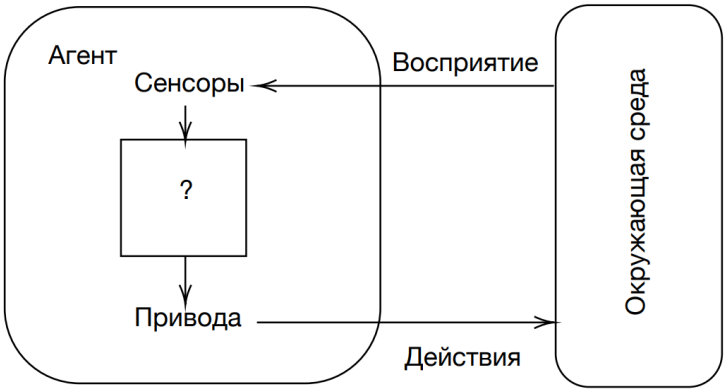
Агент — это все, что можно рассматривать как воспринимающее окружающую среду с помощью датчиков и воздействующее на нее с помощью исполнительных механизмов.



Роботизированный агент

- камеры и инфракрасные дальномеры и другие датчики
- различные исполнительные механизмы, например двигатели

Программный агент

- нажатия клавиш, содержимое файлов, полученные сетевые пакеты
- отображение на экране, файлы, отправленные сетевые пакеты в качестве исполнительных устройств

Человек, как агент

- глаза, уши и другие органы
- руки, ноги, рот и другие части тела

Агенты и среда

- Последовательность восприятия: вся история о том что агент получил с датчиков
- Выбор действия агента в любой момент времени может зависеть от всей последовательности восприятия, наблюдаемой на данный момент
- Поведение агента характеризуется агентной функцией, что отображает пространство наблюдений в пространство действий
- Можно представить себе таблицу, которая будет описывать все действия отдельно взятого агента
- Внутренняя функция агента будет реализована агентской программой, которая запускается на физической архитектуре для производства действия
- агент = архитектура + программа

Робот-пылесос

Последовательность восприятия	Действия
[А, чисто]	Вправо
[А, грязно]	Чистить
[Б, чисто]	Влево
[Б, грязно]	Чистить
[А, чисто], [А, чисто]	Вправо
[А, чисто], [А, грязно]	Чистить
. . .	. . .
[А, чисто], [А, чисто], [А, чисто]	Вправо
[А, чисто], [А, чисто], [А, грязно]	Чистить
. . .	. . .

Рациональный агент

- Агент должен стремиться делать «правильные вещи», основываясь на том, что он может воспринимать и какие действия он может выполнять
- Правильное действие — это то, которое приведет агента к наибольшему успеху
- Мера эффективности: Объективный критерий успешности поведения агента
- Например, мерой эффективности агента-пылесоса может быть количество вычищенной грязи, затраченное время, количество потребленной электроэнергии, количество производимого шума и т.д.
- Как правило, лучше разрабатывать показатели эффективности в соответствии с тем, что человек на самом деле хочет видеть в окружающей среде. А не в соответствии с тем, как, по мнению пользователя, должен вести себя агент (количество убранной грязи по сравнению с чистым полом)
- Более подходящим показателем будет вознаграждение агента за чистый пол

Рациональность

То, что является рациональным в любой момент времени, зависит от четырех вещей:

- Мера эффективности, определяющая критерий успеха
- Предварительные знания агента об окружающей среде
- Действия, которые агент может выполнить
- Последовательность восприятия агента на текущий момент времени

Рациональный агент: Для каждой возможной последовательности восприятия рациональный агент должен выбрать действие, которое, как ожидается, максимизирует его показатель эффективности, учитывая данные, предоставленные последовательностью восприятия, и любые встроенные знания, которыми обладает агент.

Агент робот-пылесос 1

Предположим следующее:

- Мера производительности присуждает один балл за каждый чистый квадрат на каждом временном шаге в течение 1000 временных шагов
- География окружающей среды известна, распределение грязи и начальное местоположение агента неизвестны. Чистые квадраты остаются чистыми, иначе пылесос может отчистить текущий квадрат. Действия Влево и Вправо перемещают агента влево и вправо, за исключением случаев, когда это выводит агента за пределы среды, в этом случае агент остается на месте.
- Единственные доступные действия: «влево», «вправо», «чистить» и «бездействовать»
- Агент правильно воспринимает свое местоположение и то, содержит ли это место грязь

В этих обстоятельствах агент рационален: его ожидаемая производительность по крайней мере столь же высока, как и у любого другого агента.

Агент робот-пылесос 2

Один и тот же агент будет иррационален в разных обстоятельствах

- После того, как грязь будет очищена, агент будет безостановочно двигаться назад и вперед
- Если оценка эффективности включает штраф в один пункт за каждое движение влево или вправо, то агент будет работать плохо
- Более эффективным вариантом является обеспечение чистоты всех квадратов
- Если чистые квадраты могут снова стать грязными, агенты должны время от времени проверять и очищать их, если это необходимо
- Если география окружающей среды неизвестна, агентам необходимо изучить ее, а не придерживаться квадратов A и B

Рациональные агенты

- Рациональность максимизирует ожидаемую производительность, в то время как в идеале нужно максимизировать фактическую производительность
- Агенты могут выполнять действия для изменения будущего восприятия с целью получения полезной информации (сбор информации, разведка)
- Агент автономен, если его поведение определяется его собственным опытом (с возможностью обучения и адаптации)

Характеристики рабочей среды

PEAS:

- Performance measure,
- Environment,
- Actuators,
- Sensors

При разработке агента первым шагом всегда необходимо как можно полнее охарактеризовать рабочую среду (PEAS).

PEAS автоматизированного такси

- P: Безопасная, быстрая, законная, комфортная поездка, максимальная прибыль
- E: Дороги, другой транспорт, пешеходы, клиенты
- A: Рулевое колесо, акселератор, тормоз, сигнал, гудок
- S: Камеры, сонар, спидометр, GPS, одометр, датчики двигателя, клавиатура

PEAS системы медицинской диагностики

- P: Здоровый пациент, минимизация затрат, судебные иски
- E: Пациент, больница, персонал
- A: Экранный дисплей (вопросы, тесты, диагнозы, лечение, направления)
- S: Клавиатура (ввод симптомов, результатов обследования, ответы пациента)

PEAS системы анализа спутниковых изображений

- P: правильная категоризация изображения
- E: нисходящая связь с орбитального спутника
- A: отображение категоризации сцены на дисплее
- S: массивы цветных пикселей

PEAS робота для подбора деталей

- P: Процент деталей в правильных контейнерах
- E: Конвейерная лента с деталями, контейнеры
- A: Шарнирные рука и кисть
- S: Камера, датчики угла наклона звеньев

Типы окружающей среды 1

- Полностью наблюдаемые против частично наблюдаемых
- Детерминированные против стохастических
- Эпизодическая против последовательной
- Статические против динамических

- Дискретный против непрерывного
- Одноагентный против многоагентного

Полностью наблюдаемые против частично наблюдаемых:

- Среда является полностью наблюдаемой, если датчики агента дают ему доступ к полному состоянию среды в каждый момент времени
- Полностью наблюдаемые среды удобны, поскольку агенту не нужно поддерживать внутреннее состояние, чтобы следить за миром
- Среда может быть частично наблюдаемой из-за шумных и неточных датчиков или из-за того, что часть состояния просто отсутствует в данных датчиков
- Примеры: пылесос с локальным датчиком загрязнения, водитель такси

Детерминированные против стохастических:

- Среда является детерминированной, если следующее состояние среды полностью определяется текущим состоянием и действием, выполняемым агентом
- В принципе, агенту не нужно беспокоиться о неопределенности в полностью наблюдаемой, детерминированной среде
- Если среда частично наблюдаема, то она может казаться стохастической
- Примеры: Мир робота-пылесоса детерминирован, а водитель такси – нет
- Если среда детерминирована, за исключением действий других агентов, то среда является стратегической

Эпизодическая среда против последовательной:

- В эпизодических средах опыт агента делится на атомарные «эпизоды» (каждый эпизод состоит из восприятия агентом и последующего выполнения одного действия), и выбор действия в каждом эпизоде зависит только от самого эпизода
- Примеры: задачи классификации
- В последовательных средах текущее решение может повлиять на все будущие решения.
- Примеры: шахматы и таксист

Статика против динамики:

- Окружение остается неизменным, пока агент размышляет
- Со статическими средами легко работать, потому что агенту не нужно постоянно смотреть на мир, пока он принимает решение о действии, и не нужно беспокоиться о течении времени
- Динамические среды постоянно спрашивают агента, что он хочет сделать
- Среда является полудинамической, если сама среда не меняется с течением времени, но меняется оценка работы агента
- Примеры: вождение такси является динамичным, шахматы при игре с часами являются полудинамичными, кроссворды – статичными

Дискретность в сравнении с непрерывностью:

- Ограниченное количество отдельных, четко определенных состояний, восприятий и действий
- Примеры: Шахматы имеют конечное число дискретных состояний и дискретный набор восприятий и действий. Вождение такси имеет непрерывные состояния и действия

Одиночный агент против мультиагента:

- Агент, действующий сам по себе в среде, является одиночным агентом
- Примеры: Кроссворд - одноагентный, а шахматы - двухагентные

- Вопрос: Должен ли агент A относиться к объекту B как к агенту или его можно рассматривать как стохастически ведущий себя объект?
- Можно ли поведение B лучше всего описать как максимизацию меры эффективности, значение которой зависит от поведения агента A.
- Примеры: шахматы - это конкурентная мультиагентная среда, а вождение такси - частично кооперативная мультиагентная среда.

Агентские функции и программы

- Агент полностью определен агентской функцией, отображающая множество последовательности восприятия на множество действий
- Одна функция агента (или небольшой класс эквивалентности) рациональна
- Цель: найти способ реализации рационального агентской функции в сжатом виде -> разработать программу агента Агент = агентская программа + архитектура
- Архитектура: некое вычислительное устройство с физическими датчиками и исполнительными механизмами (ПК, роботизированный автомобиль), должно быть соответствие: для действия ходьбы нужны ноги

Программа агента:

- Принимает текущее восприятие в качестве входных данных от датчиков
- Возвращает действие исполнительным механизмам
- В то время как функция агента принимает всю историю восприятия, программа агента принимает на вход только текущее восприятие, которое является единственным доступным входом из окружающей среды
- Агенту необходимо запомнить всю последовательность восприятий, если она ему нужна

Агент на основе табличного поиска

- Тривиальная программа агента: отслеживает последовательность восприятия и затем использует ее для индексации в таблице действий, чтобы решить, что делать
- Разработчики должны построить таблицу, которая содержит соответствующее действие для каждой возможной последовательности восприятия
- Недостатки:
    - Большая таблица
    - Место для хранения таблицы
    - Долгое время для построения таблицы
    - Нет автономности
    - Даже при обучении требуется много времени для изучения записей в таблице

Типы агентов

Как из небольшого объема кода можно получить рациональное поведение, а не таблицу

Четыре основных типа в порядке возрастания общности:

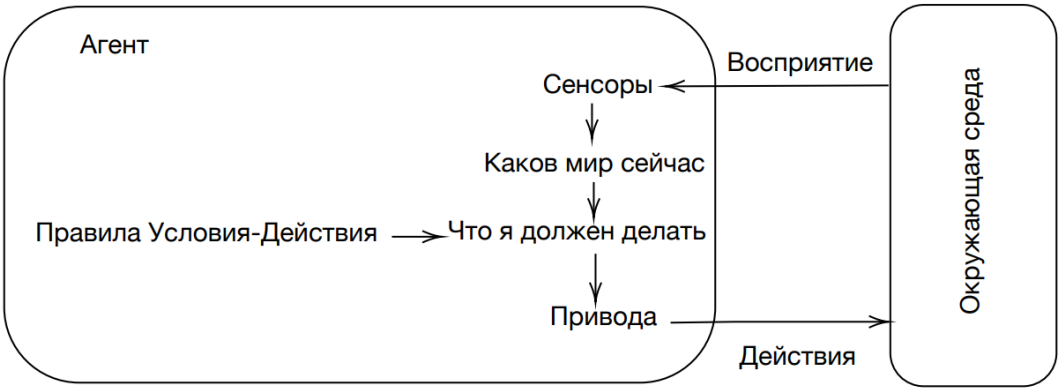
- Простые рефлекторные агенты
- Модельные рефлекторные агенты
- Агенты, основанные на целях
- Агенты на основе полезности

Простой рефлекторный агент

- Выбор действий на основе текущего восприятия, игнорируя остальную последовательность восприятия
- Пример: простой рефлекторный агент пылесоса

Algorithm 5.1: REFLEX-VACUUM-AGENT($location, status$)

```
if status = Dirty
  then return (Suck)
  else if location ← A
  then return (Right)
  else if location ← B
  then return (Left)
```



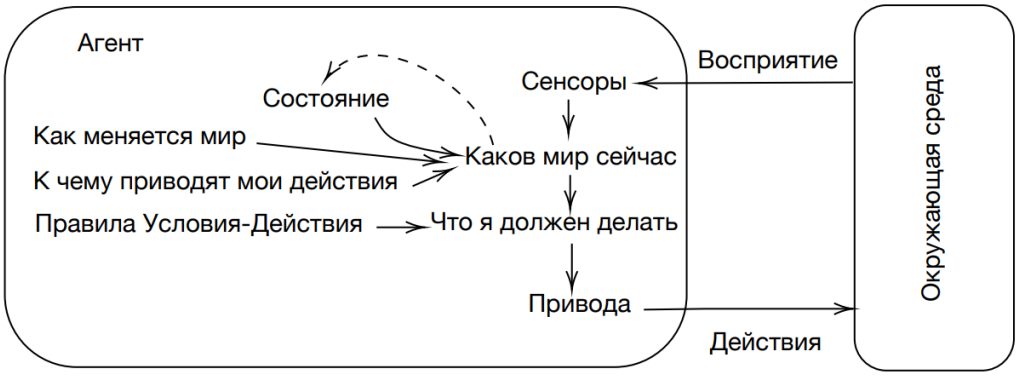
Algorithm 5.2: SIMPLE-REFLEX-AGENT($percept$)

```
global rules
state ← INTERPRET_INPUT(percept)
rule ← RULE_MATCH(state, rules)
action ← RULE_ACTION(rule)
return (action)
```

- Простые рефлекторные агенты оказываются очень ограниченного интеллекта
- Агент будет работать, только если правильное решение может быть принято на основе текущего восприятия, то есть только если окружающая среда полностью наблюдаема
- Бесконечные циклы часто неизбежны — избежать их можно путем рандомизации

Модульные рефлекторные агенты

- Агент должен отслеживать ту часть мира, которую он не может видеть сейчас
- Агент должен поддерживать некое внутреннее состояние, которое зависит от истории восприятия и отражает, по крайней мере, некоторые ненаблюдаемые аспекты текущего состояния.
- Обновление информации о внутреннем состоянии с течением времени требует двух видов знаний, которые должны быть закодированы в программе агента
    - Информация о том, как мир развивается независимо от агента
    - Информация о том, как собственные действия агента влияют на мир
- Модель мира - Модельные рефлекторные агенты

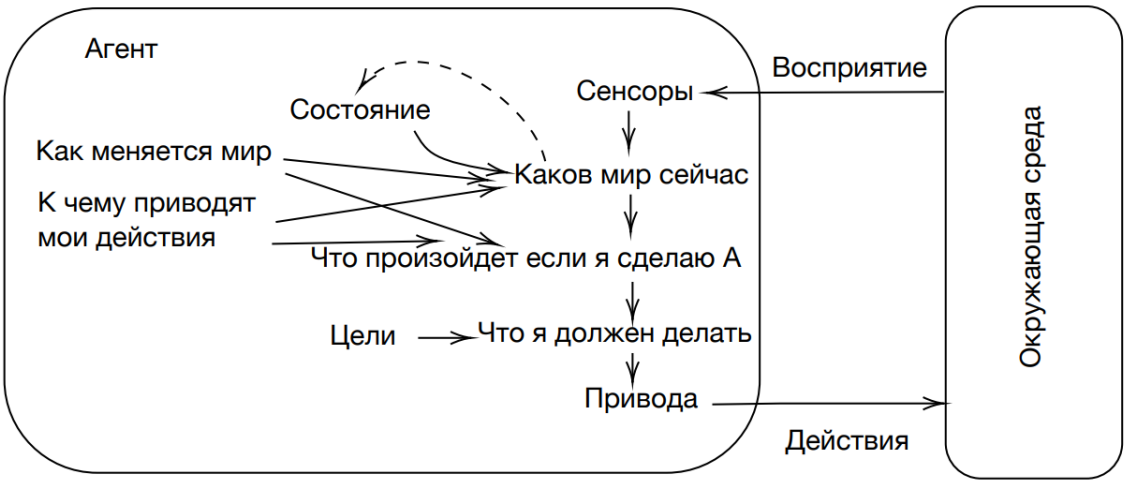


Algorithm 5.3: REFLEX-AGENT-WITH-STATE($percept$)

global $state, rules, action$
$state \leftarrow$ UPDATE_INPUT$(state, action, percept)$
$rule \leftarrow$ RULE_MATCH$(state, rules)$
$action \leftarrow$ RULE_ACTION$(rule)$
return $(action)$

Модели основанные на цели

- Знание текущего состояния среды не всегда достаточно, чтобы решить, что делать (например, принятие решения на дорожной развязке)
- Агенту необходима некая информация о цели, которая описывает ситуации, которые являются желательными
- Программа агента может объединить ее с информацией о результатах возможных действий, чтобы выбрать действия, которые достигнут цели
- Обычно это требует поиска и планирования



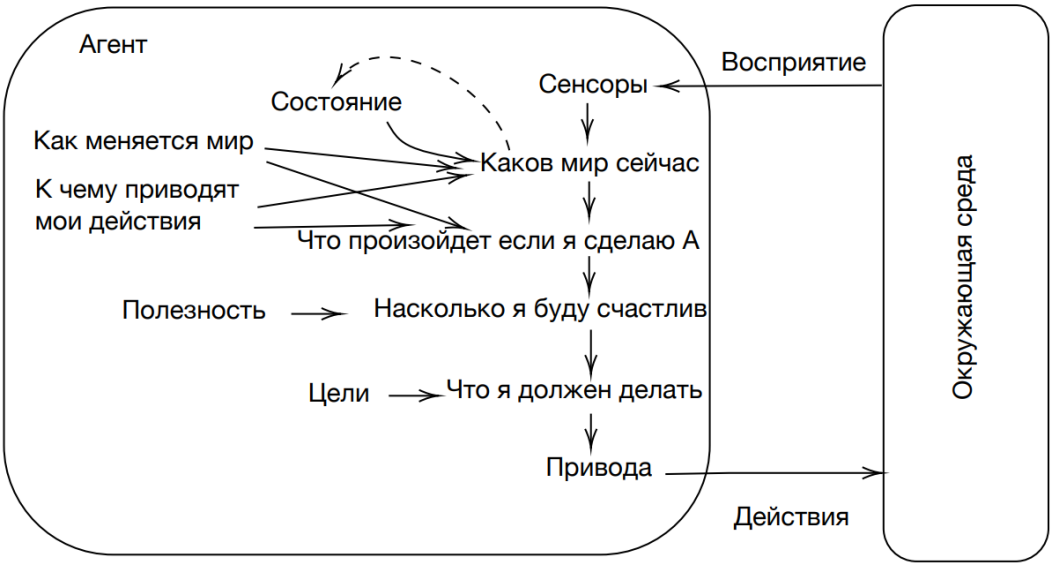
GA vs. MBRM

- Хотя агенты, основанные на целях, кажутся менее эффективными, они более гибкие, потому что знания, которые поддерживают их решение, представлены в явном виде и могут быть изменены
- С другой стороны, для рефлекторного агента нам пришлось бы переписать множество правил ”условие-действие”

- Поведение агента, основанного на цели, может быть легко изменено
- Правила рефлекторного агента должны быть изменены для новой ситуации

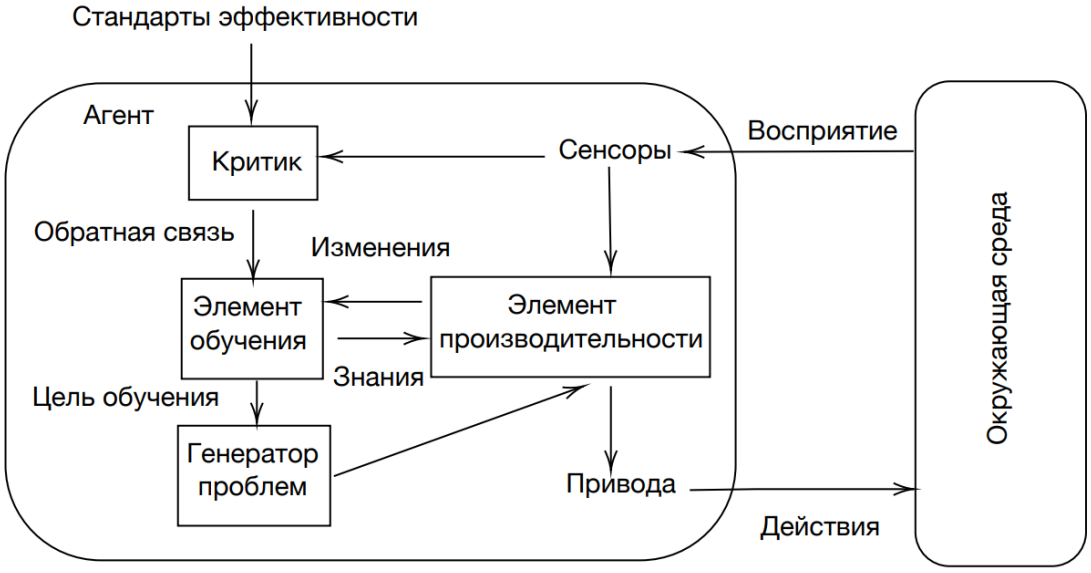
Агенты на основе полезности

- Одних целей недостаточно для создания высококачественного поведения в большинстве сред — они просто обеспечивают бинарное различие между счастливыми и несчастливыми состояниями
- Более общая мера эффективности должна позволять сравнивать различные состояния мира в соответствии с тем, насколько счастливыми они сделают агента, если они могут быть достигнуты
- Счастье - Полезность (качество быть полезным)
- Функция полезности отображает состояние на вещественное число, которое описывает связанную с ним степень счастья



Обучающиеся агенты

- Тьюринг - вместо того, чтобы программировать интеллектуальные машины вручную, что слишком трудоемко, создайте обучающиеся машины, а затем обучайте их
- Обучение также позволяет агенту действовать в изначально неизвестных условиях и становиться более компетентным, чем это возможно благодаря его первоначальным знаниям.



- Элемент обучения - отвечает за внесение улучшений
- Элемент производительности – отвечает за выбор внешних действий (это то, что мы раньше определяли как весь агент).
- Обучающий элемент использует обратную связь от критика о том, как агент работает, и определяет, как элемент производительности должен быть изменен, чтобы в будущем работать лучше
- Генератор проблем отвечает за предложение действий, которые приведут к новому и информативному опыту

Представление агентов и среды

Представление агентов

Агент наблюдает за миром и выполняет действия в среде, у него также есть внутреннее состояние, которое он обновляет.

```
class Agent(object):
    def __init__(self,env):
        """Инстанцирование агента"""
        self.env=env

    def go(self,n):
        """Агент действует в течении n временных шагов"""
        raise NotImplementedError("go") # абстрактный метод
```

Класс трассировки

Реализуем логирование:

```
class Displayable(object):
    """Класс трасировки кода
    Глубину трассировки контроллирует max_display_level
    """
    max_display_level = 1    # может быть переписано в подклассе

    def display(self,level,*args,**nargs):
        """Выводит агрументы, если уровен меньше или равен текущему
    max_display_level.
        Уровень – целочисленное.
        остальные аргументы – любые аргументы, которые может принимать
    print.
        """
        if level <= self.max_display_level:
            print(*args, **nargs)  # Только в Python3
```

Представление среды

Среда принимает действия агентов, обновляет свое внутреннее состояние и возвращает восприятия.

```
class Environment(Displayable):
    def initial_percepts(self):
        """возвращает начальные восприятия для агента"""
        raise NotImplementedError("initial_percepts") # абстрактный метод

    def do(self,action):
        """выполняет действие в окружающей среде
        возвращает следующее восприятие"""
        raise NotImplementedError("do") # абстрактный метод
```

Закупка бумаги

```
class TP_env(Environment):
    prices = [234, 234, 234, 234, 255, 255, 275, 275, 211, 211, 211,
    234, 234, 234, 234, 199, 199, 275, 275, 234, 234, 234, 234, 255,
    255, 260, 260, 265, 265, 265, 265, 270, 270, 255, 255, 260, 260,
    265, 265, 150, 150, 265, 265, 270, 270, 255, 255, 260, 260, 265,
    265, 265, 265, 270, 270, 211, 211, 255, 255, 260, 260, 265, 265,
    260, 265, 270, 270, 205, 255, 255, 260, 260, 265, 265, 265, 265,
    270, 270]
    max_price_addon = 20  # максимум случайной величины, добавляемой для получения цены

    def __init__(self):
        """Агент закупки бумаги"""
        self.time=0
        self.stock=20
        self.stock_history = []  # память об истории запасов
        self.price_history = []  # память об истории цен

    def initial_percepts(self):
        """Начальное восприятие"""
        self.stock_history.append(self.stock)
        price = self.prices[0]+random.randrange(self.max_price_addon)
        self.price_history.append(price)
        return {'price': price,
                'instock': self.stock}

    def do(self, action):
        """выполняет действие купить() и возвращает восприятие цена( и наличие на складе)"""
        used = pick_from_dist({6:0.1, 5:0.1, 4:0.2, 3:0.3, 2:0.2, 1:0.1})
        bought = action['buy']
        self.stock = self.stock+bought-used
        self.stock_history.append(self.stock)
        self.time += 1
        price = (self.prices[self.time%len(self.prices)] # повторяющийся шаблон
                 +random.randrange(self.max_price_addon) # плюс случайность
                 +self.time//2)                          # плюс инфляция
        self.price_history.append(price)
        return {'price': price,
                'instock': self.stock}
```

```
def pick_from_dist(item_prob_dist):
    """ возвращает значение из распределения.
    item_prob_dist – это словарь item:probability, где
        вероятности равны 1.
    возвращает элемент, выбранный пропорционально его вероятности
    """
    ranreal = random.random()
    for (it,prob) in item_prob_dist.items():
        if ranreal < prob:
            return it
        else:
            ranreal -= prob
    raise RuntimeError(str(item_prob_dist)+" не является распределением
    вероятности")
```

Агент не имеет доступа к ценовой модели, а может наблюдать только текущую цену и количество товара на складе. Он должен решить, сколько купить.

Состояние убеждений агента - это оценка средней цены бумаги и общее количество денег, которое агент потратил.

```
class TP_agent(Agent):
    def __init__(self, env):
        self.env = env
        self.spent = 0
        percepts = env.initial_percepts()
        self.ave = self.last_price = percepts['price']
        self.instock = percepts['instock']

    def go(self, n):
        """Агент действует в течении n временных шагов"""
        for i in range(n):
            if self.last_price < 0.9*self.ave and self.instock < 60:
                tobuy = 48
            elif self.instock < 12:
                tobuy = 12
            else:
                tobuy = 0
            self.spent += tobuy*self.last_price
            percepts = env.do({'buy': tobuy})
            self.last_price = percepts['price']
            self.ave = self.ave+(self.last_price-self.ave)*0.05
            self.instock = percepts['instock']
```

```
env = TP_env()
ag = TP_agent(env)
ag.go(90)
ag.spent/env.time    ## средние затраты за период времени
```

```
import matplotlib.pyplot as plt

class Plot_prices(object):
    """Настраиваем график для истории цен и количества на складе"""
    def __init__(self, ag,env):
        self.ag = ag
        self.env = env
        plt.ion()
        plt.xlabel("Время")
        plt.ylabel("Количество на складе.                                                    Цена.")

    def plot_run(self):
        """график истории цен и складских запасов"""
        num = len(env.stock_history)
        plt.plot(range(num),env.stock_history,label="In stock")
        plt.plot(range(num),env.price_history,label="Price")
        #plt.legend(loc="upper left")
        plt.draw()

pl = Plot_prices(ag,env)
ag.go(90); pl.plot_run()
```