Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №6 з дисципліни
«Програмування інтелектуальних інформаційних систем»

Виконав(ла) *ІП-11 Прищепа В.С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Баришич Л. М.* _____
(прізвище, ім'я, по батькові)

Київ 2023

Завдання

1. Розробити ітеративного агента, що проходить заданий грід за допомогою q-learning.

2. Пройти “міст”.

3. Оптимізувати агента, змінюючи початкові політики, щоб він ефективно проходив грід п'ятьма різними шляхами, враховуючи ризики та нагороди.

4. оповнивши Q-агента, виконати реалізацію вибору дій з епсилон- жадібністю.

Хід роботи:

1. Розробити ітеративного агента, що проходить заданий грід за допомогою q-learning.

Для цього імплементовано функцію runValIter для оновлення станів згідно Q-значень:

```
def runValIter(self):
        for iter in range(self.iters):
                temp = util.Counter()
                for state in self.mdp.getStates():
                        if self.mdp.isTerm(state):
                                temp[state] = 0
                        else:
                                maxVal = -99999
                                acts = self.mdp.getPosAct(state)
                                for act in acts:
                                        t = self.mdp.getTransStatesAndProbs(state, action)
                                        val = 0
                                        for stateProb in t:
                                                val += stateProb[1] *
(self.mdp.getReward(state,  action, stateAndProb[1]) + self.discount *
self.vals[stateAndProb[0]])
                                        maxVal = max(val, maxVal)
                                if maxVal != -99999:
```

```
                temp[state] = maxVal
        self.vals = temp
```

Q-значення розраховуються за формулою:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

```
def compQValFromVals(self, state, action):
    qval = 0
    for nextState, prob in self.mdp.getTransStatesAndProbs(state, action):
        qval += prob * (self.mdp.getReward(state, action, nextState) +
self.discount * self.vals[nextState])
    return qval
```

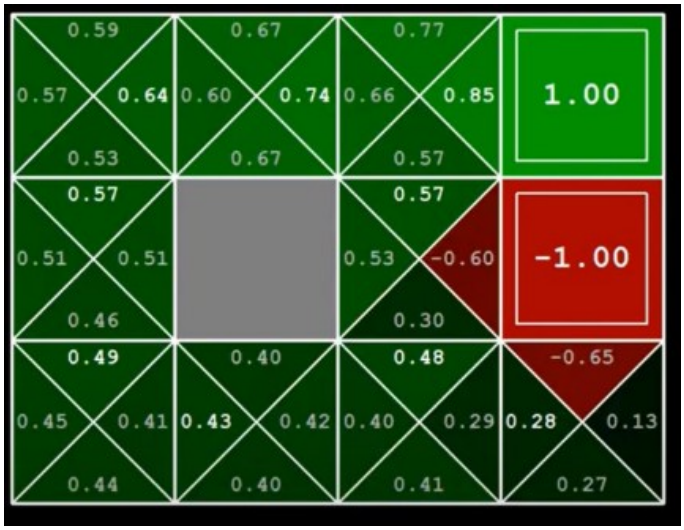
Дії на основі Q-значень розраховуються за даною функцією:

```
def compActFromVals(self, state):
    if self.mdp.isTerm(state):
        return None
    acts = self.mdp.getPosAct(state)
    allActs = {}
    for act in acts:
        allActs[act] = self.compQValFromVals(state, act)
    return max(allActs, key=allActs.get)
```

Далі запускаю програму через термінал командою:

```
python gridworld.py -a value -i 100 -k 10
```

Отримані значення після 100 ітерацій

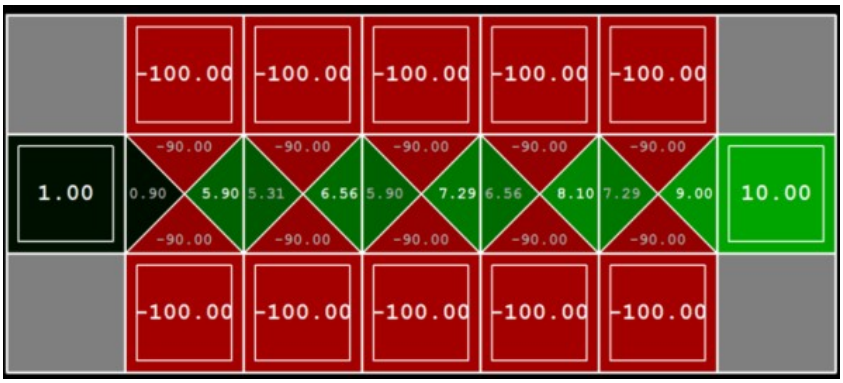


2. Застосувати готовий модуль від Берклі, змінивши один параметр, щоб агент успішно перейшов "міст".

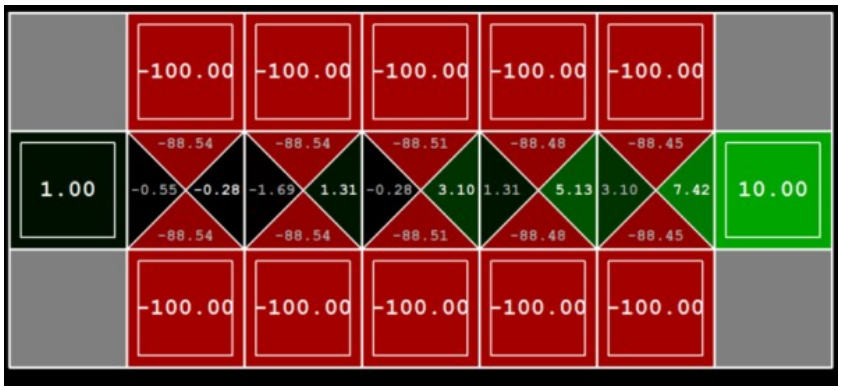
Для того, щоб агент успішно пройшов міст, потрібно змінити параметр шуму, аби дії агенту були більш детерміновані. Допустимий інтервал значень шуму – [0, 0.016] (виявлено евристично), discount – 0.9. Команда запуску в терміналі:

python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 –noise 0

Для шуму 0



Для шуму 0.016



3. Оптимізувати агента, змінюючи початкові політики, щоб він ефективно проходив грід п'ятьма різними шляхами, враховуючи ризики та нагороди.

Команда запуску в терміналі:

python gridworld.py -a value -I 100 -g DiscountGrid –discount 0.1 –noise 0.1 –livingReward 100

Далі наведено функцію з параметрами для реалізації цієї логіки. Віддаєте перевагу близькому виходу (+1), ризикуючи зірватися з обриву (-10)

```
def question3a():
    answerDiscount = .1
    answerNoise = 0
    answerLivingReward = -4
    return answerDiscount, answerNoise, answerLivingReward
```

Віддаєте перевагу близькому виходу (+1), але уникаєте обриву (- 10)

```
def question3b():
    answerDiscount = .1
    answerNoise = .1
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

Віддаєте перевагу дальньому виходу (+10), ризикуючи зірватися з обриву (-10)

```
def question3c():
    answerDiscount = 1
    answerNoise = 0
    answerLivingReward = -1
    return answerDiscount, answerNoise, answerLivingReward
```

Віддаєте перевагу дальньому виходу (+10), уникаючи обриву (- 10)

```
def question3d():
    answerDiscount = 1
    answerNoise = .1
    answerLivingReward = -0.5
    return answerDiscount, answerNoise, answerLivingReward
```

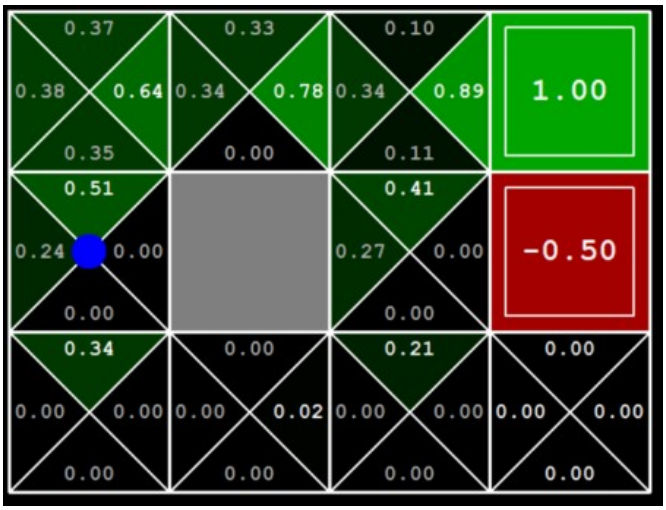
Уникаєте обох виходів і обриву (тобто епізод ніколи не повинен завершуватися)

```
def question3e():
    answerDiscount = 1
    answerNoise = .1
    answerLivingReward = 100
    return answerDiscount, answerNoise, answerLivingReward
```

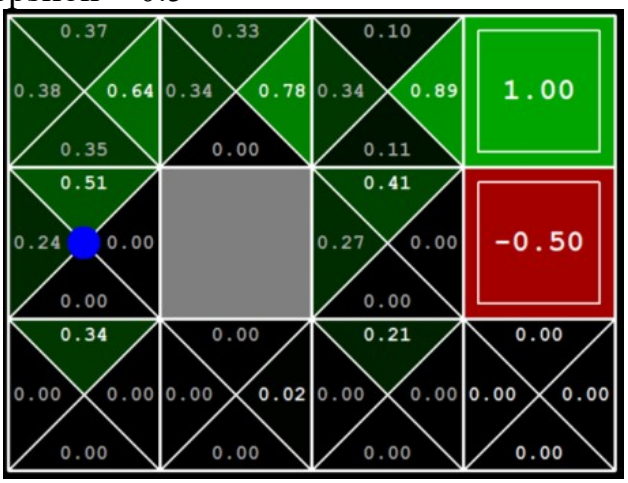
4. Доповнивши Q-агента, виконати реалізацію вибору дій з епсилон-жадібністю. Після доповнення коду Q-агента, можна запустити наступну команду в терміналі:

python gridworld.py -a q -k 100 –noise 0.0 -e 0.5

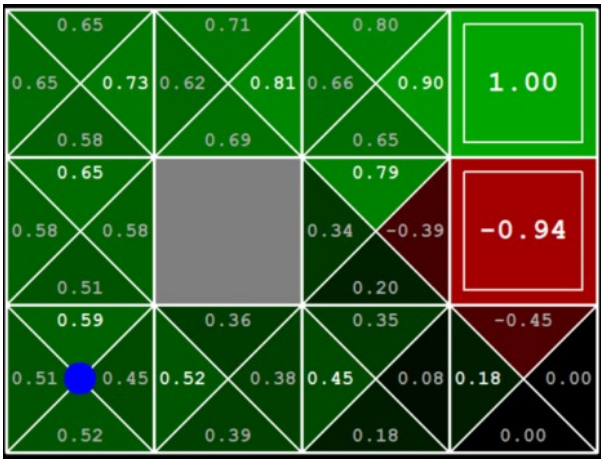
epsilon = 0.1



epsilon = 0.5



epsilon = 0.9



Висновок: Під час виконання цієї лабораторної роботи я розробив та оптимізував ітеративного агента за допомогою Q-learning для ефективного проходження визначеного гріду. Шляхом внесення змін у параметри готового модулю від Берклі, я успішно налаштував агента на подолання викликів, зокрема, переходів через "міст". Оптимізація агента включала зміну початкових політик, що дозволило йому ефективно проходити грід різними шляхами, враховуючи потенційні ризики та отримані нагороди. Результати свідчать про успішну адаптацію агента до середовища та його здатність вдосконалювати стратегії на основі набутого досвіду.