Севастопольский государственный университет
Институт информационных технологий

“МЕТОДЫ И СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА" (МиСИИ)

Бондарев Владимир Николаевич

Лекция 6

ПОИСК РЕШЕНИЙ
В ИГРОВЫХ ПРОГРАММАХ
(поиск в условиях противодействия)

В.Бондарев

Основные понятия

Рассмотрим детерминированные **игры с полной информацией** о текущей игровой ситуации, где два игрока-противника по очереди делают ходы. Успех одного игрока – такая же по величине потеря для другого игрока (**игры с нулевыми суммами**).

Допустимые ходы игры определяются **конечным набором правил**, которые исключают случайность. Игра начинается из специфического начального положения и завершается **победой того или иного игрока, либо ничьей.**

Сложность поиска решений в игровых программах **весьма высока**. Например, общее количество вершин дерева игры в шахматы оценивается значением 10^{120}. Если предположить, что можно обследовать 10^{9} вершин в секунду, то построение полного дерева игры в шахматы заняло бы 10^{103} лет . Из этого следует, **что полный просмотр дерева игры невозможен.**

Основные понятия

Поэтому при поиске с использованием игровых деревьев необходимо выполнять следующее:

1) прогнозировать граничную глубину поиска;
2) оценивать перспективность позиций игры с помощью эвристических функций.

Для этого в каждой позиции игры формируется **дерево возможных продолжений игры**, имеющее определенную глубину, и с помощью эвристической оценочной функции вычисляются **оценки концевых вершин такого дерева.** Затем полученные **оценки распространяются вверх по дереву**, и корневая вершина, соответствующая текущей позиции, получает оценку, позволяющую выяснить силу игрока в данной позиции.

Минимаксный метод

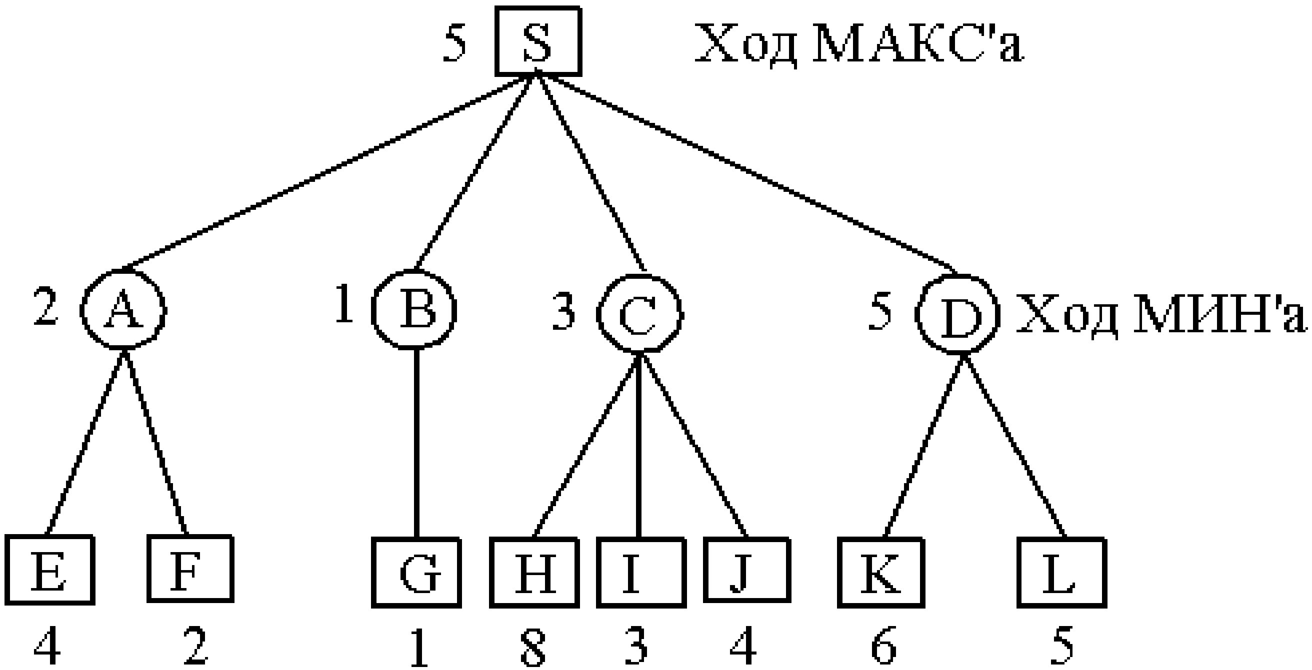
Здесь вместо полного просмотра дерева игры обследуется лишь его небольшая часть. В этом случае говорят, что дерево подвергается ***подрезке.*** Простейший способ подрезки — это просмотр дерева игры на определенную глубину. Это означает, что ***процесс обследования заканчивается в вершинах, которые не являются терминальными.*** Поэтому ***дерево поиска*** – это только верхняя часть дерева игры.

Для оценки силы игрока в различных позициях поискового дерева применяются оценочные функции. Различают два вида оценок: **статические и динамические**.

Статические оценки приписываются концевым вершинам дерева поиска и вычисляются с помощью функции оценки состояния *eval*(*s*), в основу построения которой кладутся некоторые эвристические правила.

Минимаксный метод

Динамические оценки получаются при распространении статических оценок вверх по дереву. Метод, которым это достигается, называется ***минимаксным.***



Для вершины, в которой ход выполняет игрок (МАКС), выбирается наибольшая из оценок вершин нижнего уровня Для вершины, в которой ход выполняет противник (МИН), выбирается наименьшая из оценок дочерних вершин.

В.Бондарев

Минимаксный метод

Если обозначить **статическую** оценку в вершине s через $eval(s)$, а **динамическую** – через $V(s)$, то формально оценки, приписываемые вершинам в соответствии с минимаксным принципом, можно записать так :

$$V(s)=eval(s),$$

если s – концевая вершина дерева поиска;

$$V(s)=\max_i V(s_i),$$

если s – вершина с ходом МАКС'а;

$$V(s)=\min_i V(s_i),$$

если s – вершина с ходом МИН'а. Здесь s_i дочерние вершины для вершины s.

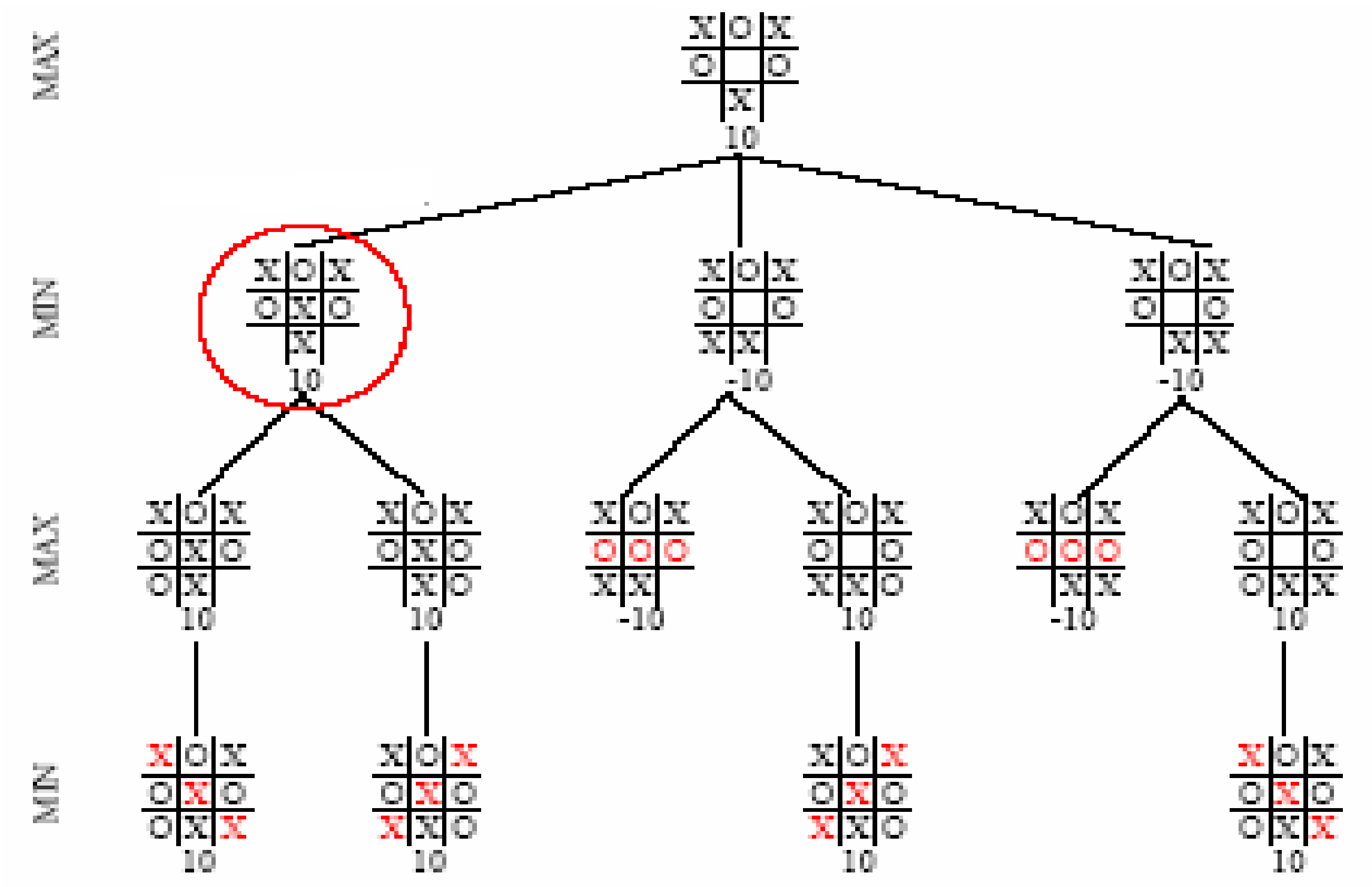
Для приближения минимаксной стратегии игры к гарантирующей приходится увеличивать глубину поиска.

Минимаксный метод
(псевдокод)

```
def value(state):
        if state соответсв. концевой вершине: return eval(state)
        if  агент MAX: return max_value(state)
        if  агент MIN: return min_value(state)

def max_value(state):
        v = -∞
        for succ in successor(state):  # для всех дочерних состояний
                v=max(v, value(succ)) # рекурсивный вызов value(state)
        return v

def min_value(state):
        v = +∞
        for succ in successor(state):
                v=min(v, value(succ)) # рекурсивный вызов value(state)
        return v
```

Минимаксный поиск за счет рекурсивного погружения ведет себя подобно поиску в глубину, вычисляя оценки состояний в том же порядке, что и DFS. Поэтому временная сложность алгоритма $O(b^m)$, где m — максимальная глубина дерева поиска.

Минимаксный метод
(уровень терминальных вершин, функция полезности)



Стат. оценки в **терминальных** вершинах (полезность, utility(s)): выигрыш +10; проигрыш -10; ничья 0.

Функция оценки

Полезность нетерминальных состояний определяют с помощью тщательно выбранной **функции оценки** (evaluation function), которая дает приближенное значение полезности этих состояний.

Чаще всего функция оценки *eval(s)* представляет собой линейную комбинацию функций $f_i(s)$:

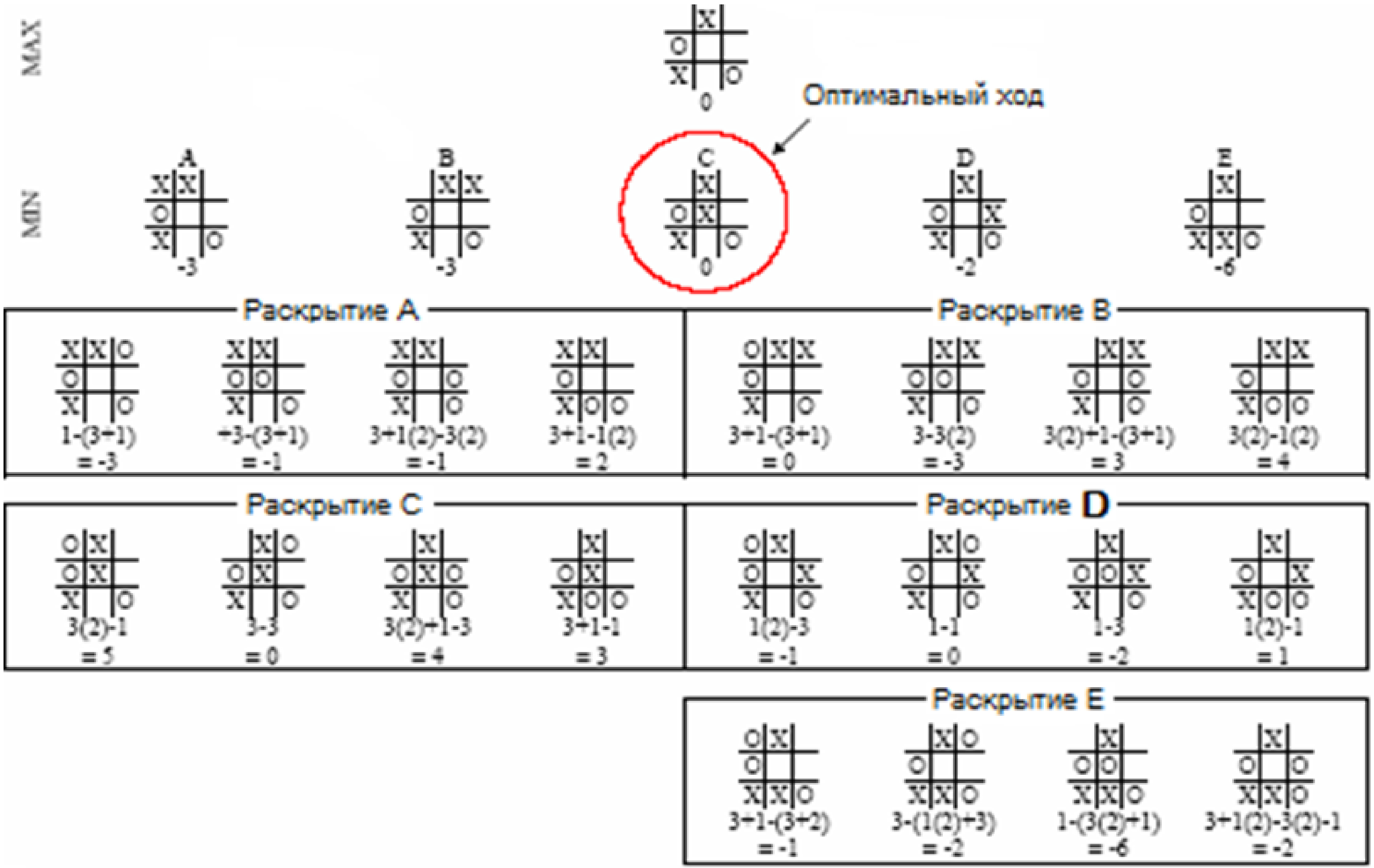
$$eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s),$$

где каждая функция $f_i(s)$ вычисляет некоторую **характеристику состояния** s, и каждой характеристике назначается соответствующий вес w_i.

Например, в шашках можно построить оценочную функцию с 4-мя характеристиками: количество пешек у агента, количество королев у агента, количество пешек у противника и количество королев у противника.

Структура оценочной функции может быть совершенно произвольной, и она не обязательно должна быть линейной

Минимаксный метод
(уровень нетерминальных вершин, функция оценки)

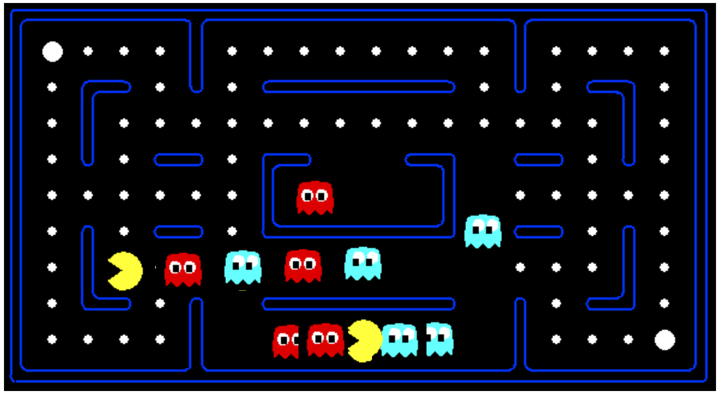


$$\mathrm{eval}(s) = 10X_3(s) + 3X_2(s) + X_1(s) - (10O_3(s) + 3O_2(s) + O_1(s))$$

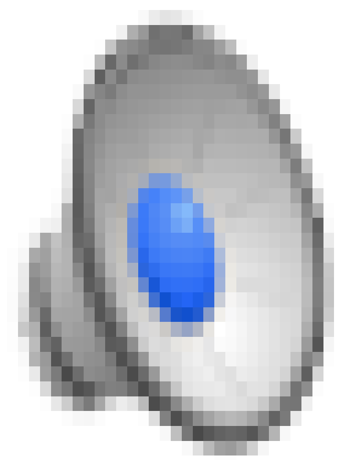
Xn(s)- кол-во строк, столбцов или диагоналей, содержащих **n** **X**-ов и несодержащих **O**

On(s)- кол-во строк, столбцов или диагоналей, содержащих **n** нулей и несодержащих крестиков

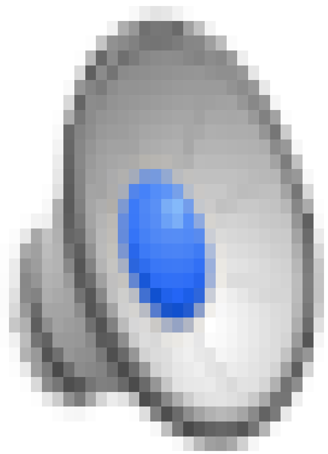
Функция оценки в Pacman



Видео: метания Пакмана (d=2)



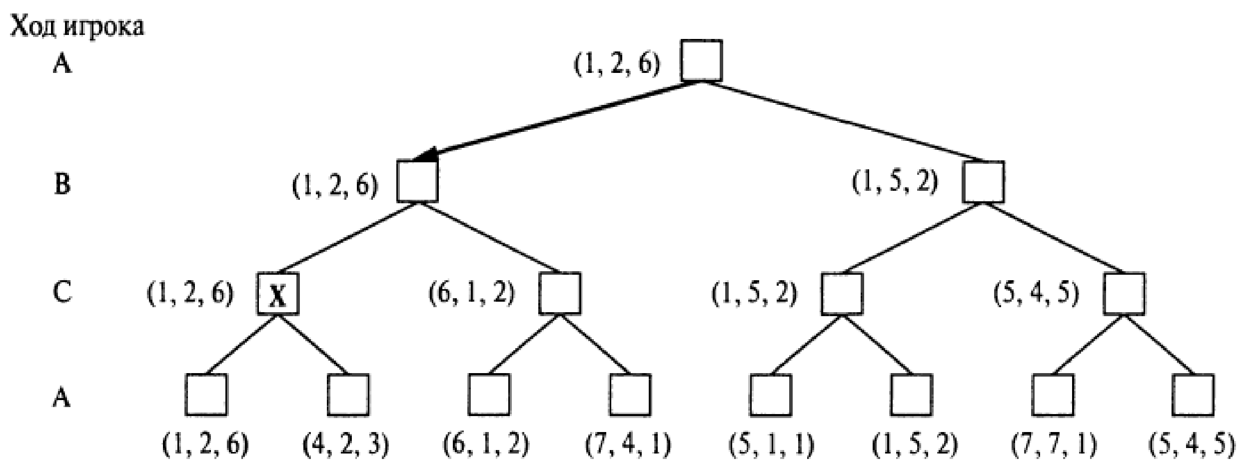
Видео: метания Пакмана (d=2) - решение



Мультиагентный поиск

Распространим идею минимаксного поиска на игры с несколькими игроками (агентами)

Заменим единственное значение оценки полезности для каждого узла вектором значений оценок . Например, в игре с тремя игроками A, B и C с каждым узлом ассоциируется вектор (V_A , V_B , V_C) с тремя оценками. Для концевого состояния этот вектор задает полезность данного состояния с точки зрения каждого игрока.



Поскольку для игрока C в состоянии X ход с V_C = 6 предпочтительнее, то игрок C в состоянии X выбирает первый ход и состоянию X приписывается вектор оценок (1, 2, 6).

Мультиагентный поиск для Pacman

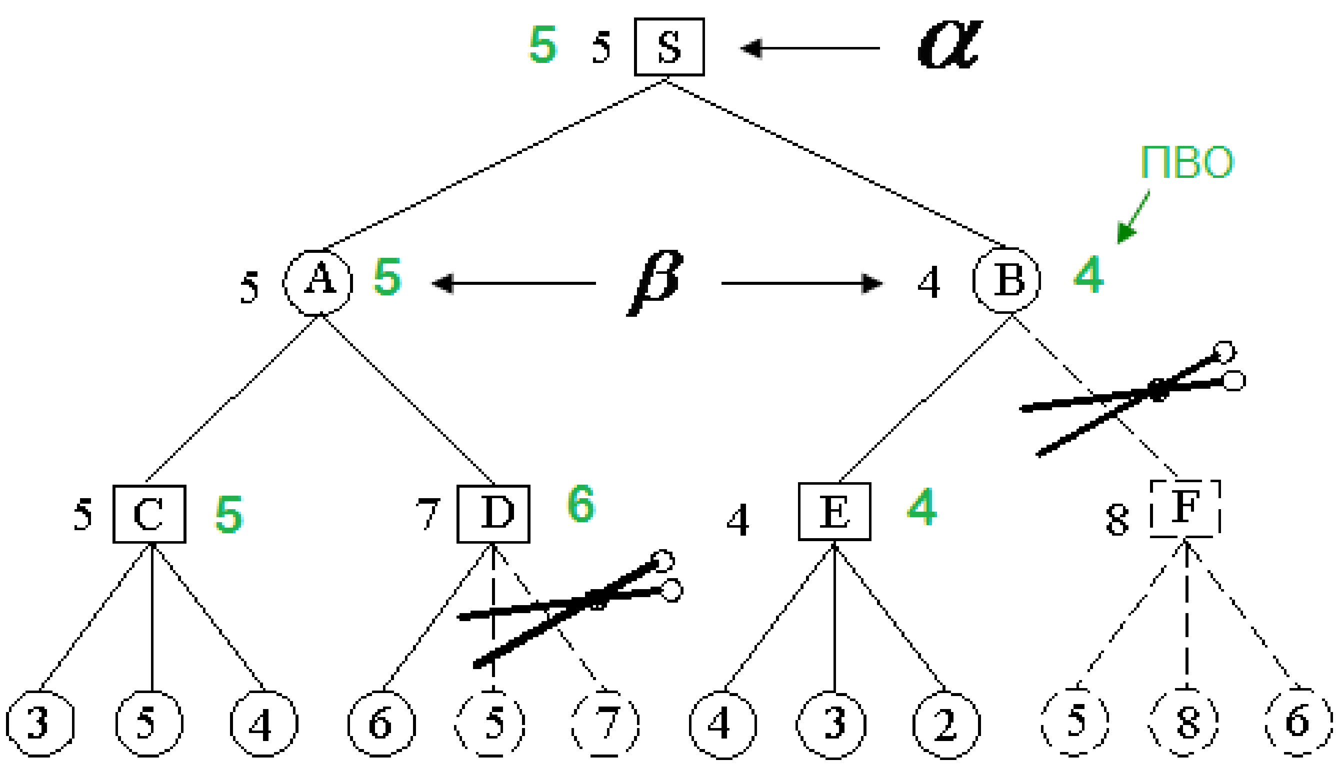
Чтобы организовать сценарий мультиагентной игры в Пакман необходимо расширить пседокод функции **min_value(state)** на случай нескольких агентов-призраков. Для этого функция **min_value** должна обеспечивать реализацию поиска в глубину путем косвенных рекурсивных вызовов функции **value** с дополнительными входными параметрами, задающими индекс агента **agentIndex** и уровень глубины **depth.** При очередном косвенном рекурсивном вызове **min_value** на одном и том же уровне **depth** дерева поиска параметр **agentIndex** должен увеличиваться на 1 (для передачи хода следующему агенту-призраку). Например:

```
def min_value(state, depth, agentIndex):
        …
    v = +∞
    # Для всех допустимых действий агента с номером agentIndex
    for action in множество_допустимых_действий(state, agentIndex):
        if agentIndex == номер_последнего_агента:
            v = min(v, value(successor(state,agentIndex, action),depth + 1, 0))
         else:
             v = min(v, value(successor(state, agentIndex, action), depth, agentIndex+1))
      return v
```

Когда все агенты-призраки выполнят свои действия, ход передается Пакману (индекс агента 0) и глубина поиска увеличивается на 1.

Альфа-бета поиск

Минимаксный метод поиска предполагает полное обследование дерева поиска. Чтобы получить оценку корневой вершины, необязательно выполнять систематический обход дерева поиска. Рассмотрим пример



Альфа-бета поиск

В альфа-бета методе статическая оценка каждой концевой вершины вычисляется сразу, как только такая вершина будет построена. Затем полученная оценка распространяется вверх по дереву и с каждой из родительских вершин связывается **предположительно возвращаемая оценка** (**ПВО**).

При этом используются следующие правила:

1) если ПВО для бета-вершины становится меньше или равной ПВО родительской вершины, вычисленной на предыдущем шаге, то нет необходимости строить дальше поддерево, начинающееся ниже этой бета-вершины ***(альфа-отсечение);***

2) если ПВО для альфа-вершины становится больше или равной ПВО родительской вершины, вычисленной на предыдущем шаге, то нет необходимости строить дальше поддерево, начинающееся ниже этой альфа-вершины ***(бета-отсечение).***

Альфа-бета поиск

Для дерева поиска, изображенного на рисунке, ситуация **альфа-отсечения** имеет место при вычислении ПВО вершины *B* ([$H(B)$=4] < [$H(S)$=5]), а ситуация **бета-отсечения** – для вершины *D* ([$H(D)$=6] > $H(A)$=5]).

Альфа-бета поиск позволяет **увеличить глубину** дерева поиска примерно в два раза по сравнению минимаксным алгоритмом, что приводит к более сильной игре.

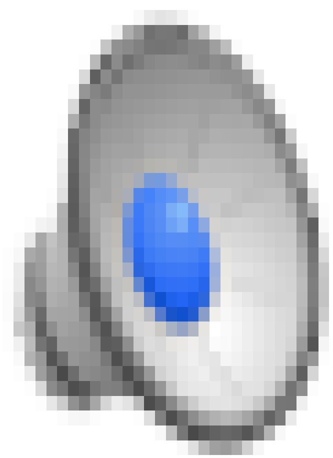
Альфа-бета поиск

Псевдокод альфа-бета поиска аналогичен минимаксному поиску, но требует переопределения функций, вычисляющих минимальные и максимальные оценки:

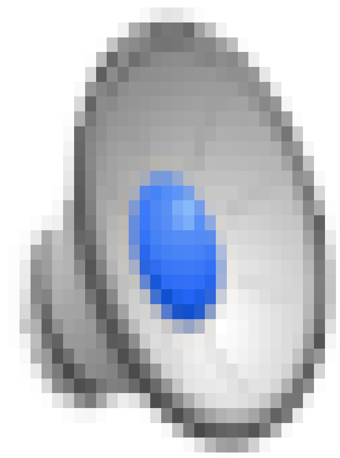
```
def max_value(state, α, β):
        v = -∞
        for succ in successor(state):
                v=max(v, value(succ, α, β))
                if v >= β: return v          # альфа отсечение
                α=max(α, v)
        return v

def min_value(state, α, β):
        v = +∞
        for succ in successor(state):
                v=min(v, value(succ, α, β))
                if v<= α: return v           # бета отсечение
                β =min(β, v)
        return v
```

Оценка временной сложности алгоритма альфа-бета поиска соответствует $O(b^{m/2})$.

Видео Pacman (d=2)



Видео Pacman (d=10)



Expectimax

Минимаксный поиск бывает чрезмерно пессимистичным, т.к. оппонент выбирает с точки зрения игрока наихудший ход. В ситуациях, когда в действиях оппонента присутствует неопределенность, более оптимистичный вариант действий игрока обеспечивает метод, известный как Expectimax.

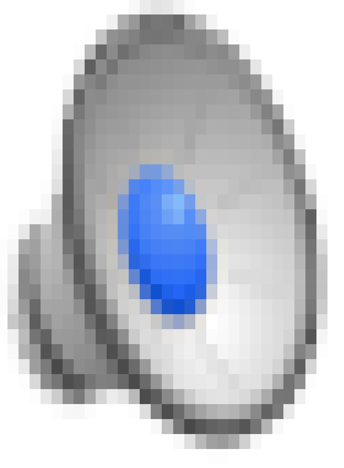
Expectimax вводит в дерево игры **узлы жеребьевки** (chance node), вместо узлов в которых выбирается минимальная оценка. При этом в узлах жеребьевки вычисляется **ожидаемая оценка** в виде взвешенного среднего значения

$$V(s) = \sum_{\mathrm{i}} p(s_i \mid s)\, V(s_i),$$

где $p(s_i|s)$ – условная вероятность того, что данное недетерминированное действие с индексом i приведет к переходу из состояния s в s_i.

Минимакс - это частный случай Expectimax: узлы, возвращающие минимальную оценку - это узлы, которые присваивают вероятность 1 своему дочернему узлу с наименьшим значением и вероятность 0 всем другим дочерним узлам.

Видео Pacman Minimax



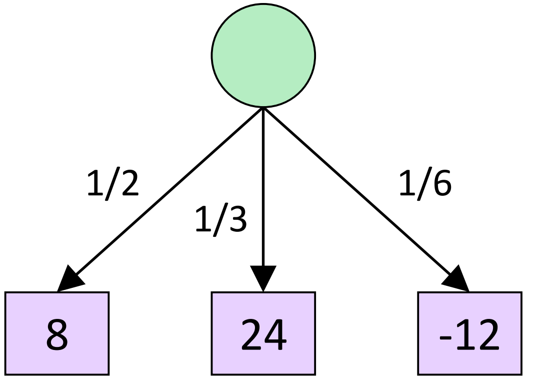
Видео Minimax vs Expectimax

Expectimax псевдокод

Поэтому псевдокод метода Expectimax очень похож на минимакс, за исключением необходимых изменений, связанных с вычислениями ожидаемой оценки полезности вместо минимальной оценки полезности:

ожидаемая оценка

```
def exp_value(state):
      v = 0
      for succ in successor(state):
            p=probability(succ)
            v+=p* value(succ))
       return v
```



v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

Expectimax

```
def value(state):
        if state соотв. концевой вершине: return eval(state)
        if  агент MAX: return max_value(state)
        if  агент EXP: return exp_value(state)

def max_value(state):                                   # максимальная оценка
        v = -∞
        for succ in successor(state):
                v=max(v, value(succ))
        return v

def exp_value(state):                                   # ожидаемая оценка
        v = 0
        for succ in successor(state):
                p=probability(succ)
                v+=p* value(succ))
        return v
```

Временная сложность Expectimax пропорциональна $\boldsymbol{O(b^m\, n^m)}$, где $\boldsymbol{n}$ — количество вариантов выпадения жребия.