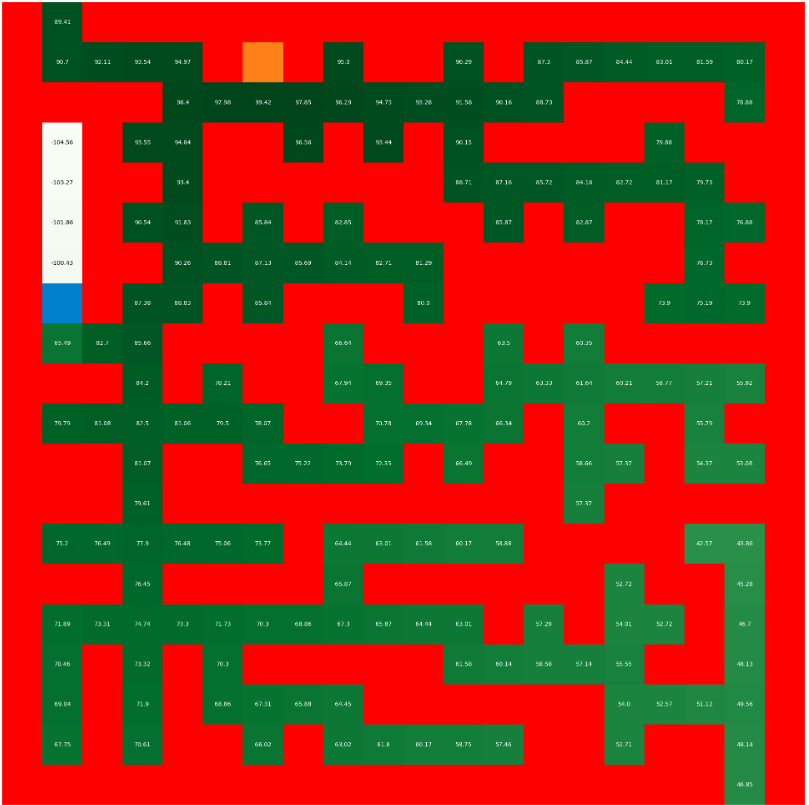
Детальнішу реалізацію можна переглянути за посиланням:

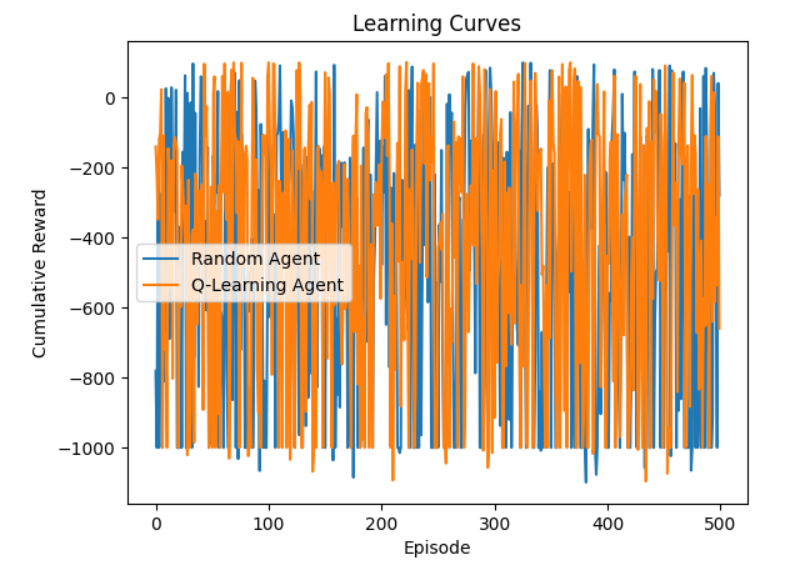
<https://github.com/IP15-MieshkovAndrii/AI-3-4-5/tree/main/lab3>

**Результати застосування розробленого методу:**

Було створено середовище лабіринту розміром 20x20 з випадковим розташуванням стін, золота та бомб.

Приклад лабіринту:





Агент розпочав свою подорож у лабіринті з початкової позиції (15, 3). За допомогою методу Value Iteration було знайдено оптимальну стратегію для агента, що дозволяє уникати бомб та досягати золота.

**Оцінка результатів:**

В результаті виконання 49200 ітерацій методу Value Iteration було досягнуто збіжності, що дозволило агенту ефективно знаходити шлях до золота і уникати небезпек. Аналіз лабіринту показує, що агент зміг навчитися оптимальній стратегії, що підтверджується високими значеннями винагород за успішні епізоди.

Для порівняння з методом Value Iteration було також реалізовано Q-агента, який використовує алгоритм Q-learning для навчанн, мета якого полягає в тому, щоб оцінити ефективність цього онлайн-методу навчання у порівнянні з офлайн-методом Value Iteration.

class QLearningAgent:  
 def \_\_init\_\_(self, alpha, gamma, epsilon):  
 self.alpha = alpha  
 self.gamma = gamma  
 self.epsilon = epsilon  
 self.q\_table = {}  
  
 def initialize\_q\_table(self, env):  
 for state in env.state\_space:  
 for action in env.actions:  
 self.q\_table[(state, action)] = 0.0  
  
 def choose\_action(self, env):  
 current\_state = env.current\_location  
 if np.random.uniform(0, 1) < self.epsilon:  
 return np.random.choice(env.actions)  
 else:  
 q\_values = []  
 for action in env.actions:  
 if (current\_state, action) not in self.q\_table:  
 self.q\_table[(current\_state, action)] = 0.0  
 q\_values.append(self.q\_table[(current\_state, action)])  
 max\_q\_value = max(q\_values)  
 return env.actions[q\_values.index(max\_q\_value)]  
  
 def learn(self, old\_state, reward, new\_state, action):  
 for a in env.actions:  
 if (new\_state, a) not in self.q\_table:  
 self.q\_table[(new\_state, a)] = 0.0  
 if (old\_state, a) not in self.q\_table:  
 self.q\_table[(old\_state, a)] = 0.0  
 max\_future\_q = max([self.q\_table[(new\_state, a)] for a in env.actions])  
 current\_q = self.q\_table[(old\_state, action)]  
 self.q\_table[(old\_state, action)] = current\_q + self.alpha \* (reward + self.gamma \* max\_future\_q - current\_q)

Q-агент швидко адаптується до середовища, але його навчання може бути менш стабільним порівняно з Value Iteration.

Результати, що наведені нижче, показують, що Q-learning може досягти кращих результатів у короткостроковій перспективі, але Value Iteration забезпечує стабільніше і більш надійне навчання.

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Середнє значення винагороди** | **Стандартне відхилення** | **Медіана** |
| Value Iteration | -450.24 | 367.05 | -346.00 |
| Q-learning | -423.55 | 345.60 | -360.50 |

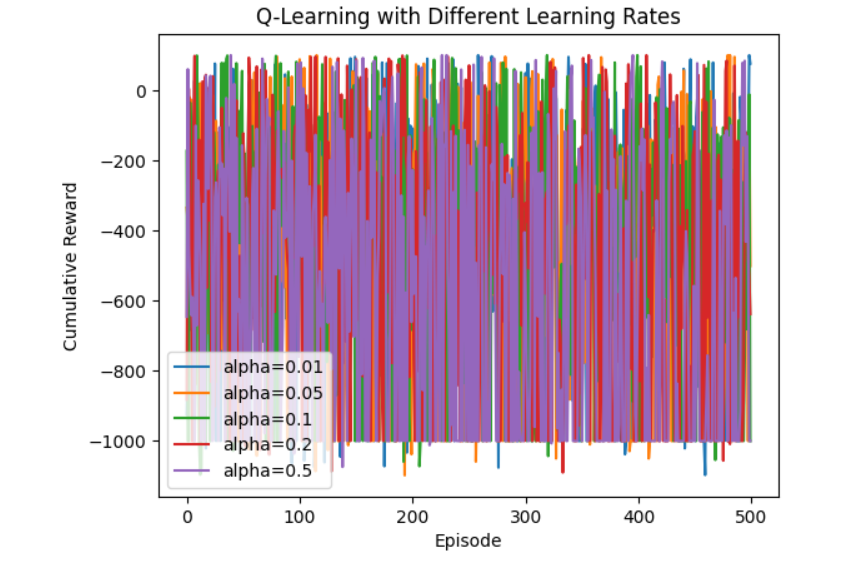
**Задача дослідження впливу параметра алгоритму:**

Задача дослідження полягала у визначенні впливу значення коефіцієна швидкості навчання α на ефективність акгента.

Для кожного значення α було проведено по 500 епізодів навчання під час яких оцінювалась кумулятивна винагорода за кожен епізод та обраховано статистичні показники:

Випробуванні значення та результати наведено нижче:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **alpha** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| 0.01 | -450.24 | 367.05 | -1095.00 | -801.25 | -346.00 | -144.75 | 99.00 |
| 0.05 | -423.55 | 345.60 | -1067.00 | -689.00 | -360.50 | -139.75 | 100.00 |
| 0.1 | -450.19 | 362.87 | -1095.00 | -782.00 | -377.00 | -140.75 | 100.00 |
| 0.2 | -481.65 | 375.90 | -1098.00 | -859.25 | -430.00 | -152.50 | 100.00 |
| 0.5 | -549.15 | 381.58 | -1077.00 | -1000.00 | -500.00 | -204.75 | 100.00 |



Найбільш стабільне та ефективне навчання спостерігалося на α = 0,05 а от високі значення даного параметра призводять до гірших результатів через надмірну адаптацію до поточних винагород але в свою чергу низькі значення α можуть призвести до повільного навчання.

Отже найкращим значення можна уважати 0,05, воно забезпечує найкращий компроміс між швидкістю навчання та стабільністю роботи.

**Висновок:**

У результаті виконання комп'ютерного практикуму було розглянуто два методи навчання з підкріпленням: Value Iteration та Q-learning.

Value Iteration є офлайн методом, який не вимагає безпосередньої взаємодії з середовищем під час навчання, тоді як Q-learning є онлайн методом, який оновлює свої оцінки під час взаємодії агента зі середовищем.

Для наочного результату було створено середовище лабіринту розміром 20x20 з випадковим розташуванням стін, золота та бомб. Агент успішно навчався в цьому середовищі, використовуючи метод Value Iteration, щоб уникати небезпек і досягати мети, і це метод показав стабільні результати після 49200 ітерацій, дозволяючи агенту ефективно знаходити шлях до золота і уникати бомб, а аналіз результатів підтвердив, що агент зміг навчитися оптимальній стратегії в середовищі лабіринту.

Для порівняння було реалізовано Q-агента, який використовує алгоритм Q-learning для навчання. Результати показали, що Q-learning може досягти кращих результатів у короткостроковій перспективі, але метод Value Iteration забезпечує більш стабільне і надійне навчання.

Також було проведено дослідження впливу коефіцієнта швидкості навчання α на ефективність агента. Найбільш стабільне та ефективне навчання спостерігалося при α = 0.05, тоді як високі значення α призводили до гірших результатів через надмірну адаптацію до поточних винагород, а дуже низькі значення α могли призвести до повільного навчання. Значення α = 0.05 було визнано найкращим компромісом між швидкістю навчання та стабільністю.