ПРОЕКТ 3: ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

Pacman ищет вознаграждение.
Съесть или убежать?
Если сомневаешься, используй Q-обучение.

ВВЕДЕНИЕ

В этом проекте вы примените функцию полезности и q–обучение. Вы будете тестировать своих агентов в первую очередь на GridWorld (в классе), а затем примените их к смоделированному управляющему роботу (Crawler) и Pacman.

Как и в предыдущих проектах, этот проект включает в себя систему автоматического оценивания (автогрейдер), которая позволяет вам оценивать ваши решения на вашем компьютере. Он может быть запущен для всех вопросов с помощью команды:

```
python autograder.py
```

Он так же может быть запущен для определенного вопроса, например q2, с помощью команды:

```
python autograder.py -q q2
```

А для одного определенного теста, с помощью команд вида:

```
python autograder.py -t test_cases/q2/1-bridge-grid
```

Для того чтобы более подробнее ознакомиться с автогрейдером, смотрите обучение в проекте 0.

Код данного проекта состоит из ниже перечисленных файлов, которые расположены в файле zip archive:

Файлы, которые вы будете изменять:	
valueIterationAgents.py	Функция полезности агента для ситуаций, представленных в виде Марковского процесса принятия решений (МППР).
qlearningAgents.py	Агенты Q-обучения для Gridworld, Crawler и Pacman.
analysis.py	Файл для сохранения ваших ответов на вопросы проекта.
Файлы которые вы должны прочесть, но НЕ ДОЛЖНЫ изменять:	
mdp.py	Определения основных методов МППР.
learningAgents.py	Определение основных классов `ValueEstimationAgent` и `QLearningAgent`, которые ваши агенты будут расширять.
util.py	Утилиты, включая `util.Counter`, используемый для Q-обучения.

gridworld.py	Файлы установки Gridworld.
featureExtractors.py	Классы для разделения характеристик на пары (состояние. действие). Используется в основном для агента приближенного Q-обучения (в qlearningAgents.py).
Файлы, которые вы можете проигнорировать:	
environment.py	Абстрактный класс для общих условий для обучения с подкреплением. Используется командой gridworld.py.
graphicsGridworldDisplay.py	Графический дисплей Gridworld.
graphicsUtils.py	Графические утилиты.
textGridworldDisplay.py	Модуль для текстового интерфейса Gridworld.
crawler.py	Код Crawler и тестовая программа. Вы будете запускать этот файл, но не будете его редактировать.
graphicsCrawlerDisplay.py	Графический интерфейс для Crawler.

autograder.py	Автогрейдер.
testParser.py	Разбирает файлы тестов и решений автогрейдера.
testClasses.py	Основные классы теста автогрейдера.
`test_cases/`	Каталог, содержащий тестовые задачи на каждый вопрос.
reinforcementTestClasses.py	Тестовые классы Проекта 3 для автогрейдера.

Файлы для редактирования и добавления: Вы будете заполнять части команд valueIterationAgents.py, qlearningAgents.py, и analysis.py в течении всего задания. Вы должны передать эти файлы с вашим кодом и комментариями. Пожалуйста, *не изменяйте* другие файлы во время предоставления и не передавайте никакие другие исходные файлы, кроме указанных трех.

Оценка: Ваш код будет автоматически оценен на техническую корректность. Пожалуйста, *не изменяйте* имена, представляемых функций или классов внутри кода, иначе вы внесете неопределенность в работу автогрейдера. Тем не менее, правильность вашей реализации (а не оценка автогрейдера) – будет решающей при получении вами оценки. При необходимости, мы рассмотрим и оценим задание индивидуально, чтобы удостовериться, что вы получите заслуженную оценку за вашу работу.

Плагиат: Мы будем сравнивать ваш код с другими представленными заданиями на логическую избыточность. Если вы скопировали, чей-либо

чужой код и предоставили его с небольшим изменением, мы это узнаем. Обмануть данную проверку практически невозможно, пожалуйста, даже не пытайтесь. Мы вам полностью доверяем и считаем, что вы предоставите свою личную работу, *пожалуйста*, не разочаруйте нас. Но если вы смухлюете, мы применим к вам самые строжайшие меры, которые нам доступны.

Помощь: Вы не одиноки! Если вы обнаружите, что вы застряли на чем-то, обратитесь за помощью к преподавателям курса. Приёмные часы, секция и обсуждения на форуме существуют для помощи вам; пожалуйста, используете их. Если вас не устраивают наши часы работы, сообщите нам об этом и мы найдем для вас время. Мы хотим, чтобы эти занятия были ценными и познавательными, а не разочаровывающими и не приносящими знаний. Но, мы не знаем, где и когда вам помочь, пока вы к нам не обратитесь.

Обсуждение: Пожалуйста, будьте осторожны, не оставляйте подсказок.

Марковский процесс принятия решений (МППР)

Чтобы начать работу, запустите Gridworld в режиме ручного управления, в котором используются клавиши со стрелками:

```
python gridworld.py -m
```

Вы увидите уровень с двумя выходами. Синяя точка - это агент. Обратите внимание, что когда вы нажмете клавишу *вверх,* агент будет двигается на север только 80% времени. Такова жизнь Gridworld агента!

Вы можете управлять многими аспектами моделирования. Для ознакомления с полным листом доступных опций запустите команду:

```
python gridworld.py -h
```

По умолчанию агент двигается случайным образом

```
python gridworld.py -g MazeGrid
```

Вы должны заметить, что агент случайным образом прыгает внутри сетки, до тех пока не достигнет выхода. Не лучшее время препровождения для агента с искусственным интеллектом.

Обратите внимание: Gridworld МППР таков, что изначально вы должны войти в предварительное конечное состояние (двойные клетки, показанные в графическом интерфейсе) и только затем перейти к определенному «выходу» до того, как процесс закончится (в истинное конечное состояние `TERMINAL STATE`, которое не показано в графическом интерфейсе пользователя). Если вы запустите процесс вручную, ваш общий результат может быть меньше чем вы ожидаете, из-за ставки дисконтирования (`-d` чтобы изменить; 0.9 по умолчанию).

Посмотрите на консоль вывода, которая сопровождает графическое представление (или используйте `-t` для просмотра всего текста). Вам сообщат о каждом переходе и действии агента (чтобы выключить используйте `-q`).

Как и в Pacman, местоположение определяется с помощью значений `(x,y)` декартовой системы координат и любых массивов `[x][y]`, где возрастание координаты `y` является направлением `'север'`, и т.д. По умолчанию, большинство перемещений приносит нулевое

вознаграждение, хотя вы можете изменить это с помощью вознаграждение за существование (`-r`).

Вопрос 1 (6 баллов): Функция полезности

Напишите агент для формирования функции полезности в `ValueIterationAgent`, которая была частична определена для вас в `valueIterationAgents.py`. Ваш агент представляет собой автономный планировщик, а не обучаемый с подкреплением агент, и, таким образом, необходимым обучением является набор итераций функции полезности, которая должна запускаться (опция `-i`) на начальном этапе планирования. `ValueIterationAgent` берет на себя конструирование МППР и запускает функцию полезности на определенное количество итераций, пока завершит работу конструктор.

Функция полезности вычисляет оценки оптимальных значений шкалы k-шага, V_k. В дополнение к запущенной функции полезности, обеспечьте выполнение следующих методов для `ValueIterationAgent`, используя V_k:

- `computeActionFromValues(state)` вычисляет наилучшее действие исходя из значения функции, выдаваемой `self.values`.
- `computeQValueFromValues(state, action)` возвращает значение для пары Q-значения (состояние, действие), полученной из значения функции `self.values`.

Их величины представлены в графическом интерфейсе: значения – это числа в квадратах, Q-значения– числа в четвертях квадратов, а маршрут – это стрелки исходящие из каждого квадрата.

Важно: Используйте «пакетную» версию функции полезности, в которой каждый вектор V_k вычисляется из фиксированного вектора V_{k-1}(как на лекции), а не «онлайн» версию, где единичный вектор обновляется на месте. Это значит, что, когда положение обновляется в итерации k на основе значений предыдущих положений, предыдущие состояния значения, используемые для получения нового значения, должны быть теми же, что были на итерации k-1 (даже если некоторые предыдущие состояния уже были обновлены в итерации k). Различия изложены в книге Sutton & Barto в шестом параграфе главы 4.1.

Обратите внимание: Маршрут, полученный из значений глубины k (отражает следующие k вознаграждений) будет на самом деле отражать следующее k+1 вознаграждений (т.е. вы возвращаете $\pi k+1$). Аналогично,

Q- значения будут так же отражать на одно больше вознаграждение, чем при обычных значениях (т.е. вы получите Q_{k+1}).
Вы должны получить маршрут $\pi k+1$.

Подсказка: Используйте класс `util.Counter` в `util.py`, который является словарем со значением по умолчанию равным 0. Такие методы, как `totalCount` должны упростить ваш код. Тем не менее, будьте осторожны с `argMax`: `argMax` который вам нужен, может быть ключом в счетчике!

Обратите внимание: Убедитесь, что вы можете справляться с задачами, когда состояние не предполагает доступных действий в МППР (подумайте о том, что это может значить для будущих вознаграждений).

Чтобы протестировать вашу реализацию запустите автогрейдер:

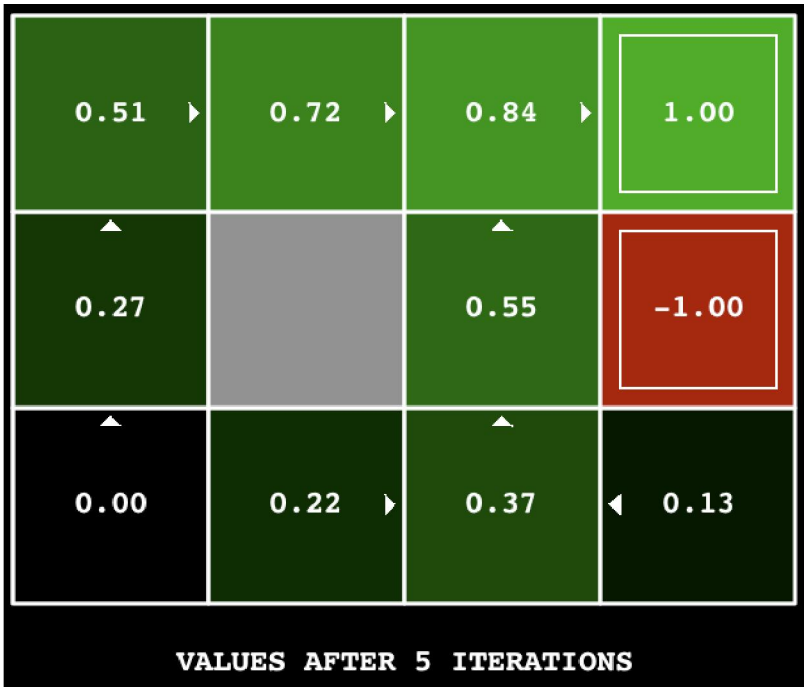
```
python autograder.py -q q1
```

Следующая команда загружает ваш `ValueIterationAgent`, который вычислит маршрут и выполнит его 10 раз. Нажмите на клавишу для переключения между значениями, Q-значениями и моделированием. Вы обнаружите, что значения начального состояния (`V(start)`, которое можно считать в графическом интерфейсе) и, полученное в результате опытов, среднее вознаграждение (показанное после 10 раундов) достаточно близки.

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

Подсказка: По умолчанию BookGrid, запускающий функцию полезности, для 5 итерации, должен дать вам следующий результат:

```
python gridworld.py -a value -i 5
```



Система оценивания: Ваш агент для формирования функции полезности будет оцениваться на новой сетке. Мы проверим ваши значения, Q-значения и маршрут, после фиксированного числа итераций (например, после 100 итераций) и проведем проверку на сходимость.

Вопрос 2 (1 балл): Анализ пересечения моста

`BridgeGrid` представляет собой сетчатую карту мира с двумя конечными состояниями – одно с минимальным вознаграждением, второе с максимальным, разделенными узким “мостом”, с каждой стороны которого находится пропасть с отрицательными вознаграждениями большой величины. Агент начинает движение от состояния с минимальным вознаграждением. При стандартном значении дисконтирования равном 0,9 и стандартном уровне шума равном 0,2 оптимальный маршрут заключается в том, чтобы не пересекать мост. Измените только ОДИН из параметров дисконтирования или шума, чтобы изменить оптимальный маршрут и заставить агента пересечь мост. Запишите свой ответ в `question2()` в `analysis.py` (шум означает то, как часто агент оказывается в незапланированном последующем состоянии при совершении действий). Использование стандартных значений соответствует:

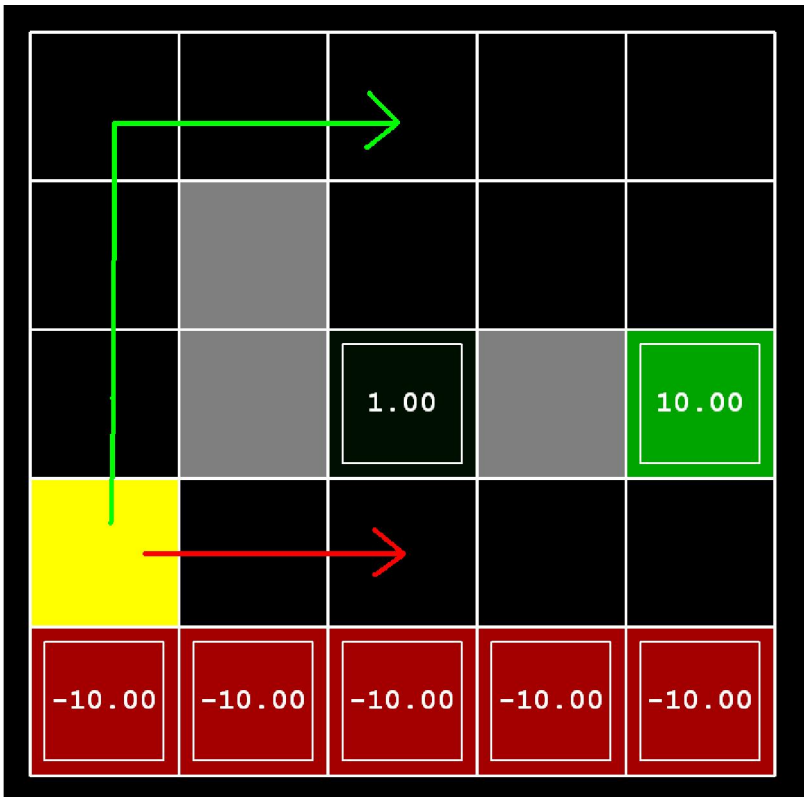
```
python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9
--noise 0.2
```

Система оценивания: Мы проверим, изменили ли вы только один из заданных параметров, и что с этим изменением правильный агент функции полезности пересечет мост. Для проверки вашего ответа, запустите автогрейдер:

```
python autograder.py -q q2
```

Вопрос 3 (5 баллов): Маршруты

Рассмотрите схему DiscountGrid, показанную ниже. Данная сетка имеет два конечных состояния с положительным выигрышем (средняя линия), ближайший выход с выигрышем +1 и дальний выход с выигрышем +10. Нижняя линия сетки состоит из конечных состояний с отрицательным выигрышем (выделена красным). Каждое состояние в данной “пропасти” имеет выигрыш -10. Начальное состояние находится в желтом квадрате. Мы выделяем два направления пути: (1) путь, который “рискует” и проходит рядом с нижней строкой сетки; этот путь короче, но вы рискуете получить большой отрицательный выигрыш (красная стрелка на рисунке); (2) путь “минует пропасть” и проходит по верхнему краю сетки. Такой путь длиннее, но вы с меньшей вероятностью приведет к отрицательному выигрышу (зеленая стрелка на рисунке).



В этом вопросе вы выберете настройки дисконтирования, шума и вознаграждения за существование для данного МППР для выработки оптимальных маршрутов различных видов. Ваши настройки значений параметров должны иметь свойство, что, если ваш агент примет решение использовать свой оптимальный маршрут и не подвергнется воздействию шумов, он покажет заданное поведение. Если конкретное поведение не

может быть достигнуто для любой настройки параметров, укажите, что маршрут невозможен, вернув строку “NOT POSSIBLE”.

Ниже представлены оптимальные стратегии, которые вы должны попытаться создать:

1. предпочесть ближайший выход (+1), пройти рядом с пропастью (-10)

2. предпочесть ближайший выход (+1), избегая пропасть (-10)

3. предпочесть дальний выход (+10), пройти рядом с пропастью (-10)

4. предпочесть дальний выход (+10), избегая пропасть (-10)

5. избежать обоих выходов и пропасти (чтобы процесс никогда не закончился)

Для проверки вашего ответа запустите автогрейдер:

```
python autograder.py -q q3
```

`question3a()` через `question3e()` должен вернуть трёхзначный набор (значения для дисконтирования, шума, вознаграждения за существование) в `analysis.py`

Примечание: вы можете проверить ваши маршруты в графическом интерфейсе. Например, при использовании верного ответа в 3(a) стрелка (0,1) должна показать на восток, стрелка (1,1) также должна показать на восток, а стрелка (2,1) должна показать на север.

Примечание: на некоторых компьютерах стрелка не видна. В таком случае, нажмите клавишу на клавиатуре для перехода к экрану qValue, и в уме просчитайте стратегию, взяв среднее максимальное значение доступных qValue для каждого состояния.

Система оценивания: Мы проверим, что нужный маршрут будет получен в каждом случае.

Вопрос 4 (5 баллов): Q-обучение

Учтите, что ваш агент для формирования функции полезности на самом деле не учится на собственном опыте. Наоборот, он обдумывает свою модель МППР для построения полного маршрута до взаимодействия с окружающей средой. При взаимодействии с окружающей средой, он следует заранее полученному маршруту (например, становится отражающим агентом). Это различие может быть незначительным в такой искусственной среде как Gridworld, но имеет большое значение в реальном мире, где МППР не доступен.

Приступим к написанию кода агента с Q-обучением, который делает очень мало на стадии построения, а вместо этого учится на испытаниях и ошибках при взаимодействии с окружающей средой через свой метод `update(state, action, nextState, reward)`. Подпрограмма Q-обучения указана в `QLearningAgent` в `qlearningAgents.py`, и вы можете выбрать ее с помощью команды `'-a q'`. В этом вопросе вы должны применить методы `update`, `computeValueFromQValues`,`getQValue` и `computeActionFromQValues`

Примечание: Для `computeActionFromQValues` вы должны случайным образом разорвать связи для лучшего поведения. В этом поможет функция `random.choice()`. В определенном случае, действия, которые ваш агент *еще не видел* все еще имеют Q-значение, в особенности Q-значение равное нулю, и, если все действия, которые ваш агент *уже видел* имеют отрицательные Q-значения, незнакомые действия могут быть оптимальными.

Важно: убедитесь, что в ваших функциях `computeValueFromQValues` и `computeActionFromQValues` вы получаете доступ к Q-значениям только при вызове `getQValue`. Данная абстракция будет полезной в 8 вопросе при перезаписи `getQValue` для использования свойств пар состояние-действие, а не их использования напрямую.

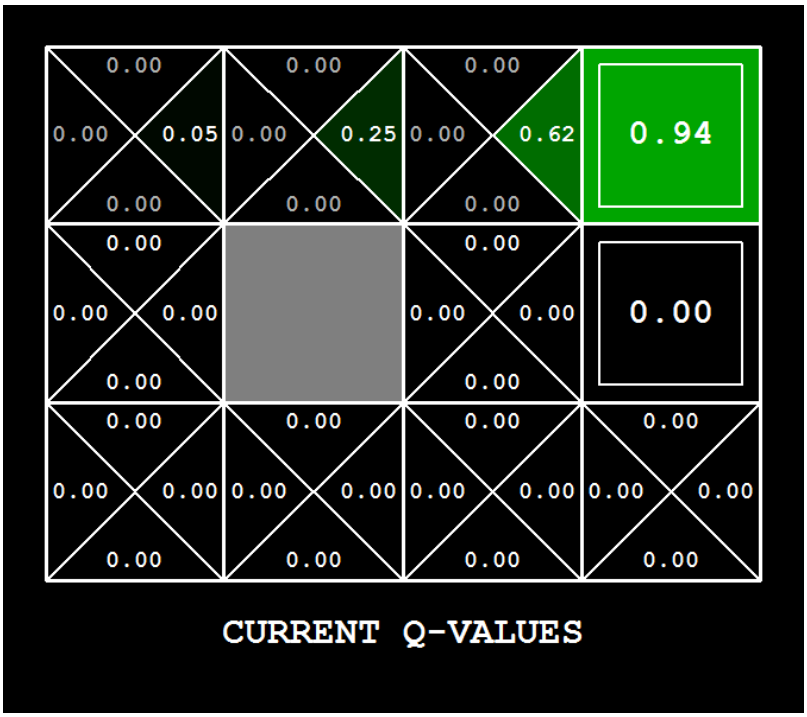
При обновлении Q-обучения на месте, вы можете наблюдать как ваш агент учится при ручном управлении с использованием клавиатуры:

```
python gridworld.py -a q -k 5 -m
```

Напомним, что `-k` будет контролировать число эпизодов обучения вашего агент. Наблюдайте как агент узнает о состоянии, в котором он только что

был, а не о том, к которому он идет, и “оставляет обучение как оно есть”.

Подсказка: для помощи в отладке, вы можете отключить шум используя параметр `--noise 0.0` (хотя это очевидно делает Q-обучение менее интересным). Если вы вручную направите Pacman на север, а затем на запад по оптимальному пути для четырех эпизодов, вы увидите следующие Q-значения:



Система оценивания: Мы запустим агента Q-обучения и проверим, что он узнает те же Q-значения и маршруты, которые указаны в базовой реализации, при условии, что каждая переменная и стратегия представлена таким же набором примеров. Для оценки вашей реализации, запустите автогрейдер:

```
python autograder.py -q q4
```

Вопрос 5 (3 балла): Эпсилон-жадные маршруты

Завершите вашего агента Q-обучения путем внедрения эпсилон-жадных действий выбора в `getAction,` согласно которым он выбирает случайные действия в участке времени эпсилон. В противном случае он будет придерживается своих лучших Q-значений. Учтите, что выбор случайного действия может повлиять на выбор наилучшего действия – поэтому вы должны выбрать не случайное неоптимальное действие, а *любое* случайное законное действие.

```
python gridworld.py -a q -k 100
```

Ваши полученные Q-значения должны напоминать значения вашего агента, особенно в хорошо известных путях. Однако, из-за случайных действий и начальной стадии обучения, ваш средний результат будет ниже чем предполагают Q-значения.

Вы можете выбрать элемент из списка случайным образом путем вызова функции `random.choice`. Вы можете смоделировать двоичную переменную с вероятностью успеха p использую `util.flipCoin(p)`, которая возвращает `True` с вероятностью `p` и `False` с вероятностью `1-p`.

Для проверки, используйте автогрейдер:

```
python autograder.py -q q5
```

Теперь вы можете запустить Q-обучение робота-контролёра без дополнительного кода:

```
python crawler.py
```

Если не сработает, значит ваш код слишком специфичен для `GridWorld` и вы должны сделать его более общим для всех МППР.

Это вызовет робота-контролёра из класса, используя вашего агента Q-обучения. Используйте различные параметры для того, чтобы посмотреть, как они влияют на действия и маршруты агента. Учтите, что задержка шага — это параметр симуляции, в то время как уровень обучения и эпсилон – параметры вашего алгоритма обучения, а фактор дисконтирования – свойство окружающей среды.

Вопрос 6 (1 балл): Пересмотр Анализа Пересечения Моста

Во-первых, обучите абсолютно случайного Q-обучающегося с темпом обучения по умолчанию на бесшумном режиме BridgeGrid для 50 эпизодов и наблюдайте, соответствует ли это оптимальному маршруту.

```
python gridworld.py -a q -k 50 -n 0 -g BridgeGrid -e 1
```

Теперь попробуйте этот эксперимент с эпсилон, равным 0. Существует ли эпсилон и темп обучения, для которых весьма вероятно (более 99%), что оптимальное правило будет изучено после 50 итераций? `question6()` в `analysis.py` будет возвращать ЛИБО двухзначный кортеж `(epsilon, learning rate)` ИЛИ строку `'NOT POSSIBLE'`, если его нет. Эпсилон управляется `-e`, темп обучения `-l`.

Примечание: Ваш ответ не должен быть зависим от механизма, используемого для выбора действия. Это означает, что ваш ответ должен быть правильным, даже если, например, мы повернули всю сетку на 90 градусов.

Чтобы оценить ваш ответ, запустите автогрейдер:

```
python autograder.py -q q6
```

Вопрос 7 (1 балл): Q-обучение и Pacman

Время сыграть в Pacman! Pacman будет функционировать в два этапа. На первом этапе, *этапе обучения*, Pacman начнет узнавать о значениях позиций и действиях. Для того чтобы установить точные Q-значения даже для самых небольших сеток требуется большое количество времени, в связи с чем обучающие игры Pacman работают по умолчанию в автоматическом режиме, без отображения в графическом интерфейсе пользователя (или консоли). После того, как обучение полностью завершено осуществляется переход в режим *тестирования*. В процессе тестирования, `self.epsilon` и `self.alpha` для Pacman будут установлены в 0.0, останавливая Q-обучение и отключая исследование, для того, чтобы позволить Pacman использовать свой полученный в результате обучения маршрут. По умолчанию тестовые игры отображаются в графическом интерфейсе пользователя. Вы можете запускать Q-обучение Pacman для самых малых сеток без какого-либо изменения кода следующим образом:

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

Отметим, что `PacmanQAgent` уже определен для вас в терминах `QLearningAgent`, описанных ранее.`PacmanQAgent` отличается только тем, что по умолчанию он имеет более эффективные параметры обучения (`epsilon=0.05, alpha=0.2, gamma=0.8`). Вы получите максимальный балл за задание, если вышеуказанная команда выполнится без отработки исключений и ваш агент будет выигрывать по меньшей мере 80% времени. Автогрейдер будет работать 100 тестовых игр после 2000 обучающих.

Подсказка: Если ваш `QLearningAgent` работает для `gridworld.py` и `crawler.py` , но обучение не осуществляется по необходимым для Pacman правилам на `smallGrid`, это может быть связано с тем, что ваши методы `getAction` и/или `computeActionFromQValues`methods в некоторых случаях недолжным образом рассматривают непредусмотренные действия. В частности, это связано с тем, что непредвиденные действия имеют по определению Q-значение равное нулю, и если все видимые действия, имеют отрицательные Q-значения, то такое непредвиденное действие может являться наиболее оптимальным. Остерегайтесь функции argmax function из util.Counter!

Примечание: Для оценки ответа, запустите:

```
python autograder.py -q q7
```

Примечание: Если вы хотите поэкспериментировать с параметрами обучения, вы можете использовать параметр `-a`, например `-a epsilon=0.1,alpha=0.3,gamma=0.7`. Эти значения затем будут доступны в качестве `self.epsilon, self.gamma` and`self.alpha` внутри агента.

Примечание: Пока будут разыгрываться в общей сложности 2010 игр, первые 2000 игр не будут отображаться, это связано с параметром `-x 2000`, который определяет первые 2000 игр обучения, как не отображаемые. Таким образом, вы будете видеть только последние 10 игр. Число обучающих игр также передается вашему агенту в качестве параметра `numTraining`.

Примечание: Если вы хотите посмотреть за 10 обучающими играми, используйте команду:

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
```

В течение обучения, вы увидите выводы каждых 100 игр со статистикой о том, что происходит с Pacman. Эпсилон положителен во время обучения, поэтому Pacman будет играть плохо, даже после того, как обучился необходимым правилам: это потому, что Pacman время от времени делает случайное поисковые движение и попадает в призрака. В качестве ориентира, эпсилон следует брать от 1,000 до 1400 игр пока вознаграждение Pacman по итогам выборки из 100 эпизодов не станет положительным, отражая, что Pacman начал выигрывать больше, чем проигрывать. К концу обучения эпсилон должна оставаться положительной и быть достаточно высокой (от 100 до 350).

Убедитесь, что вы понимаете: состояние МППР- это *точная* конфигурация уровня для Pacman с маршрутом переходов, описывающими все слои изменений этого состояния. Промежуточные игровые конфигурации, в которые перешел Pacman, а призраки не ответили этим конфигурациям, не являются состоянием МППР, но являются состояниями, объединенными в переходы.

После того, как Pacman прошел обучение, он должен стабильно выиграть в тестовых играх (по крайней мере, в 90% случаях), так как теперь он использует изученный маршрут.

Тем не менее, вы установите, что обучение одного агента на, казалось бы, простой `mediumGrid` работает плохо. В нашей реализации средние обучающие вознаграждения Pacman остаются отрицательными на протяжении всего обучения. Во время тестирования он плохо играет, вероятно, проигрывая все из своих тестовых игр. Обучение будет также занимать длительное время, несмотря на его неэффективность.

Pacman не в состоянии выиграть на больших уровнях, потому что каждая конфигурация уровня является отдельным набором условий с отдельными Q-значениями. Он не имеет возможности сделать вывод о том, что настигнуть призрака - это плохо для всех позиций. Очевидно, что такой подход не масштабируется.

Вопрос 8 (3 балла): Приближенное Q-обучение

Реализуйте программу-агента по приближенному Q-обучению, которая запоминает веса для характеристик состояний, где многие состояния могут обладать одинаковыми характеристиками. Записывайте свою реализацию класса `ApproximateQAgent` в `qlearningAgents.py`, который является подклассом `PacmanQAgent`.

Примечание: Приближенное Q-обучение допускает существование функции характеристик f(s,a) от пары состояния и действия, составляющей вектор значений характеристик $f_1(s,a)$ .. $f_i(s,a)$ .. $f_n(s,a)$. Мы представляем Вам функции характеристик `featureExtractors.py`. Векторы характеристик расположены в `util.Counter` (подобно словарю), объекты, содержащие ненулевые пары функций и значений; все исключенные характеристики имеют значение ноль.

Приближенное Q-обучение имеет следующую форму

$$Q(s,a)=\sum i=1nfi(s,a)wi$$

где каждый вес w_i связан с конкретной характеристикой $f_i(s,a)$. В своем коде Вам следует вводить весовой вектор как словарь преобразования функций (который будет возвращаться разделителем характеристик) в весовые значения. Вы будете обновлять свой весовые вектора аналогично тому, как вы обновляли Q-значения:

$$wi\leftarrow wi+\alpha\cdot difference\cdot fi(s,a)$$
$$difference=(r+\gamma maxa'Q(s',a'))-Q(s,a)$$

Отметим, что термин difference - это тоже самое, что и обычное Q-обучение, и r – это полученное опытным путем вознаграждение.

По умолчанию, `ApproximateQAgent` использует `IdentityExtractor`, который присваивает одну характеристику для каждой пары (состояние, действие). С таким разделителем характеристик ваша программа-агент по приближенному Q-обучению должна работать идентично `PacmanQAgent`. Вы можете проверить это с помощью следующей команды:

```
python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l
smallGrid
```

Важно: `ApproximateQAgent` это подкласс `QLearningAgent`, и поэтому он наследует несколько методов таких, как `getAction`. Будьте уверены, что ваши методы в `QLearningAgent` вызывают `getQValue` вместо обращения к Q-значениям напрямую, так что, когда Вы отменяете `getQValue` в приближенном агенте, новые приближенные Q-значения используются для вычисления действий.

После того, как вы убедились, что ваш агент по приближенному Q-обучению корректно работает с `IdentityExtractor`, запустите его с нашим собственным разделителем характеристик, который сможет обучить побеждать с легкостью:

```
python pacman.py -p ApproximateQAgent -a
extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid
```

Даже самые большие уровни не должны стать проблемой для вашего `ApproximateQAgent`. (*внимание: обучение может занять несколько минут*)

```
python pacman.py -p ApproximateQAgent -a
extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic
```

Если у вас нет ошибок, ваш агент по приближенному Q-обучению должен выигрывать почти каждый раз с этими простыми характеристиками, даже для 50 обучающих игр.

Система оценивания: Мы будем запускать ваш агент по приближенному Q-обучению и проверять, что он обучается тем же Q-значениям и весам характеристик, как и у нашей эталонной реализации, когда каждый из них получил одинаковый набор примеров. Чтобы поставить оценку вашей реализации, запустите автогрейдер:

```
python autograder.py -q q8
```

Поздравляем! Вы обучили агента Pacman!