Хабаров С.П.

Реализация интеллектуальных задач поиска в пространстве состояний в среде CLIPS

Индивидуальные задания

Санкт-Петербург
2014

СОДЕРЖАНИЕ

ВЕДЕНИЕ

Одним из распространенных направлений решения задач искусственного интеллекта является поиск в пространстве состояний (англ. state space search). Он объединяет в своем составе группу методов, характерным свойством которых является то, что они осуществляют последовательный просмотр конфигураций или состояний задачи с целью обнаружения целевого состояния, имеющего заданные характеристики.

Во многих задачах присутствует множество состояний, в которых может находиться определённый объект или система. При этом являются известными или могут быть априорно сформулированы правила перехода из одного состояния в другое. Такое часто встречается, например, в играх. Подобные задачи могут быть формально определены с помощью четырёх компонентов:

- *Начальное состояние* — это то состояние, в котором система находится в начальный момент;
- *Функция определения преемника* — описание возможных переходов из одного состояния в другое;
- *Проверка цели*, позволяющая определить, является ли данное состояние целевым;
- *Функция стоимости пути* — функция, назначающая каждому пути определённую стоимость.

Большинство алгоритмических формулировок поиска на графах использует понятие *явного графа*, который может быть представлен в виде матрицы смежности или списка смежности. В то время, как в алгоритмах поиска в пространстве состояний применяется понятие *неявного графа*.

Отличие состоит в том, что рёбра графа не хранятся в памяти явно, а порождаются на лету набором правил перехода. Определение графа пространства состояний включает в себя начальную вершину, множество целевых вершин и процедуру развёртывания вершины. Производительность алгоритма обычно оценивают по четырем основным показателям:

- *Полнота* — свойство алгоритма всегда находить решение, если оно существует;
- *Оптимальность* — свойство алгоритма всегда находить решение с наименьшей стоимостью;
- *Временная сложность* — оценка времени работы алгоритма;
- *Пространственная сложность* — оценка объёма памяти, необходимого для реализации алгоритма.

Решением задачи называется путь от начального состояния до целевого состояния, а оптимальным называется решение, имеющее наименьшую стоимость среди всех прочих решений. Выделяют два класса методов поиска в пространстве состояний: информированные и неинформированные.

Неинформированные методы поиска (методы слепого поиска, методы грубой силы) не используют никакой информации о конкретной задаче. Они шаг за шагом просматривают все состояния до тех пор, пока не будет найдено решение.

Информированные методы поиска (эвристические методы) пользуются дополнительной информацией о конкретной задаче, что позволяет сократить перебор путём исключения заведомо бесперспективных вариантов. Такой подход ускоряет работу алгоритма по сравнению с полным перебором. Платой за это является отсутствие гарантии того, что выбрано правильное или наилучшее из всех возможных решение.

Большинство поисковых задач формулируются как задачи поиска в пространстве состояний пути от исходного состояния заданной задачи до целевого состояния путем повторения возможных преобразований. Для организации поиска в пространстве состояний обычно используют дерево поиска (или его еще более общую форму – граф), а алгоритмы поиска формулируются в терминах изменения состояний или прохода по узлам графа.

В теории графов и в направлениях, связанных с решением задач искусственного интеллекта [1,2,18], используются такие алгоритмы, как поиск в ширину, поиск в глубину, алгоритм Дейкстры [2,14,15,16,17] и ряд других, среди которых выделяют направление так называемых эвристических алгоритмов.

Эвристический алгоритм — это алгоритм решения задачи, правильность которого для всех возможных случаев не доказана, но про который известно, что он даёт достаточно хорошее решение в большинстве случаев. Проще говоря, эвристика — это не полностью математически обоснованный, но при этом практически полезный алгоритм, который обладает рядом особенностей: не гарантирует поиск лучшего решения, не гарантирует нахождение решения, даже если оно заведомо существует (возможен «пропуск цели»), может дать неверное решение в некоторых случаях.

Эти алгоритмы широко применяются для решения задач высокой вычислительной сложности, когда вместо полного перебора вариантов, занимающего существенное время, а иногда технически невозможного, применяется значительно более быстрый, но недостаточно обоснованный теоретически алгоритм. Различные эвристические подходы находят применение в системах распознавание образов, в антивирусных программах, компьютерных играх и т.д. Например, программы, играющие в шахматы, проводят середину игры, основываясь, преимущественно, на эвристических алгоритмах. Если в дебюте используют базы данных, в эндшпиле — таблицы Налимова, то в миттельшпиле часто количество возможных ходов исключает полный перебор, а точных алгоритмов правильной игры не существует. Эвристические методы основаны на интеллектуальном поиске стратегий компьютерного решения проблемы с использованием нескольких альтернативных подходов.

1. АЛГОРИТМЫ ПОИСКА В ПРОСТРАНСТВЕ СОСТОЯНИЙ

Интеллектуальные системы, которые используют поиск в пространстве состояний, имеют специфическую организацию. Она состоит в том, что эти системы осуществляют поиск цели (конечного состояния) на основе исходных посылок и набора фактов, которые характеризуют некоторое начальное состояние. При этом поиск конечного состояния выполняется автоматически на основе, реализованной в системе стратегии поиска, которая обеспечивает возможность выбора и позволяет выполнять шаги от начального состояния к новым состояниям, более или менее близким к цели (рис. 1.1).

Рис. 1.1. Организация процедуры поиска.

Таким образом, реализованные в таких системах стратегии поиска отыскивают цель, «шагая» от одного состояния системы к другому, обеспечивая при этом распознавание ситуаций, когда они находят цель или попадают в тупик. Как правило, эти системы на промежуточных стадиях вычисляют некоторое число (критерий), посредством которого программа поиска оценивает свой ход и определяет дальнейшее направление поиска требуемого конечного состояния. При этом цель может быть одна или же может иметься некоторый набор приемлемых целей (конечных состояний).

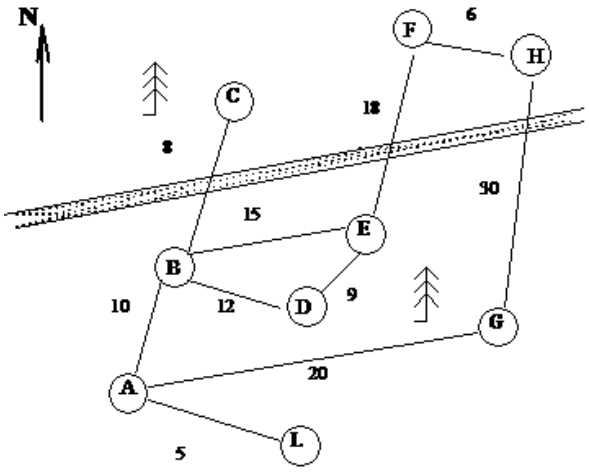


Рис. 1.2. Исходная карта дорог.

Рассмотрим процесс поиска на конкретном примере. Допустим, что есть карта дорог (рис. 1.2) и требуется найти маршрут движения из конкретного города в любой другой.

Система, построенная на основе знаний об этой карте дорог, должна поработать с картой и спланировать наше путешествие. Например из города «A» в город «F». При решении представленной задачи компьютер преобразует исходную карту и представит ее в виде дерева поиска, которое без учета циклов будет иметь вид как на рис. 1.3.

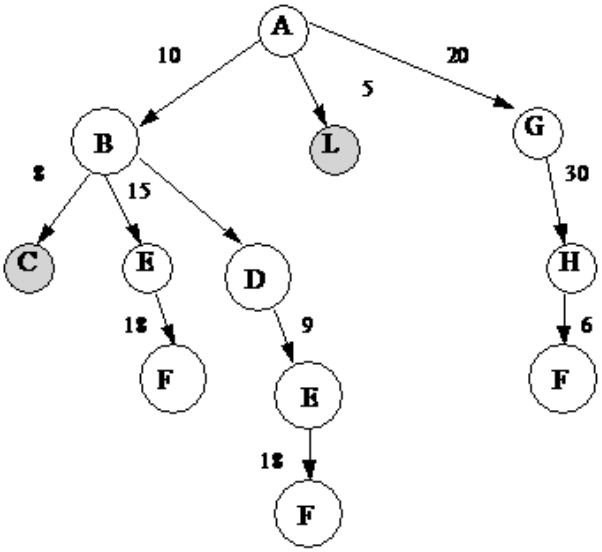


Рис. 1.3. Дерево поиска, построенное на базе исходной карты

Дерево поиска показывает все возможные варианты выбора при движении из «A» (начальное состояние), а также и все варианты выбора в каждом из промежуточных пунктов (состояний). Место, откуда начинается поиск, находится в вершине дерева поиска. Все дороги, начинаясь в «A», либо приводят в тупик, либо заканчиваются в «F».

Задача компьютера состоит в том, чтобы найти подходящую дорогу из вершины дерева к конечному состоянию (цели) в его нижней части. Человек при взгляде на дерево поиска сразу же выберет подходящую дорогу, но компьютер на это не способен. Какую стратегию поиска применит компьютер?

Рассмотрим пример, который заставит нас думать как компьютер. Предположим, что мы находимся в темной комнате, на стене весит рисунок в виде дерева поиска. У нас в руках фонарик, который может освещать всего два соседних пункта. Нам известно, где расположена вершина дерева, откуда надо начать поиск. Как наиболее эффективно перемещать луч фонарика, чтобы найти маршрут до пункта «F»?

При отсутствии какой-либо дополнительной информации компьютер для поиска использует стратегию «грубой силы». При этом программа поиска осуществляет поиск цели, идя от состояния к состоянию, и систематически исследуя все возможные пути. Для поиска каждого последующего шага применяется простая механическая стратегия, которая имеет две основные разновидности: поиск в глубину и поиск в ширину.

1.1. Стратегия поиска в глубину

Стратегия поиска в глубину, как следует из ее названия, состоит в том, чтобы идти "вглубь" графа, насколько это возможно и предполагает полное исследовании, одного из возможных вариантов, до изучения других вариантов.

Поиск выполняется по несложной методике, при которой исследуются все ребра, выходящие из вершины, открытой последней, и вершина покидается, только когда не остается неисследованных ребер.

Например, компьютер всегда первой исследует неизвестную левую ветвь дерева. Когда процесс поиска заходит в тупик, он возвращается вверх в последний пункт выбора, где имеются неизученные альтернативные варианты движения, и затем осуществляется следующий вариант выбора (рис. 1.4).

Этот процесс будет повторяется до тех пор, пока не будут открыты все вершины пространства состояний. Считается, что поиск в глубину в основном подходит для решения задач, в которых все пути поиска от вершины дерева до его основания имеют одинаковую длину.

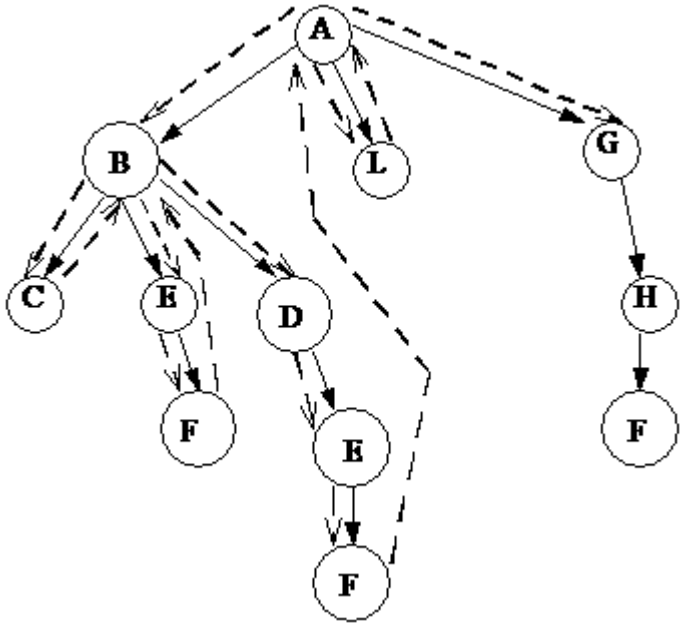


Рис. 1.4. Стратегия поиска в глубину

1.2. Стратегия поиска в ширину

Стратегия поиска в ширину предусматривает переход в первую очередь к вершинам, ближайшим к стартовой вершине (т.е. отстающим от нее на одну связь), затем к вершинам, отстающим на две связи, затем на три и т. д., пока не будет найдена целевая вершина (рис. 1.5).

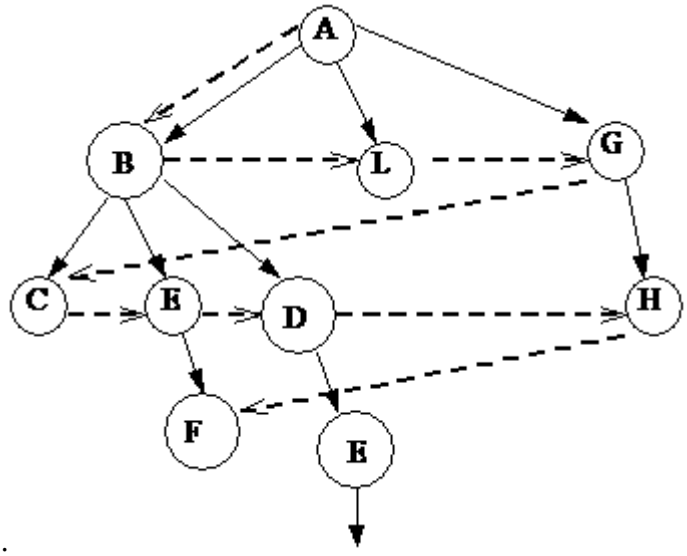


Рис. 8.5. Стратегия поиска в ширину.

Программа поиска продвигается по дереву решений слева направо, расширяя каждый из маршрутов, и отбрасывая те из них, которые являются тупиковыми. Таким образом, поиск в ширину пригоден при решении проблем, где ветви поиска в дереве (от

вершины до основания) имеют неодинаковую длину, и не существует указаний на то, какая именно дорога приводит к цели.

Обе эти стратегии предполагают последовательный перебор возможных вариантов. Поиск будет более эффективным, если некоторый механизм в пунктах выбора сам сможет делать наиболее желательный выбор. Это, так называемая, эвристика поиска. Эвристика - это эмпирическое правило, с помощью которого человек - эксперт в отсутствии формулы или алгоритма пытается осуществить свои намерения.

1.3. Стратегия эвристического поиска

Поиск на графах, как правило, невозможен без решения проблемы комбинаторной сложности, возникающей из-за быстрого роста числа альтернатив. Эффективным средством борьбы с этим служит эвристический поиск. Один из путей использования эвристической информации о задаче – это получение численных эвристических оценок для всех вершин пространства состояния. Численная оценка вершины указывает, насколько данная вершина перспективна с точки зрения достижения цели.

Идея состоит в том, чтобы всегда продолжать поиск, начиная с наиболее перспективной вершины, выбранной из всего множества кандидатов. Существует целый ряд эвристических методов. Простейший из них носит название «взбирание на гору», суть которого состоит в следующем:

- если есть приемлемые варианты выбора, надо выбрать наилучший из них, используя любой критерий рассуждения;
- если попали в тупик, надо вернуться в последнее место, где имеются альтернативные варианты выбора, и сделать иной выбор, наиболее близкий к наилучшему выбору.

Чтобы алгоритм поиска маршрута выполнял эвристический поиск, он должна иметь некоторую базу для осуществления выбора. Предположим, в качестве базы алгоритм будет использовать соответствие направления каждой дороги определенному азимуту: предпочтение будет отдаваться той дороге, которая более всего совпадает по направлению с прямой, соединяющей стартовую и целевую вершины (рис. 1.6).

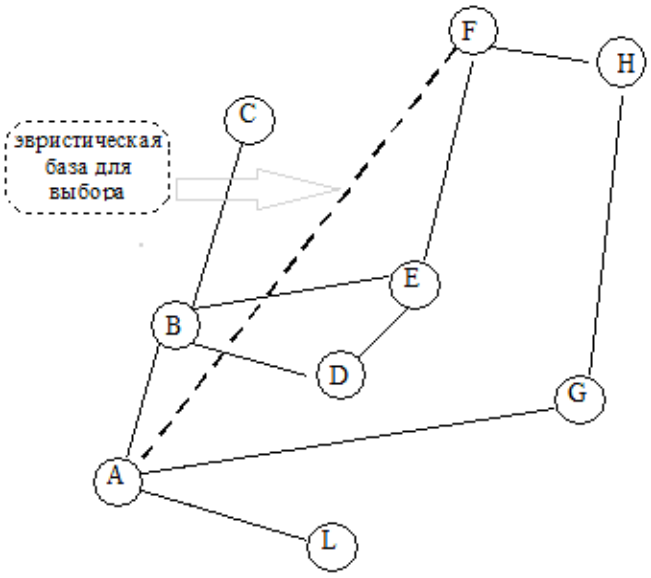


Рис. 1.6. Эвристическая стратегия по методу «взбирание на гору»

Такой подход не всегда действенен, но это хорошая стратегия. В нашем примере программа исследует маршрут «A» —> «B» —> «C» и найдет тупик, затем вернется назад, в то место, где существует альтернатива выбору, т.е. в пункт «B», а затем сразу отправится по маршруту «B» —> «E» —> «F», который приведет к цели.

Мы рассмотрели разные варианты поиска, но возникает вопрос: откуда взялось само дерево поиска? Обычно алгоритмы, входящие в состав интеллектуальных систем, сами создают дерево поиска на основе предлагаемых ей начальных сведений, причем делают это постепенно в процессе самого поиска.

1.4. Реализованные в CLIPS стратегии поиска

CLIPS является одной из оболочек, позволяющих проектировать и разрабатывать интеллектуальные системы. В том числе и тех, что базируются на поиске в пространстве состояний. CLIPS поддерживает семь различных стратегий разрешения конфликтов: стратегия глубины (depth strategy), стратегия ширины (breadth strategy), стратегия упрощения (simplicity strategy), стратегия усложнения (complexity strategy), LEX (LEX strategy), MEA (MEA strategy) и случайная стратегия (random strategy).

По умолчанию в CLIPS установлена стратегия глубины. Текущая стратегия может быть изменена и установлена командой set-strategy, которая переупорядочит текущий план решения задачи, базируясь на новой стратегии. Подробно с реализованными в CLIPS стратегиями можно познакомиться по специальной литературе [3,4,12,13]. Здесь ограничимся только кратким пояснением двух наиболее часто используемых стратегий:

- Стратегия глубины — устанавливается командой (set-strategy depth).

Только что активированное правило помещается выше всех правил с таким же приоритетом. Например, допустим, что факт А активировал правила 1 и 2 и факт Б активировал правило 3 и правило 4, тогда, если факт А добавлен перед фактом Б, в плане решения задачи правила 3 и 4 будут располагаться выше, чем правила 1 и 2. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

- Стратегия ширины — устанавливается командой (set-strategy breadth).

Только что активированное правило помещается ниже всех правил с таким же приоритетом. Например, допустим, что факт А активировал правила 1 и 2 и факт Б активировал правила 3 и 4, тогда, если факт А добавлен перед фактом В, в плане решения задачи правила 1 и 2 будут располагаться выше, чем правила 3 и 4. Однако позиция правила 1 относительно правила 2 и правила 3 относительно правила 4 будет произвольной.

Узнать текущую стратегию можно, используя команду (get-strategy).

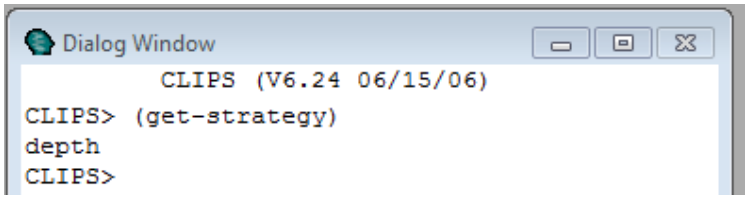


Рис. 1.7. Результат выполнения команды (get-strategy).

Изменить порядок применения правил при установленной стратегии можно путем назначения правилам индивидуальных приоритетов — (declare (salience <приоритет>))

1.5. Индивидуальное задание “Числовой ребус”.

Содержание задания:

1. Решить задачу в среде CLIPS,
2. Найти соответствие между буквами и цифрами.
3. Разным буквам соответствуют разные цифры.
4. В результате выполнения арифметических действий над числами, которые заменят буквы ребуса должно получиться число после знака = .
5. Число не может начинаться с цифры 0.

Варианты заданий:

1. ДЕДКА + БАБКА + РЕПКА = СКАЗКА
2. ВЕТКА + ВЕТКА = ДЕРЕВО
3. ЛАДЬЯ + ЛАДЬЯ = ФЕРЗЬ
4. СИНУС + СИНУС + КОСИНУС = ТАНГЕНС
5. CROSS + ROADS = DANGER
6. ДВЕСТИ + ТРИСТА = ПЯТЬСОТ
7. МАГНИЙ + ТАНТАЛ = МЕТАЛЛЫ
8. НАТАША + ТОНЯ = СЕСТРЫ
9. КОРОВА + ДОЯРКА + ТРАВА = МОЛОКО
10. ШЕПНУЛ*5 = КРИКНУЛ
11. ПЛОМБА*5 = АПЛОМБ
12. КНИГА*3 = НАУКА
13. АТАКА+УДАР+УДАР = НОКАУТ
14. SIX + SEVEN + SEVEN = TWENTY
15. АНДРЕЙ+ЖАННА=ДРУЖБА

Замечание.

До выполнения задания рассмотреть приведенные примеры. Разобраться в чем по структуре и процедуре поиска отличаются два варианта реализации. Провести полное документирование вариантов, вставив в тексты программ все необходимые комментарии. Рассмотреть поиск в глубину и в ширину.

Первая версия реализации программы ”числовой ребус”

```
(defrule startup
   =>
   (printout t "The problem is" t)
   (printout t "   xxxxxx" " | KOROVA" t)
   (printout t " + xxxxxx" " | DOJRKA" t)
   (printout t "    xxxxx" " |  TRAVA" t)
   (printout t "   --------- | ------" t)
   (printout t " = xxxxxx" " | MOLOKO" t)
   (assert  (number 0) (number 1) (number 2) (number 3) (number 4)
            (number 5) (number 6) (number 7) (number 8) (number 9)
            (letter A) (letter D) (letter J) (letter K) (letter L)
            (letter M) (letter O) (letter R) (letter T) (letter V)
   )
)

(defrule generate
   (number ?x)
   (letter ?y)
   =>
   (assert (combination ?y ?x))
)

(defrule find
   (combination A ?a)
   (combination O ?o&~?a)
   (test (= (mod (+ ?a ?a ?a) 10) ?o))

   (combination V ?v&~?a&~?o)
```

```
   (combination K ?k&~?v&~?a&~?o)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10))
                    100)
         (+ (* 10 ?k) ?o)))

   (combination R ?r&~?a&~?o&~?v&~?k)
   (combination L ?l&~?a&~?o&~?v&~?k&~?r)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10)
                    (* (+ ?o ?r ?a) 100))
                    1000)
         (+ (* 100 ?o) (* 10 ?k) ?o)))

   (combination D ?d&~?a&~?o&~?v&~?k&~?r&~?l)
   (combination J ?j&~?d&~?a&~?o&~?v&~?k&~?r&~?l)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10)
                    (* (+ ?o ?r ?a) 100)
                    (* (+ ?r ?j ?r) 1000))
                    10000)
         (+ (* 1000 ?l) (* 100 ?o) (* 10 ?k) ?o)))

   (combination M ?m&~?d&~?a&~?o&~?v&~?k&~?r&~?l&~?j)
   (combination T ?t&~?d&~?a&~?o&~?v&~?k&~?r&~?l&~?j&~?m)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10)
                    (* (+ ?o ?r ?a) 100)
                    (* (+ ?r ?j ?r) 1000)
                    (* (+ ?o ?o ?t) 10000)
                    (* (+ ?k ?d) 100000))
                    1000000)
  (+ (* 100000 ?m) (* 10000 ?o) (* 1000 ?l) (* 100 ?o) (* 10 ?k) ?o)))

   =>
   (printout t "A Solution is:" t)
   (printout t "  A = " ?a t)
   (printout t "  D = " ?d t)
   (printout t "  J = " ?j t)
   (printout t "  K = " ?k t)
   (printout t "  L = " ?l t)
   (printout t "  M = " ?m t)
   (printout t "  O = " ?o t)
   (printout t "  R = " ?r t)
   (printout t "  T = " ?t t)
   (printout t "  v = " ?v t)
   (printout t t)
   (printout t "   " ?k ?o ?r ?o ?v ?a " | KOROVA" t)
   (printout t " + " ?d ?o ?j ?r ?k ?a " | DOJRKA" t)
   (printout t "    " ?t ?r ?a ?v ?a " |  TRAVA" t)
   (printout t "   ------ | ------" t)
   (printout t " = " ?m ?o ?l ?o ?k ?o " | MOLOKO" t)
)
```

Вторая версия реализации программы ”числовой ребус”.

```
;; задается шаблон
(deftemplate Letter
   (slot A(type NUMBER)(default -1))
   (slot D(type NUMBER)(default -1))
   (slot J(type NUMBER)(default -1))
   (slot K(type NUMBER)(default -1))
   (slot L(type NUMBER)(default -1))
   (slot M(type NUMBER)(default -1))
   (slot O(type NUMBER)(default -1))
   (slot R(type NUMBER)(default -1))
   (slot T(type NUMBER)(default -1))
   (slot V(type NUMBER)(default -1))
)

(deffacts Numbers
   (number 0)
   (number 1)
   (number 2)
   (number 3)
   (number 4)
   (number 5)
   (number 6)
   (number 7)
   (number 8)
   (number 9)
)

(defrule startup
   (declare (salience 1000))
   =>
   (printout t "  The problem is:" t)
   (printout t "    xxxxxx | KOROVA" t)
   (printout t "  + xxxxxx | DOJRKA" t)
   (printout t "     xxxxx |  TRAVA" t)
   (printout t " --------- | ------" t)
   (printout t "  = xxxxxx | MOLOKO" t)
)

(defrule column_0
;   (declare (salience 100))
   (number ?a)
   (number ?o&~?a)
   (test (= (mod (+ ?a ?a ?a) 10) ?o))
    =>
   (assert (Letter (A ?a) (O ?o)))
)

(defrule column_1
;   (declare (salience 90))
   (Letter (A ?a) (O ?o) (V -1) (K -1))
   (number ?v&~?a&~?o)
```

```
   (number ?k&~?a&~?o&~?v)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10))
             100)
     (+ (* 10 ?k) ?o))
    )
    =>
   (assert (Letter (A ?a) (O ?o) (V ?v) (K ?k)))
)

(defrule column_2
;   (declare (salience 80))
   (Letter (A ?a) (O ?o) (V ?v) (K ?k) (R -1) (L -1))
   (number ?r&~?a&~?o&~?v&~?k)
   (number ?l&~?a&~?o&~?v&~?k&~?r)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10)
                    (* (+ ?o ?r ?a) 100))
             1000)
             (+ (* 100 ?o) (* 10 ?k) ?o)))
    =>
   (assert (Letter (A ?a) (O ?o) (V ?v) (K ?k) (R ?r) (L ?l)))
)

(defrule column_3
;   (declare (salience 70))
   (Letter (A ?a) (O ?o) (V ?v) (K ?k) (R ?r) (L ?l))
   (number ?d&~?a&~?o&~?v&~?k&~?r&~?l)
   (number ?j&~?d&~?a&~?o&~?v&~?k&~?r&~?l)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10)
                    (* (+ ?o ?r ?a) 100)
                    (* (+ ?r ?j ?r) 1000))
             10000)
             (+ (* 1000 ?l) (* 100 ?o) (* 10 ?k) ?o)))
    =>
   (assert (Letter (A ?a) (O ?o) (V ?v) (K ?k)
                   (R ?r) (L ?l) (D ?d) (J ?j)))
)

(defrule column_4_5
;   (declare (salience 60))
   (Letter (A ?a) (O ?o) (V ?v) (K ?k) (R ?r) (L ?l) (D ?d) (J ?j))
   (number ?m&~?d&~?a&~?o&~?v&~?k&~?r&~?l&~?j)
   (number ?t&~?d&~?a&~?o&~?v&~?k&~?r&~?l&~?j&~?m)
   (test (= (mod (+ ?a ?a ?a
                    (* (+ ?v ?k ?v) 10)
                    (* (+ ?o ?r ?a) 100)
                    (* (+ ?r ?j ?r) 1000)
                    (* (+ ?o ?o ?t) 10000)
                    (* (+ ?k ?d  0) 100000))
             1000000)
```

```
             (+ (* 100000 ?m) (* 10000 ?o) (* 1000 ?l) (* 100 ?o) (*
10 ?k) ?o)))
    =>
   (assert (Letter (A ?a) (O ?o) (V ?v) (K ?k) (R ?r)
                   (L ?l) (D ?d) (J ?j) (M ?m) (T ?t)
           )
   )
)

(defrule result
   (Letter (A ?a) (O ?o) (V ?v) (K ?k) (R ?r) (L ?l) (D ?d) (J ?j) (M
?m) (T ?t))
   (test (> ?m 0))
   =>
   (printout t crlf)
   (printout t "A Solution is:" t)
   (printout t "  A = " ?a t)
   (printout t "  D = " ?d t)
   (printout t "  J = " ?j t)
   (printout t "  K = " ?k t)
   (printout t "  L = " ?l t)
   (printout t "  M = " ?m t)
   (printout t "  O = " ?o t)
   (printout t "  R = " ?r t)
   (printout t "  T = " ?t t)
   (printout t "  v = " ?v t)
   (printout t t)
   (printout t "    " ?k ?o ?r ?o ?v ?a " | KOROVA" t)
   (printout t " + " ?d ?o ?j ?r ?k ?a " | DOJRKA" t)
   (printout t "     " ?t ?r ?a ?v ?a " |  TRAVA" t)
   (printout t "   ------ | ------" t)
   (printout t " = " ?m ?o ?l ?o ?k ?o " | MOLOKO" t)
)
```

2. РЕАЛИЗАЦИЯ ЭВРИСТИЧЕСКИХ АЛГОРИТМОВ ПОИСКА НА ПРИМЕРЕ АЛГОРИТМА A*

Целью данной лабораторной работы является:

1. Знакомство с алгоритмами поиска в пространстве состояний.
2. Изучение одного из эвристических алгоритмов – алгоритма A^*.
3. Получение навыков реализации поисковых алгоритмов в среде CLIPS.

2.1. Эвристический алгоритм поиска A*

Одним из наиболее распространенных эвристических алгоритмов поиска является так называемый алгоритм A^*, где используются априорные оценки стоимости пути до целевого состояния, что обеспечивает высокую эффективность поиска. Алгоритм A^* предполагает наличие двух списков вершин графа: открытого и закрытого. В первом находятся вершины, еще не проверенные алгоритмом, а во втором те вершины, которые уже встречались в ходе поиска решения. На каждом новом шаге, из списка открытых вершин выбирается вершина с наименьшим весом.

Основная идея алгоритма [4] состоит в использовании для каждого узла *n* на графе пространства состояний оценочной функции вида

$$f(n) = g(n) + h(n).$$

Здесь *g(n)* – соответствует расстоянию на графе от узла n до начального состояния, а *h(n)* – оценка расстояния от узла n до узла, представляющее конечное (целевое) состояние. Чем меньше значение оценочной функции *f(n)* тем лучше, т.е. узел *n* лежит на более коротком пути от исходного состояния к целевому. Идея алгоритма в том, чтобы с помощью *f(n)* отыскать кратчайший путь на графе от исходного состояния к целевому.

Описание алгоритма.

Введем следующие обозначения:

s – узел начального состояния;

g – узел конечного (целевого) состояния;

OPEN – список выбранных, но необработанных узлов;

CLOSED – список обработанных узлов.

Шаги алгоритма:

1. OPEN := {s}.
2. Если OPEN := {}, то прекратить выполнение. Пути к целевому состоянию на графе не существует.
3. Удалить из списка OPEN узел *n*, для которого *f(n)< f(m)* для любого узла *m*, уже присутствующего в списке OPEN, и перенести его в список CLOSED.
4. Сформировать список очередных узлов, в который возможен переход из узла *n*, и удалить из него все узлы, образующие петли. С каждым из оставшихся узлов связать указатель на узел *n*.
5. Если в сформированном списке очередных узлов присутствует узел *g*, то следует завершить выполнение. Сформировать результат - путь, порожденный прослеживанием указателей от узла *g* до узла *s*.
6. В противном случае для каждого очередного узла *n'*, включенного в список выполнить следующую последовательность операций:

   6.1. Вычислить $f(n')$.

6.2. Если *n′* не присутствует ни в списке *OPEN*, ни в списке *CLOSED*, то добавить его в список, присоединить к нему оценку *f(n′)* и установить обратный указатель на узел *n*.
6.3. Если *n′* уже присутствует в списке *OPEN* или в списке *CLOSED*, то сравнить новое значение *f(n′) = new* с прежним *f(n′) = old*.
6.4. Если *old* < *new*, прекратить обработку нового узла.
6.5. Если *old* > *new*, заменить новым узлом прежний в списке, причем, если прежний узел был в списке *CLOSED*, перенести его в список *OPEN*.

2.2. Пример решения задачи поиска в пространстве состояний

Возьмем классическую головоломку - игру в "Восемь". В этой игре используется 8 пронумерованных фишек, которые могут перемещаться по игровому полю 3x3. Цель состоит в том, чтобы из некоторого случайного расположения фишек (рис. 2.1) перейти к фиксированному упорядоченному расположению фишек.

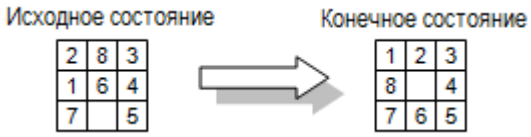


Рис.2.1. Цель игры в ”Восемь”.

Если представить пространство поиска в игре "Восемь" в виде дерева, то *g(n)* - это глубина от первой вершины до *n*-й вершины. В качестве *h(n)* можно, например, выбрать число фишек, стоящих не на своих местах. Стратегия прохождения вершин - по минимуму оценочной функции.

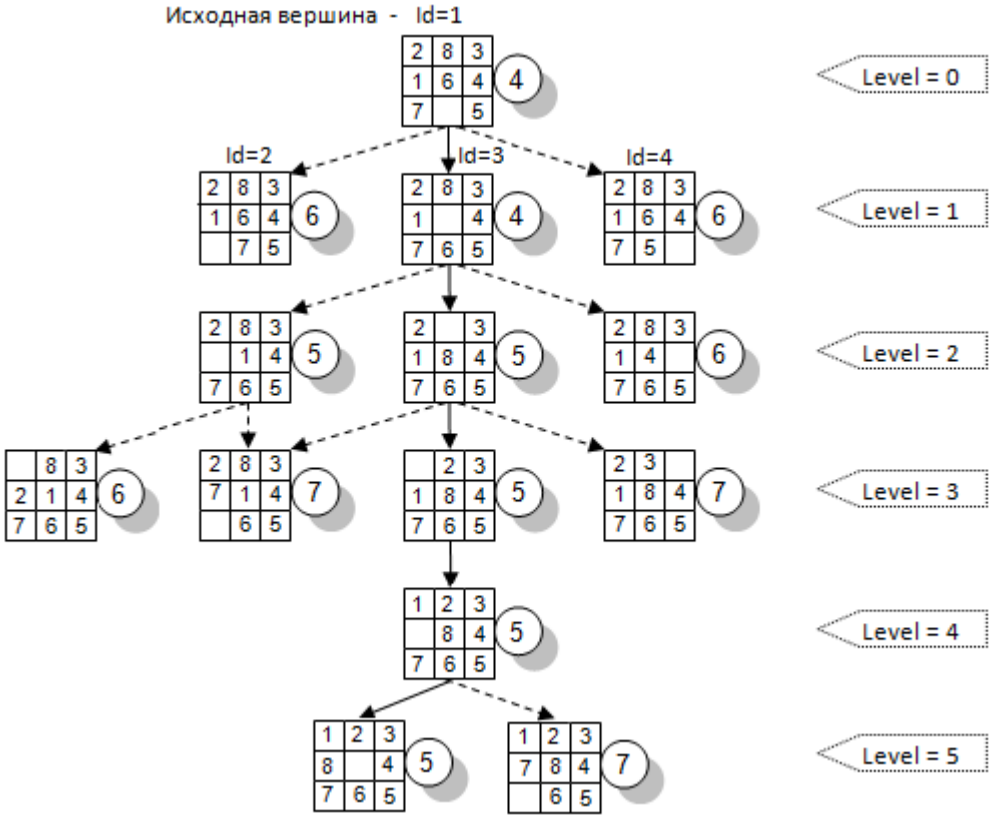


Рис.2.2. Поиск решения в игре ”Восемь” по алгоритму A*.

На рис. 2.2 показан результат поиска в пространстве состояний, при котором на каждом шаге работы алгоритма выбирается вершина с наименьшим для данного уровня графа значением оценочной функции. Для этой вершины формируются все возможные

дочерние вершины следующего уровня графа за исключением уже появившихся вершин. Эта процедура повторяется до достижения целевого состояния.

На этом же рисунке представлены результаты вычисления оценочной функции *f(n)* для каждой из вершин. Эти значения отображаются в кружке справа от вершины. Видно, что найден решающий путь, при котором использование оценочной функции позволило раскрыть значительно меньше вершин чем при полном поиске на графе.

При реализации решения данной задачи на CLIPS необходимо будет создать шаблон ситуации игры, который будет характеризовать состояния каждой из вершин графа:

```
(deftemplate Field
   (slot s11  (type NUMBER))
   (slot s12  (type NUMBER))
   (slot s13  (type NUMBER))
   (slot s21  (type NUMBER))
   (slot s22  (type NUMBER))
   (slot s23  (type NUMBER))
   (slot s31  (type NUMBER))
   (slot s32  (type NUMBER))
   (slot s33  (type NUMBER))
   (slot Level(type NUMBER))              ; Уровень дерева
   (slot Id   (type NUMBER)(default 0))   ; Уникальный номер вершины
   (slot State(type NUMBER)(default 0))   ; Принадлежность списку узлов
   (slot From (type NUMBER))              ; Номер вершины родителя
   (slot Exp  (type NUMBER))              ; Значение целевой функции
)
```

Слоты s11, s12, s13, s21, s22, s23, s31, s32, s33 соответствуют ячейкам игрового поля, в которых задаются номера соответствующих фишек. Если в ячейке фишка отсутствует, то значение соответствующего слота будет равно 0.

Слот Level - определяет уровень дерева решений, то есть число шагов от начальной ситуации к текущей, а Id – это уникальный номер состояния. Слот State - определяет принадлежность данной ситуации множеству *OPEN* если его значение равно 0 и множеству *CLOSED* – при значении 1. Если ситуация является промежуточной на пути к решению, то данному слоту будет присвоено значение 2. Это должно произойти в тот момент, когда цель будет достигнута. Слот From – соответствует Id номера ситуации, из которой был осуществлен переход в данное состояние. Слот Exp – предназначен для хранения значение целевой функции, вычисленной для данной ситуации.

Так как идентификатор Id каждой текущей ситуации должен быть уникальным, то целесообразно присваивать новой ситуации идентификатор на единицу больший, чем Id предыдущей ситуации. Хранить номер предыдущей ситуации будем в глобальной переменной ?*Id*, а вычислять Id новой ситуации будем, используя функцию Get_Id():

```
(defglobal
   ?*Id* = 0
)
(deffunction Get_Id()
   (bind ?*Id* (+ ?*Id* 1))
   ?*Id*
)
```

Для вычисления эвристической оценки текущей ситуации определим функцию, параметрами которой будет число шагов от начальной ситуации до текущей ситуации, а также номера фишек, соответствующие позициям на игровом поле. Первый параметр

позволит определить составляющую $g(n)$ оценочной функции $f(n)$, а другие параметры будут использованы для вычисления составляющей *h(n)* этой же функции.

```
(deffunction W(?Level ?s11 ?s12 ?s13 ?s23 ?s33 ?s32 ?s31 ?s21)
   (bind ?a ?Level)
   (if (not (= ?s11 1)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s12 2)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s13 3)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s23 4)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s33 5)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s32 6)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s31 7)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s21 8)) then (bind ?a (+ 1 ?a)) )
   ?a
)
```

Используя конструктор deffacts определим в базе фактов два факта, которые будут инициализировать исходное состояние решения задачи. Факт Field задает начальное расположение фишек на игровом поле, значение нулевого уровня дерева решений для этой ситуации, вычисляет значение оценочной функции и уникальный номер для этой ситуации. Факт min исходныйи инициализируем минимум:

```
(deffacts start
   (Field (s11 2) (s12 8) (s13 3)        ; Исходное положение
          (s21 1) (s22 6) (s23 4)        ; фишек на игровом
          (s31 7) (s32 0) (s33 5)        ; поле
          (Level 0)                      ; Это корень дерева
          (From 0)
          (Exp (W 0 2 8 3 4 5 0 7 1))    ; Значение оценочной функции
          (Id (Get_Id))                  ; Уникальный номер вершины
   )
   (min (W 0 2 8 3 4 5 0 7 1))
)
```

Определив параметры начальной инициализации и структуру, описывающую состояния каждой из вершин графа, далее можно перейти к реализации эвристического алгоритм поиска. С этой целью необходимо создать правила для следующих действий:

- Если решений найдено, то выделить его.
- Удалить ситуации, которые не являются промежуточными между начальной и конечной ситуацией.
- Останов программы, если найдено решение или решений нет.
- Удаление повторяющихся ситуаций.
- Поиск *min* - минимального значения из значений целевой функции ситуаций множества OPEN.
- Порождение новых всевозможных ситуаций из текущей ситуации, для которой значение целевой функции равно *min*.

Обратим внимание, что список действий составлен в порядке приоритета, то есть прежде чем приступить к следующему шагу, необходимо выполнить предыдущий. Аналогично в программе будут расставлены приоритеты соответствующих правил.

Считается, что решение найдено если на каком-то этапе решения задачи была порождена ситуация соответствующая конечной. В этом случае необходимо выделить все промежуточные ситуации от конечной до начальной. Данные действия выполняются в следующих правилах:

```
(defrule start_select_answer
    ;;если нашли конечное решение, то выделяем его, задавая State=2
   (declare (salience 500))
   ?f <-(Field (s11 1) (s12 2) (s13 3)
               (s21 8) (s22 0) (s23 4)
               (s31 7) (s32 6) (s33 5)
               (State ~2) (From ?Id)
         )
   =>
   (printout t "start select answer Id-" (fact-slot-value ?f Id) crlf)
   (modify ?f (State 2))
)

(defrule select_answer
   ;выделяем все вершины дерева решение, присваивая им State=2
   (declare (salience 500))
   (Field (State 2) (From ?Id))
   ?f <- (Field (Id ?Id) (State ~2))
   =>
   (modify ?f (State 2))
   (printout t "select answer Id=" ?Id " - " ?f crlf)
)
```

В первом правиле помечается ситуация, соответствующая найденному конечному решению, что достигается присваиванием слоту State значения 2. Второе правило на основе значения слота From помеченной ситуации ищет непомеченную ситуацию, родительскую по отношению к уже помеченной ситуации. Если такая ситуации есть, то она включается в список помеченных (State = 2), образующих дерево решения.

Если предыдущие правила, как более приоритетны, больше не выполняются, но при этоми целевое решение найдено, то необходимо из базы фактов удалить все лишние промежуточные факты:

```
(defrule delete_not_answer
   (declare (salience 400))
   (Field (State 2))
   ?f <- (Field (State ~2))
   =>
   (retract ?f)
)
```

Первая предпосылка этого правила гарантирует, что его действие будет выполнено только в случае, когда целевое решение найдено, а вторая предпосылка позволяет выбрать для удаления все факты не соответствующие решению.

Возможность окончания работы программы и ее останова будет реализовано двумя следующими правилами:

```
 (defrule Stop_1
   ;; делаем останов если решений нет
   (declare (salience 200))
   (Field(State ?x))
   (not (Field(State 0|2)))
   =>
   (halt)
   (printout t "no solutions" crlf)
)
```

```
(defrule Stop_2
   ;; делаем останов если решение есть
   (declare (salience 200))
   (Field(State 2))
   =>
   (facts)
   (halt)
   (printout t "fined solution" crlf)
)
```

Действия первого правила выполняются, если решение не найдено, то есть в базе фактов отсутствуют ситуации помеченные 0 или 2. Второе правило выполняется, если решение найдено.

Для исключения из множества *OPEN* ситуаций уже присутствующих в множестве *CLOSED* создадим правило, которое удаляет соответствующие факты из базы фактов:

```
(defrule move_circle
   (declare (salience 1000))
   ?f1 <- (Field (State 1) (s11 ?LT1) (s12 ?MT1) (s13 ?RT1)
                           (s21 ?LM1) (s22 ?MM1) (s23 ?RM1)
                           (s31 ?LB1) (s32 ?MB1) (s33 ?RB1)    )
   ?f0 <- (Field (State 0) (s11 ?LT0&:(= ?LT0 ?LT1))
               (s12 ?MT0&:(= ?MT0 ?MT1))  (s13 ?RT0&:(= ?RT0 ?RT1))
               (s21 ?LM0&:(= ?LM0 ?LM1))  (s22 ?MM0&:(= ?MM0 ?MM1))
               (s23 ?RM0&:(= ?RM0 ?RM1))  (s31 ?LB0&:(= ?LB0 ?LB1))
               (s32 ?MB0&:(= ?MB0 ?MB1))  (s33 ?RB0&:(= ?RB0 ?RB1))  )
   =>
   (retract ?f0)
   (printout t ?LT0 ?MT0 ?RT1 crlf ?LM0 ?MM0 ?RM0 crlf
               ?LB0 ?MB0 ?RB0 crlf )
)
```

Для поиска минимального значения создадим правило, которое будет выполняться, когда будет существовать ситуация из множества *OPEN* в которой значение целевой функции меньше заданной в факте ***min:***

```
(defrule fmd_min
   (declare (salience 150))
   ; приоритет ниже, чем у правила исключающего циклы
   ; и выше чем у правил порождения новых ходов
   ?fmin <- (min ?min)
   (Field (Exp ?E&:(< ?E ?min)) (State 0))
   =>
   (retract ?fmin)
   (assert (min ?E))
)
```

Для порождения новых ситуаций следует создать девять правил, каждое из которых соответствует одному из возможных в игре положений пустой клетки на игровом поле. Все этих правила служит для формирования пространства состояний и в каждом из них должны выполняться следующие действия:

- Ситуация для конкретного положения пустой клетки на игровом поле выбирается в качестве активной, если она присутствует в множества *OPEN* и имеет значение целевой функции равное *min.*
- Выбранная ситуация помечается как принадлежащая множеству *CLOSED.*

- Создаются всевозможные новые ситуации, которые доступны из этой ситуации.
- Вычисляется значение целевой функции одной из новых ситуаций и модифицируется соответствующим образом факте *min* в базе фактов**.**

Полный набор правил, формирующих пространство состояний, приведен в листинге программы. Так как принципы их реализации идентичны, то рассмотрим только одно из них, которое соответствует включению в список *CLOSED* ситуации с пустой верхней правой ячейкой игрового поля (рис. 2.3).

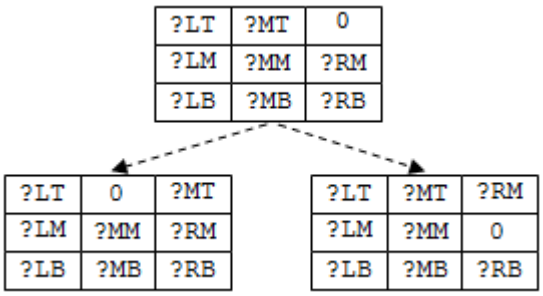


Рис.2.3. Порождение новых ситуаций для верхней правой ячейки игрового поля.

Условная часть данного правила выполняется, если данная ситуация присутствует в базе фактов, имеет значение State = 0 и целевая функция равна переменной ?fmin, значение которой получено из факта *min.* При этом происходит связывание переменных ?LT, ?MT, ?LM, ?MM, ?RM, ?LB, ?MB, ?RB с номерами фишек в соответствующих для данной ситуации позициях игрового поля.

Заключительная часть правил переводит текущую ситуацию в множество *CLOSED,* присваивая слоту State значение 1, затем вычисляет значение целевой функции для одного из новых состояний и формирует новый факт *min.* Затем формирует пространство состояний, порождая дочерние вершины для этой ситуации, добавляя новые факты в базу фактов, которые соответствуют элементам множества *OPEN*.

```
(defrule make_new_path_s13
  (declare (salience 100))
  ?fmin <-(min ?min)
  ?f <-(Field (State 0) (Level ?L) (Id ?Id) (Exp ?E&:(= ?E ?min))
              (s11 ?LT) (s12 ?MT)  (s13 0)
              (s21 ?LM) (s22 ?MM)  (s23 ?RM)
              (s31 ?LB) (s32 ?MB)  (s33 ?RB)  )
   =>
  (modify ?f(State 1))
  (bind ?a (W (+ 1 ?L) ?LT 0 ?MT ?RM ?RB ?MB ?LB ?LM))
  (retract ?fmin)
  (assert (min ?a))
  (assert (Field (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
              (s11 ?LT) (s12 0)   (s13 ?MT)
              (s21 ?LM) (s22 ?MM) (s23 ?RM)
              (s31 ?LB) (s32 ?MB) (s33 ?RB)    )
   )
  (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RM)
                 (s21 ?LM) (s22 ?MM) (s23 0)
                 (s31 ?LB) (s32 ?MB) (s33 ?RB)
                 (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                 (Exp (W (+ 1 ?L) ?LT ?MT ?RM 0 ?RB ?MB ?LB ?LM))  )
   )
)
```

Следует отметить, что все девять отмеченных выше правил имеют одинаковый приоритет, что обеспечивает случайность их использования в процесс работы системы продукций.

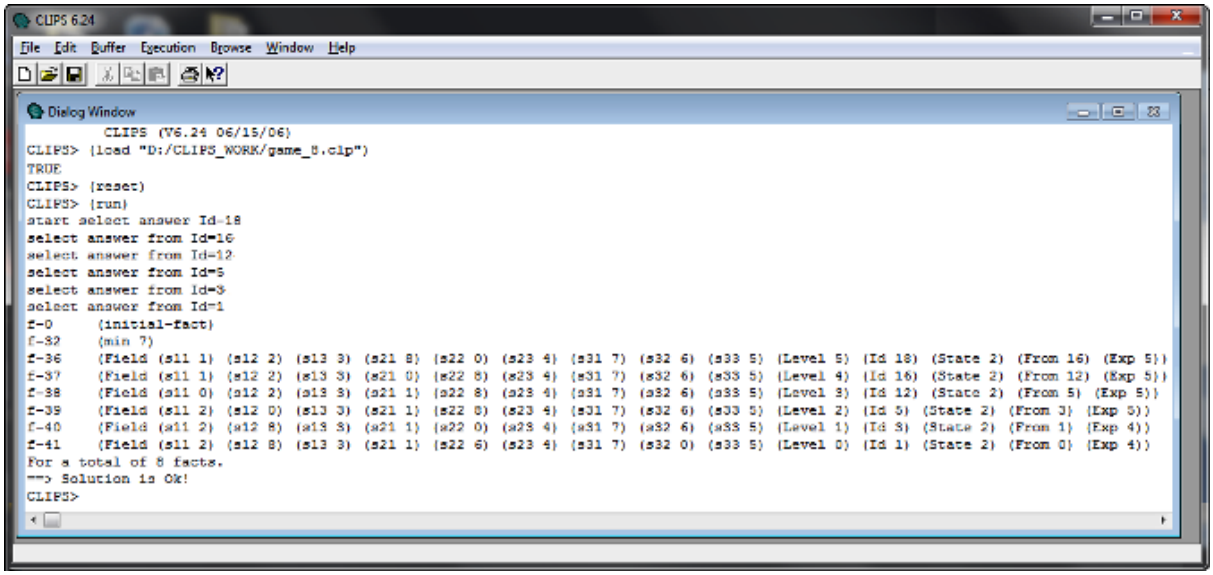


Рис. 2.4. Результат выполнения программы при ее загрузке в среду CLIPS.

Результат выполнения описанной выше программы при ее загрузке в среду CLIPS и запуску на выполнение может иметь вид, аналогичный приведенному на рис. 2.4.

2.3. Полный текст программы

Листинг программы поиска решения для игры в «Восемь» может иметь приводимый ниже вид [5]:

```
;; определение шаблона описания ситуации для квадрата 3x3
(deftemplate Field
   (slot s11   (type NUMBER))
   (slot s12   (type NUMBER))
   (slot s13   (type NUMBER))
   (slot s21   (type NUMBER))
   (slot s22   (type NUMBER))
   (slot s23   (type NUMBER))
   (slot s31   (type NUMBER))
   (slot s32   (type NUMBER))
   (slot s33   (type NUMBER))
   (slot Level(type NUMBER))              ; Уровень дерева
   (slot Id    (type NUMBER)(default 0))  ; Уникальный номер вершины
   (slot State(type NUMBER)(default 0))   ; Принадлежность списку узлов
   (slot From  (type NUMBER))             ; Номер вершины родителя
   (slot Exp   (type NUMBER))             ; Значение целевой функции
)

;; определение глобальной переменной
(defglobal
   ?*Id* = 0
)
```

```
;; функция вычисления уникального идентификатора текущего состояния
(deffunction Get_Id()
   (bind ?*Id* (+ ?*Id* 1))
   ?*Id*
)

;; целевая функция
(deffunction W(?Level ?s11 ?s12 ?s13 ?s23 ?s33 ?s32 ?s31 ?s21)
   (bind ?a ?Level)
   (if (not (= ?s11 1)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s12 2)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s13 3)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s23 4)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s33 5)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s32 6)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s31 7)) then (bind ?a (+ 1 ?a)) )
   (if (not (= ?s21 8)) then (bind ?a (+ 1 ?a)) )
   ?a
)
;; задание исходного состояния
(deffacts start
   (Field (s11 2) (s12 8) (s13 3)        ; Исходное положение
          (s21 1) (s22 6) (s23 4)        ; фишек на игровом
          (s31 7) (s32 0) (s33 5)        ; поле
          (Level 0)                      ; Это корень дерева
          (From 0)
          (Exp (W 0 2 8 3 4 5 0 7 1))    ; Значение оценочной функции
          (Id (Get_Id))                  ; Уникальный номер вершины
   )
   (min (W 0 2 8 3 4 5 0 7 1))
)

;; правило для исключения повторяющихся ситуаций
;; у этого правила самый высокий приоритет;
(defrule move_circle
   (declare (salience 1000))
   ?f1 <- (Field (State 1) (s11 ?LT1) (s12 ?MT1) (s13 ?RT1)
                           (s21 ?LM1) (s22 ?MM1) (s23 ?RM1)
                           (s31 ?LB1) (s32 ?MB1) (s33 ?RB1) )
   ?f0 <- (Field (State 0) (s11 ?LT0&:(= ?LT0 ?LT1))
                 (s12 ?MT0&:(= ?MT0 ?MT1)) (s13 ?RT0&:(= ?RT0 ?RT1))
                 (s21 ?LM0&:(= ?LM0 ?LM1)) (s22 ?MM0&:(= ?MM0 ?MM1))
                 (s23 ?RM0&:(= ?RM0 ?RM1)) (s31 ?LB0&:(= ?LB0 ?LB1))
                 (s32 ?MB0&:(= ?MB0 ?MB1)) (s33 ?RB0&:(= ?RB0 ?RB1)) )
   =>
   (retract ?f0)
   (printout t ?LT0 ?MT0 ?RT1 crlf ?LM0 ?MM0 ?RM0 crlf
               ?LB0 ?MB0 ?RB0 crlf )
)

;;--- далее определяются 9 правил формирования пространства состояний
;;--- они имеют одинаковый приоритет, что дает случайность применения
```

```
(defrule make_new_path_s11
   (declare (salience 100))
   ?fmin <- (min ?min)
   ?f <- (Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E& :(= ?E ?min))
                (s11 0)   (s12 ?MT) (s13 ?RT)
                (s21 ?LM) (s22 ?MM) (s23 ?RM)
                (s31 ?LB) (s32 ?MB) (s33 ?RB)   )
   =>
   (modify ?f(State 1))
   (bind ?a (W (+ 1 ?L) ?MT 0 ?RT ?RM ?RB ?MB ?LB ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?MT) (s12 0)   (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id)) (Exp ?a) )
   )
   (assert (Field (s11 ?LM) (s12 ?MT) (s13 ?RT)
                  (s21 0)   (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ 1 ?L) ?LM ?MT ?RT ?RM ?RB ?MB ?LB 0))   )
   )
)

(defrule make_new_path_s12
   (declare (salience 100))
   ?fmin <- (min ?min)
   ?f <-(Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E&:(= ?E ?min))
               (s11 ?LT) (s12 0)   (s13 ?RT)
               (s21 ?LM) (s22 ?MM) (s23 ?RM)
               (s31 ?LB) (s32 ?MB) (s33 ?RB)   )
   =>
   (modify ?f (State 1))
   (bind ?a (W (+ 1 ?L) 0 ?LT ?RT ?RM ?RB ?MB ?LB ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 0)   (s12 ?LT) (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id)) (Exp ?a) )
    )
   (assert (Field (s11 ?LT) (s12 ?MM) (s13 ?RT)
                  (s21 ?LM) (s22 0)   (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ 1 ?L) ?LT ?MM ?RT ?RM ?RB ?MB ?LB ?LM)) )
    )
   (assert (Field (s11 ?LT) (s12 ?RT) (s13 0)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ 1 ?L) ?LT ?RT 0 ?RM ?RB ?MB ?LB ?LM))   )
```

```
    )
)

(defrule make_new_path_s13
   (declare (salience 100))
   ?fmin <-(min ?min)
   ?f<-(Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E&:(= ?E ?min))
              (s11 ?LT) (s12 ?MT) (s13 0)
              (s21 ?LM) (s22 ?MM) (s23 ?RM)
              (s31 ?LB) (s32 ?MB) (s33 ?RB)  )
   =>
   (modify ?f(State 1))
   (bind ?a (W (+ 1 ?L) ?LT 0 ?MT ?RM ?RB ?MB ?LB ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?LT) (s12 0)   (s13 ?MT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id)) )
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RM)
                  (s21 ?LM) (s22 ?MM) (s23 0)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ 1 ?L) ?LT ?MT ?RM 0 ?RB ?MB ?LB ?LM))  )
    )
)

(defrule make_new_path_s21
   (declare (salience 100))
   ?fmin <-(min ?min)
   ?f <-(Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E& :(= ?E ?min))
               (s11 ?LT) (s12 ?MT) (s13 ?RT)
               (s21 0)   (s22 ?MM) (s23 ?RM)
               (s31 ?LB) (s32 ?MB) (s33 ?RB)   )
   =>
   (modify ?f (State 1))
   (bind ?a (W (+ 1 ?L) 0 ?MT ?RT ?RM ?RB ?MB ?LB ?LT))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 0)   (s12 ?MT) (s13 ?RT)
                  (s21 ?LT) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id))
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?MM) (s22 0)   (s23 ?RM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM ?RB ?MB ?LB ?MM)) )
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LB) (s22 ?MM) (s23 ?RM)
```

```
                (s31 0)   (s32 ?MB) (s33 ?RB)
                (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM ?RB ?MB 0 ?LB))   )
    )
)

(defrule make_new_path_s22
   (declare (salience 100))
   ?fmin <- (min ?min)
   ?f <- (Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E& :(= ?E ?min))
              (s11 ?LT) (s12 ?MT) (s13 ?RT)
              (s21 ?LM) (s22 0)   (s23 ?RM)
              (s31 ?LB) (s32 ?MB) (s33 ?RB)   )
   =>
   (modify ?f(State 1))
   (bind ?a (W (+ 1 ?L) ?LT 0 ?RT ?RM ?RB ?MB ?LB ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?LT) (s12 0)   (s13 ?RT)
                (s21 ?LM) (s22 ?MT) (s23 ?RM)
                (s31 ?LB) (s32 ?MB) (s33 ?RB)
                (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id)) )
   )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                (s21 ?LM) (s22 ?RM) (s23 0)
                (s31 ?LB) (s32 ?MB) (s33 ?RB)
                (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                (Exp (W (+ 1 ?L) ?LT ?MT ?RT 0 ?RM ?MB ?LB ?LM))  )
   )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                (s21 ?LM) (s22 ?MB) (s23 ?RM)
                (s31 ?LB) (s32 0)   (s33 ?RB)
                (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM ?RB 0 ?LB ?LM))  )
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                (s21 0)   (s22 ?LM) (s23 ?RM)
                (s31 ?LB) (s32 ?MB) (s33 ?RB)
                (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                (Exp (W (+ 1 ?L) ?LT ?MT ?RT ?RM ?RB ?MB ?LB 0))  )
    )
)

(defrule make_new_path_s23
   (declare (salience 100))
   ?fmin <- (min ?min)
   ?f <- (Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E& :(= ?E ?min))
              (s11 ?LT) (s12 ?MT) (s13 ?RT)
              (s21 ?LM) (s22 ?MM) (s23 0)
              (s31 ?LB) (s32 ?MB) (s33 ?RB)   )
   =>
   (modify ?f (State 1))
   (bind ?a (W (+ ?L 1) ?LT ?MT 0 ?RT ?RB ?MB ?LB ?LM))
```

```
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 0)
                  (s21 ?LM) (s22 ?MM) (s23 ?RT)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id)) )
   )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LM) (s22 0)   (s23 ?MM)
                  (s31 ?LB) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ ?L 1) ?LT ?MT ?RT ?MM ?RB ?MB ?LB ?LM)) )
   )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RB)
                  (s31 ?LB) (s32 ?MB) (s33 0)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RB 0 ?MB ?LB ?LM))   )
   )
)

(defrule make_new_path_s31
   (declare (salience 100))
   ?fmin<-(min ?min)
   ?f <-(Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E& :(= ?E ?min))
               (s11 ?LT) (s12 ?MT) (s13 ?RT)
               (s21 ?LM) (s22 ?MM) (s23 ?RM)
               (s31 0)   (s32 ?MB) (s33 ?RB)   )
   =>
   (modify ?f(State 1))
   (bind ?a (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB ?MB ?LM 0))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 0)   (s22 ?MM) (s23 ?RM)
                  (s31 ?LM) (s32 ?MB) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id)) )
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?MB) (s32 0) (s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB 0 ?MB ?LM))  )
    )
)

(defrule make_new_path_s32
   (declare (salience 100))
   ?fmin <-(min ?min)
   ?f <-(Field (State 0) (Id ?Id)  (Level ?L) (Exp ?E& :(= ?E ?min))
               (s11 ?LT) (s12 ?MT) (s13 ?RT)
               (s21 ?LM) (s22 ?MM) (s23 ?RM)
               (s31 ?LB) (s32 0)   (s33 ?RB)   )
```

```
   =>
   (modify ?f (State 1))
   (bind ?a (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB ?LB 0 ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?LT)(s12 ?MT)(s13 ?RT)
                  (s21 ?LM)(s22 ?MM)(s23 ?RM)
                  (s31 0)  (s32 ?LB)(s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id)) )
    )
   (assert (Field (s11 ?LT)(s12 ?MT)(s13 ?RT)
                  (s21 ?LM)(s22 0)  (s23 ?RM)
                  (s31 ?LB)(s32 ?MM)(s33 ?RB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM ?RB ?MM ?LB ?LM)) )
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 ?RB) (s33 0)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM 0 ?RB ?LB ?LM))  )
    )
)

(defrule make_new_path_s33
   (declare (salience 100))
   ?fmin <- (min ?min)
   ?f <- (Field (State 0) (Id  ?Id) (Level ?L) (Exp ?E& :(= ?E ?min))
                (s11 ?LT) (s12 ?MT) (s13 ?RT)
                (s21 ?LM) (s22 ?MM) (s23 ?RM)
                (s31 ?LB) (s32 ?MB) (s33 0)     )
   =>
   (modify ?f(State 1))
   (bind ?a (W (+ ?L 1) ?LT ?MT ?RT 0 ?RM ?MB ?LB ?LM))
   (retract ?fmin)
   (assert (min ?a))
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 0)
                  (s31 ?LB) (s32 ?MB) (s33 ?RM)
                  (Level (+ ?L 1)) (From ?Id) (Exp ?a) (Id (Get_Id)) )
    )
   (assert (Field (s11 ?LT) (s12 ?MT) (s13 ?RT)
                  (s21 ?LM) (s22 ?MM) (s23 ?RM)
                  (s31 ?LB) (s32 0)   (s33 ?MB)
                  (Level (+ ?L 1)) (From ?Id) (Id (Get_Id))
                  (Exp (W (+ ?L 1) ?LT ?MT ?RT ?RM 7MB 0 ?LB ?LM))  )
    )
) ;;--- конец определения 9 правил формирования пространства состояний

(defrule fmd_min
   (declare (salience 150))
   ; приоритет ниже чем у правила исключающего циклы
   ; и выше чем у правил порождения новых ходов
```

```
   ?fmin <- (min ?min)
   (Field (Exp ?E&:(< ?E ?min)) (State 0))
   =>
   (retract ?fmin)
   (assert (min ?E))
)
(defrule del_no_min_nodes
   (declare (salience 160))
   ?fmin <- (min ?min)
   ?node <- (Field (Exp ?E&:(> ?E ?min)) (State 0))
   =>
   (retract ?node)
   ;(printout t "del_no_min_nodes" ?node crlf)
)

;; если решение найдено, то выделяем его
(defrule start_select_answer
   (declare (salience 500))
   ?f <-(Field (s11 1) (s12 2) (s13 3)
               (s21 8) (s22 0) (s23 4)
               (s31 7) (s32 6) (s33 5)
               (State ~2) (From ?Id)   )
   =>
   (printout t "start select answer Id-" (fact-slot-value ?f Id) crlf)
   (modify ?f (State 2))
)

;; выделяем все вершины дерева решение, присваивая им State=2
(defrule select_answer
   (declare (salience 500))
   (Field (State 2) (From ?Id))
   ?f <- (Field (Id ?Id) (State ~2))
   =>
   (modify ?f (State 2))
   (printout t "select answer Id=" ?Id " - " ?f crlf)
)

;; удапяем остальные
(defrule delete_not_answer
   (declare (salience 400))
   (Field (State 2))
   ?f <- (Field (State ~2))
   =>
   (retract ?f)
   (printout t "delete not answer" crlf)
)

;; делаем останов если решений нет
(defrule Stop_l
   (declare (salience 200))
   (Field(State ?x))
   (not (Field(State 0|2)))
   =>
```

```
    (halt)
    (printout t "==> No Solutions" crlf)
)

;; делаем останов если решение есть
(defrule Stop_2
    (declare (salience 200))
    (Field(State 2))
    =>
    (facts)
    (halt)
    (printout t "==> Solution is Ok!" crlf)
)
```

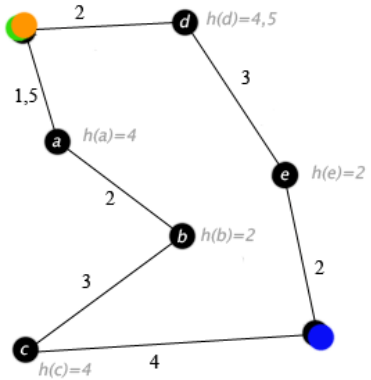
2.4. Варианты индивидуальных заданий

Задание:

Для задач из приведенных ниже вариантов придумать адекватную задаче оценочную функцию и составить программу, реализующую алгоритм A*. Причем программа должна показывать (в любом виде) дерево поиска с указанием значений узлов и их оценок.

1. Робот находится в произвольной точке склада (x_0,y_0). На площади склада имеется несколько групп стеллажей, которые заданы своими координатами. Найти кратчайший путь движения робота до выхода из склада, если известны его координаты (xk,yk).

2. Имеется n населенных пунктов, перенумерованных от 1 до n. Некоторые пары пунктов соединены дорогами. Определить, кратчайший путь из пункта x в пункт y ($1<x<n$, $1< y<n$). Пример простейшей тестовой схемы приведен на рисунке.



3. На плоской карте лесного участка, заданной в координатах (x, y), отмечен ряд непроходимых участков: озеро, болото, карьер. Спроектировать самую короткую дорогу из точки (0, 0) в некоторую произвольную точку с координатами (x_k,y_k).

4. Трем миссионерам и трем каннибалам необходимо переправиться на другой берег реки. На берегу реки находится лодка (без гребца), которая может вместить не более 2-х

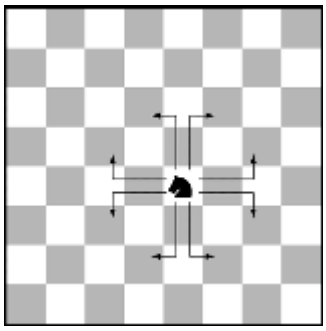
человек. Необходимо организовать переправу так, чтобы на одном берегу количество каннибалов не превышало количество миссионеров.

5. Три рыцаря, каждый в сопровождении оруженосца, съехались на берегу реки, намереваясь переправиться на другую сторону. Им удалось найти двухместную лодку и переправа произошла бы легко, если бы не одно затруднение: все оруженосцы, словно сговорившись, наотрез отказались оставаться в обществе незнакомых рыцарей без своих хозяев. Необходимо организовать переправу, соблюдая условия оруженосцев.

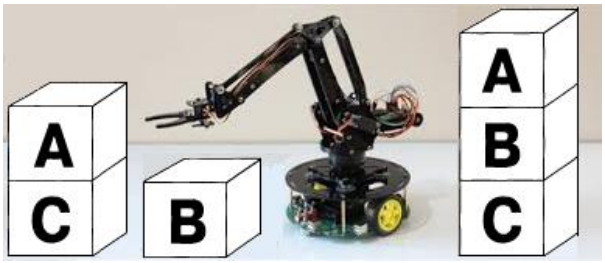
6. Разработать алгоритм решения старинной логической задачи: “Как с помощью пятилитрового бидона и трехлитровой банки набрать из родника 4 литра воды?”

7. Разработать алгоритм решения старинной логической задачи: “В бочке 12 л кваса. Как с помощью 5- и 7- литровых банок разделить квас на две равные части по 6 литров?”

8. Найти кратчайший путь перемещения шахматного коня из любой произвольной позиции на доске в любую другую, наперед заданную, позицию.

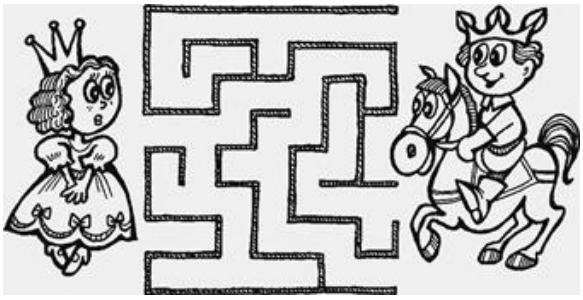


9. Робот обслуживает три площадки. На одной из них производится сборка готового изделия, на две другие поступают блоки комплектующих изделия трех типов (*A,B,C*). На каждой из двух площадок к началу сборки может находиться от 0 до 3 блоков в любых сочетаниях. Разработать алгоритм оптимального перемещения блоков роботом при сборке готового изделия.



10. Разработать алгоритм решения старинной логической задачи: «Как трем супружеским парам переправиться через реку в двухместной лодке, если правила того времени не позволяли замужней женщине находиться в обществе мужчин без своего мужа?”

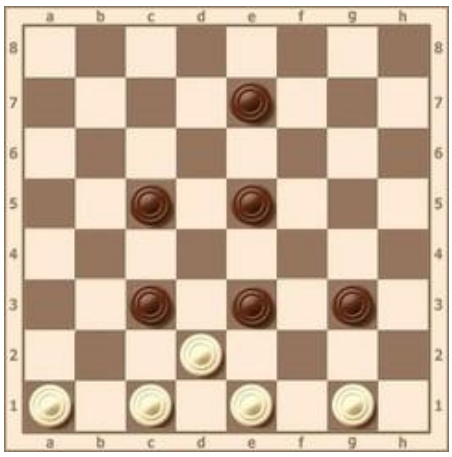
11. Разработать алгоритм определения кратчайшего пути движения через лабиринт произвольной формы размера 8x8. Координаты точек входа и выхода заданы.



12. Разработать алгоритм решения старинной логической задачи: “Как при помощи 5-ти литрового и 9-ти литрового ведра набрать из реки 3 литра воды?”.

13. На плоской карте указано n населенных пунктов, заданных своими координатами (x_i,y_i). Также определен ряд основных дорог, соединяющих некоторые пары населенных пунктов. Определить, кратчайший путь из некоторого начального пункта (x_0,y_0) в любой заданный конечный пункт (x_k,y_k).

14. Найти лучший ход белых в данной позиции шашечной партии. Исследовать работоспособность программы в аналогичной ситуации при ином расположении шашек.



15. На игровом поле множество холмов и ям. По нему гуляет «бык» и вдалеке видит «корову». Холмы обозначены красным, ямы черным. Изначально у быка - 5 жизней. Если бык залезает на холм - он теряет одну жизнь. Если падает в яму - теряет две жизни. Задача проста - бык должен дойти до коровы, не умерев. Количество жизней и их потери можно изменять при небольших размерах поля.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Рассел Стюарт, Норвиг Питер.* Искусственный интеллект: современный подход, 2-е издание: Пер с англ. - М.: Изд. дом "Вильяме", 2007, -1408 с.
2. *Люгер Дж.* Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание: Пер. с англ. — М.: Изд. дом "Вильямс", 2003,- 864 с.
3. *Джарратано Джозеф, Райли Гари.* Экспертные системы: принципы разработки и программирования, 4-е издание: Пер. с англ. — М. : ООО ”И.Д. Вильямс", 2007. — 1152 с. :
4. *Частиков А., Гаврилова Т., Белов Д.* Разработка экспертных систем. Среда CLIPS. Серия: Учебное пособие. — СПб.: БХВ-Петербург, 2003. — 393 с.:
5. *Вахтин А.А., Гаршина В.В.* Лабораторный практикум по программированию на языке CLIPS для курса “Представление знаний в информационных системах”. Учебное-методическое пособие. — Издание Воронежского ГУ, 2010. – 96 с.
6. CLIPS - примеры программ. — http://fkn.ktu10.com/?q=node/2183
7. *Белкин Е.В.* Практика применения языка CLIPS для построения экспертных систем. — http://www-csd.univer.kharkov.ua/content/files/cat16/6_primenenie_clips.pdf
8. CLIPS - Википедия — http://ru.wikipedia.org/wiki/CLIPS
9. Практическая разработка экспертных систем в среде CLIPS — http://inter-vuz. tuit.uz /Elib_baza/Kafedra/IT_adab/isrob/7/1.html
10. Программирование на языке CLIPS — http://ryk-kypc2.narod.ru/clips.htm
11. Русификация языка построения экспертных систем CLIPS — http://www.opennet.ru/ tips/info/2420.shtml
12. CLIPS Documentation — http://clipsrules.sourceforge.net/OnlineDocs.html
13. CLIPS Home Page — http://clipsrules.sourceforge.net/index.html
14. Поиск пути, алгоритмы поиска в глубину и ширину — НОУ «ИНТУИТ», http://www. youtube.com/watch?v=sNmW9nl-qzQ
15. Алгоритм обхода графа в глубину — НОУ ИНТУИТ, http://www.youtube.com/watch?v= Y5qaZL5hV98
16. Поиск в графах и обход. Алгоритм Дейкстры — НОУ ИНТУИТ, http://www. youtube. com/watch?v=IjLHY5U4y2c
17. Поиск в ширину — НОУ ИНТУИТ, http://www.youtube.com/watch?v=hErsJuHRHBo
18. Интеллектуальные робототехнические системы НОУ ИНТУИТ, http://www.intuit.ru/ studies/courses/13079/46/info