Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных технология и систем»

Лабораторная работа №7

**ИСПОЛЬЗОВАНИЕ ОТСЕЧЕНИЙ**

по дисциплине:

«Функциональное и логическое программирование»

**Отчёт**

Принял преподаватель:

\_\_\_\_\_\_\_\_\_\_ / Михайлов Д.В

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_\_\_\_ / Лехновский А. Д.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Великий Новгород**

**2021**

1. **Цель и задачи данной лабораторной работы**

Целью лабораторной работы является изучение правил использования отсечения в двух случаях: для подтверждения правильности выбранного решения и для прекращения процесса порождения и проверки возможных решений.

1. **Задания на лабораторную работу**

**Задача №1:**

Создать меню, обеспечить вызов и выполнение всех заданий из Лабораторной работы №6. Каждое задание должно выполняться в отдельном окне.

1. Написать программу сортировки списка методом Шелла. Вычисление последовательности шагов сортировки производится методом, предложенным Дональдом Кнутом.
2. Написать программу сортировки списка методом пузырька.
3. Написать программу решения головоломки. «Числа по периметру». По периметру пятиконечной звезды необходимо поставить числа от 1 до 10 так, чтобы суммы чисел в концах любого отрезка не делились ни на 3, ни на 5, ни на 7.
4. Написать программу вставки в список нового элемента на все места, где i задается в качестве аргумента, а n = 1, 2, 3, . .

**Задача №2:**

Написать для игры «Крестики-нолики» процедуру «Следующий ход», которая для заданного положения на доске находила бы наилучший ход, обеспечивающий либо предотвращение проигрыша, либо выигрыш, либо наилучший прогнозируемый результат.

**Задача №3:**

Написать программу, реализующую выигрышную стратегию для игры «Крестики-нолики» на доске 3 × 3. Игровое поле и весь процесс игры должен отображаться на экране в графическом режиме.

1. **Решения поставленных задач**

Для выполнения поставленной задачи был использован Prolog –SWI-Prolog.

**Задача №1:**

Для поставленной задачи было разработано главное окно «Меню» (см. рис.1) с 4 кнопками, которые ведут к другим окнам, которые решают задачи из предыдущей лабораторной работы:

«Сортировка Шелла» (shell\_win()) - сортировка списка методом Шелла, где вычисление последовательности шагов сортировки производится методом, предложенным Дональдом Кнутом;

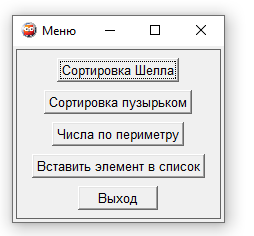
«Сортировка пузырьком» (bubble\_win()) - сортировка методом пузырька;

«Числа по периметру» (number\_result()) - решение головоломки «Числа по периметру»;

«Вставить элемент в список» (insert\_win()) - вставка в список нового элемента на все места, где i задается в качестве аргумента, а n = 1, 2, 3, . .

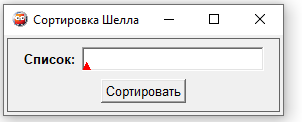
Данная программы запускается вызовом **menu().**

**Примеры работы программы**

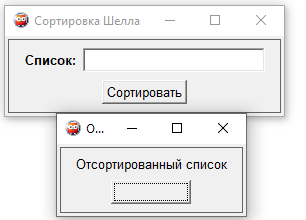


*Рисунок №1. Окно «Меню».*

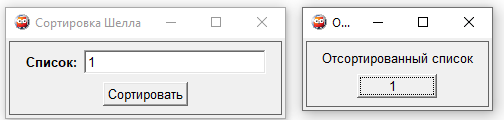
**Примеры работы сортировка Шелла:**



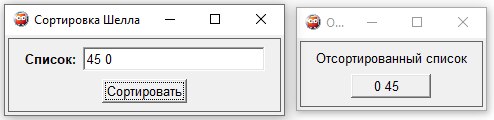
*Рисунок №2. Окно «Сортировка Шелла», ввод пользователем списка для сортировки.*



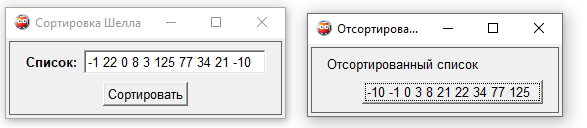
*Рисунок №3. Окно «Сортировка Шелла», если на входе пустой список.*



*Рисунок №4. Окно «Сортировка Шелла», если на входе список из 1 элемента.*

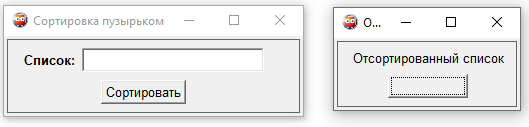


*Рисунок №5. Окно «Сортировка Шелла», если на входе список из 2 элементов.*

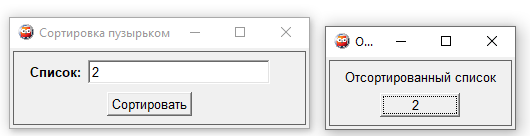


*Рисунок №6. Окно «Сортировка Шелла», пример работы сортировки Шелла.*

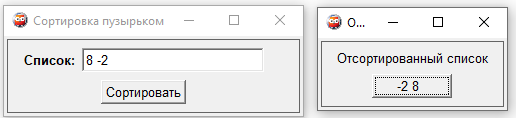
**Примеры работы сортировка методом пузырька:**



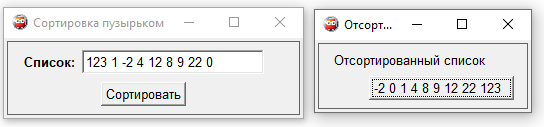
*Рисунок №7. Окно «Сортировка пузырьком», если на входе пустой список.*



*Рисунок №8. Окно «Сортировка пузырьком», если на входе список из 1 элемента.*



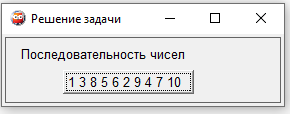
*Рисунок №9. Окно «Сортировка пузырьком», если на входе список из 2 элементов.*



*Рисунок №10. Окно «Сортировка пузырьком», пример работы сортировки пузырьком.*

**Пример работы «Числа по периметру»:**

Для этого задания было сделано окно, которое сразу выводит результат работы программы, так как здесь не требуется дополнительный ввод данных.



*Рисунок №11. Окно «Числа по периметру», результат работы программы.*

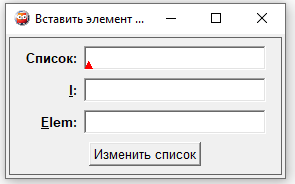
**Пример работы «Вставить элемент в список»:**

Для этого задания было сделано специально окно, где мы будем вводит дополнительные данные, а именно:

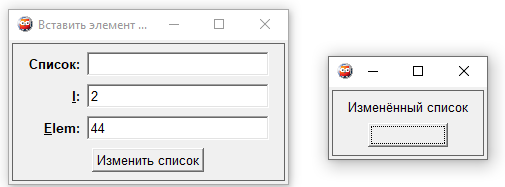
Список – это мы вводим через пробел элементы исходного списка, с которым в дальнейшим будем работать;

I – аргумент «i», который нам нужен для вставки на места;

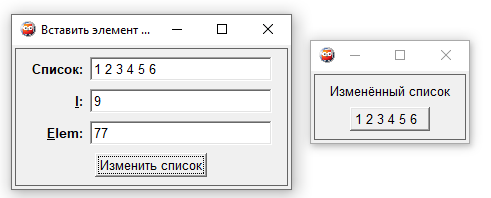
**Elem** – это вставка или элемент, который мы хотим вставить в список;



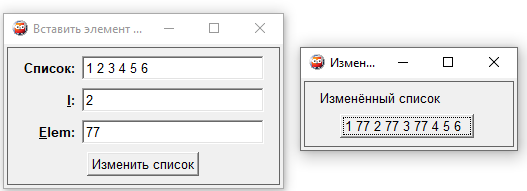
*Рисунок №12. Окно «Вставить элемент в список», ввод дополнительных данных.*



*Рисунок №13. Окно «Вставить элемент в список», если на входе пустой список, то добавлять ничего не надо.*



*Рисунок №14. Окно «Вставить элемент в список», если длина списка меньше, чем указанный I, то добавлять ничего не нужно*



*Рисунок №15. Окно «Вставить элемент в список», вставка в список элементов, находящихся на (i \* n) – х местах.*

Код программы для решения задачи приведен в *Приложении 1*.

В коде программы прокомментированы конкретные случаи использования отсечений. В большой степени использовались отсечения, которые указывают Прологу на то, что найдено нужное правило для заданного целевого утверждения.

**Задача №2:**

Для вывода состояния игры используется предикат **show()**.

Первыми ходят крестики (игрок).

Обозначения: **1** – крестик, **2** – нолик, **0** – не занятая позиция.

Для ходов ноликов были написаны правила **next\_move\_o()**, которые для заданного положения на доске находила бы наилучший ход, обеспечивающий либо предотвращение проигрыша, либо выигрыш, либо наилучший прогнозируемый результат. В правилах **next\_move\_o()** использованы отсечения, так как правила проверяются сверху вниз, а сверху расположены наиболее ценные ходы:

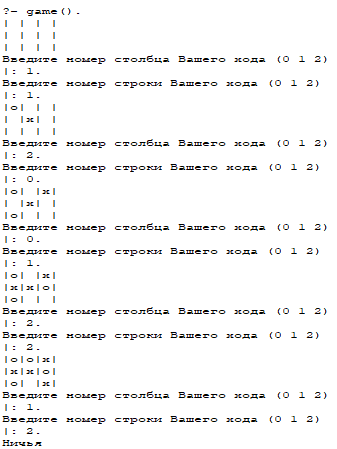
1. Если игрок не занял центральную клетку, то ставим туда нолик;
2. Если есть возможность победить, то выбираем выигрышную позицию;
3. Если у крестиков выигрышная ситуация, то предотвращаем её;
4. Если центральная ячейка занята крестиком, то выбираем левую верхнюю ячейку;
5. Если игрок занял угловую позицию, то занимаем угловую позицию на противоположной стороне;
6. В противном случае занимаем угловые позиции, если они заняты, любую свободную ячейку.

А отсечения в этих правилах задействованы для поиска первого подходящего по условиям правила.

Для запуска программы нужно ввести: **game().**

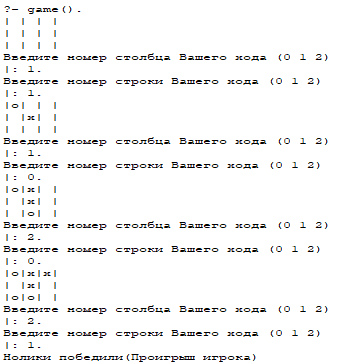
**Пример работы программы:**

**Ничья:**



*Рисунок №16. Пример работы программы «Крестики-нолики», где исход игры Ничья.*

**Проигрыш игрока:**

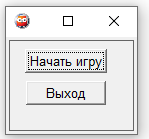


*Рисунок №17. Пример работы программы «Крестики-нолики», где исход игры Проигрыш.*

Код программы для решения задачи приведен в *Приложении 2*.

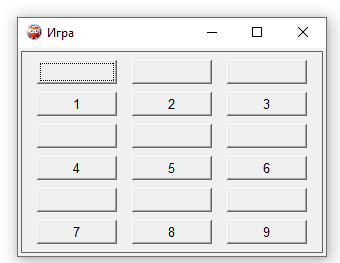
**Задача №3:**

Для игры «Крестики-нолики» было создано окно «Меню» (см. рис. 18), которое вызывается командой **game\_menu()**.



*Рисунок №18. Пример работы программы «Крестики-нолики», окно «Меню».*

Само окно игры «Крестики-нолики» (см. рис. 19) можно вызвать нажав кнопку «Начать игру» или просто вызвать командой **game().**

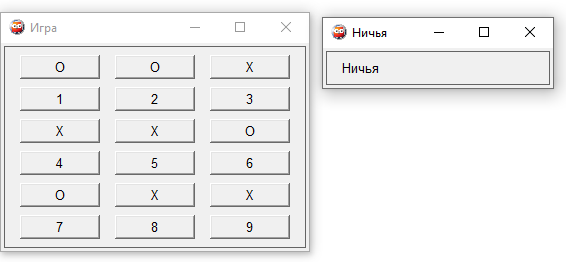


*Рисунок №19. Пример работы программы «Крестики-нолики», окно «Игра».*

Игрок ходит крестиками. Нажимать следует на кнопки с цифрами, тогда на поле выше кнопки отобразится «Х».

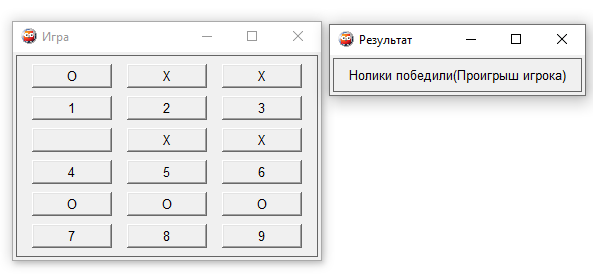
**Пример работы программы:**

**Ничья:**



*Рисунок №20. Пример работы программы «Крестики-нолики», результат ничья.*

**Проигрыш игрока:**



*Рисунок №21. Пример работы программы «Крестики-нолики», результат проигрыш.*

Отсечение используется для поиска первого правила подходящего по условиям.

Код программы для решения задачи приведен в *Приложении 3*.

1. **Вывод**

В данной лабораторной работе, я изучил правила отсечения в двух случаях: для подтверждения правильности выбранного решения и для прекращения процесса порождения и проверки возможных решений.

**Приложение 1**

Код программы:

:- use\_module(library(pce)).

% Запуск окна меню

menu():-

new(Main, dialog('Меню')),

new(Shell, button('Сортировка Шелла', message(@prolog, shell\_win))),

new(Bubble, button('Сортировка пузырьком', message(@prolog, bubble\_win))),

new(Number, button('Числа по периметру', message(@prolog, number\_result))),

new(Inser, button('Вставить элемент в список', message(@prolog, insert\_win))),

new(Exit, button('Выход', message(Main, destroy))),

send(Main, append, Shell),

send(Main, append, Bubble, below),

send(Main, append, Number, below),

send(Main, append, Inser, below),

send(Main, append, Exit, below),

send(Main, open).

% Запуск окна сортировки Шелла

shell\_win() :-

new(Shell, dialog('Сортировка Шелла')),

new(List, text\_item('Список')),

new(ButtonSort,button('Сортировать', message(@prolog, shell\_result, List))),

send(Shell, append, List),

send(Shell, append, ButtonSort),

send(Shell, open).

%Сортировка введенного списка и вывод результата в новое окно.

shell\_result(L) :-

get(L, value, L1),

new(Dialog, dialog("Отсортированный список")),

new(F, text("Отсортированный список")),

new(B, button(" ")),

send(Dialog, append, F),

send(Dialog, append, B),

send(Dialog, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

shell\_sort(ListToSort, Result),

p(Result, ResultStr),

send(B, name, ResultStr));

(send(B, name, " "))).

% Запуск окна сортировки пузырьком

bubble\_win() :-

new(Bubble, dialog('Сортировка пузырьком')),

new(List1, text\_item('Список')),

new(ButtonSort1,button('Сортировать', message(@prolog, bubble\_result, List1))),

send(Bubble, append, List1),

send(Bubble, append, ButtonSort1),

send(Bubble, open).

%Сортировка введенного списка и вывод результата в новое окно.

bubble\_result(L) :-

get(L, value, L1),

new(Dialog1, dialog("Отсортированный список")),

new(F1, text("Отсортированный список")),

new(B1, button(" ")),

send(Dialog1, append, F1),

send(Dialog1, append, B1),

send(Dialog1, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToSort),

bubble\_sort(ListToSort, Result),

p(Result, ResultStr),

send(B1, name, ResultStr));

(send(B1, name, " "))).

% Запуск окна-результа для решения задачи №3

number\_result() :-

answer(Result),

new(Dialog2, dialog("Решение задачи")),

new(F2, text("Последовательность чисел")),

new(B2, button(" ")),

send(Dialog2, append, F2),

send(Dialog2, append, B2),

send(Dialog2, open),

p(Result, ResultStr),

send(B2, name, ResultStr).

% Запуск окна "Вставка элемента в список на заданные позиции"

insert\_win() :-

new(Rever, dialog('Вставить элемент в список')),

new(List2, text\_item('Список')),

new(Dig, text\_item('I')),

new(Dig2, text\_item('Elem')),

new(ButtonAnswer,button('Изменить список', message(@prolog, insert\_result, List2, Dig, Dig2))),

send(Rever, append, List2),

send(Rever, append, Dig),

send(Rever, append, Dig2),

send(Rever, append, ButtonAnswer),

send(Rever, open).

%Запуск окна-результата задания 4.

insert\_result(L, I, Element) :-

get(L, value, L1),

new(Dialog3, dialog("Изменённый список")),

new(F3, text("Изменённый список")),

new(B3, button(" ")),

send(Dialog3, append, F3),

send(Dialog3, append, B3),

send(Dialog3, open),

(L1 \= '',

(split\_string(L1, " ", "", LTemp),

convert(LTemp, ListToWork),

get(I, value, I1),

atom\_number(I1, IRes),

get(Element, value, Element1),

atom\_number(Element1, ElementRes),

ins(ListToWork, IRes, ElementRes, Result),

p(Result, ResultStr),

send(B3, name, ResultStr));

(send(B3, name, " "))).

%Сортировка Шелла

%Если на входе пустой список

shell\_sort([], []).

%Если на входе список из 1 элемента

shell\_sort([H], [H]).

%Если на входе список из 2 элементов,где второй элемент < первого

shell\_sort([A, B|TAIL], [B, A]) :-

length\_list([A, B|TAIL], Length),

Length = 2,

A > B.

%Если на входе список из 2 элементов, первый элемент < второго

shell\_sort([A, B|T], [A, B]) :-

length\_list([A, B|T], Length),

Length = 2,

A < B.

%Запуск сортировки Шелла.

shell\_sort(List, Result) :-

length\_list(List, Length),

list\_steps(Length, Steps),

shell\_sort\_steps(List, Steps, Result).

%Проход по шагам сортировки

shell\_sort\_steps(Result, [], Result).

shell\_sort\_steps(List, [H|T], Result) :-

shell\_sort\_step(List, H, TempList),

shell\_sort\_steps(TempList, T, Result).

%Шаг сортировки

shell\_sort\_step(List, Step, Result) :-

shell\_sort\_step\_value(List, Step, 0, Step, Result).

shell\_sort\_step\_value(Result, Step, N1, \_, Result) :-

length\_list(Result, Length),

Max is (N1 + Step),

Max >= Length.

shell\_sort\_step\_value(List, Step, N1, N2, Result) :-

length\_list(List, Length),

N2 >= Length,

N3 is (N1 + 1),

N4 is (N3 + 1),

shell\_sort\_step\_value(List, Step, N3, N4, Result).

shell\_sort\_step\_value(List, Step, N1, N2, Result) :-

get\_element\_list(List, N1, Element1),

get\_element\_list(List, N2, Element2),

Element1 > Element2,

swap\_elem(List, N1, N2, NewList),

N3 is (N2 + Step),

shell\_sort\_step\_value(NewList, Step, N1, N3, Result).

shell\_sort\_step\_value(List, Step, N1, N2, Result) :-

get\_element\_list(List, N1, Element1),

get\_element\_list(List, N2, Element2),

Element1 =< Element2,

N3 is (N2 + Step),

shell\_sort\_step\_value(List, Step, N1, N3, Result).

%Получение последовательности шагов

list\_steps(Length, Steps) :-

number\_steps(Length, 2, NumberSteps),

list\_steps\_temp(NumberSteps, Steps).

list\_steps\_temp(NumberSteps, Steps) :-

NumberSteps > -1, !, %Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

step\_value(NumberSteps, Value),

NewNumberSteps is (NumberSteps - 1),

list\_steps\_temp(NewNumberSteps, StepsTail),

Steps = [Value|StepsTail];

Steps = [].

%Количество шагов сортировки

number\_steps(Length, Start, Steps) :-

Steps is Start - 2,

Length < 3 ^ Start;

NextStart is Start + 1,

number\_steps(Length, NextStart, Steps).

%Значение шагов сортировки

step\_value(0, 1).

step\_value(Step, Value) :-

Step1 is Step - 1,

step\_value(Step1, Value1),

Value is 3 \* Value1 + 1.

%Вычисление длины списка

length\_list(List, Length):-

length\_list\_temp(List, Length, 0).

length\_list\_temp([], Length, Length).

length\_list\_temp([\_|T], Length, Length1) :-

Length2 is (Length1 + 1),

length\_list\_temp(T, Length, Length2).

%Получить значение элемента по индексу

get\_element\_list([H|\_], 0, H).

get\_element\_list([\_|[H1|T]], N, Element) :-

N1 is (N - 1),

get\_element\_list([H1|T], N1, Element).

%Перестановка элементов

swap\_elem(List, N1, N2, NewList) :-

get\_element\_list(List, N1, Element1),

get\_element\_list(List, N2, Element2),

delete\_element\_list(List, N1, List1),

insert\_element\_list(List1, Element2, N1, List2),

delete\_element\_list(List2, N2, List3),

insert\_element\_list(List3, Element1, N2, NewList).

%Вставить элемент в список по индексу

insert\_element\_list([], Element, 0, [Element]).

insert\_element\_list([H|T], Element, 0, [Element|[H|T]]).

insert\_element\_list([H|T], Element, N, [H|T1]) :-

N1 is (N - 1),

insert\_element\_list(T, Element, N1, T1).

%Удалить элемент из списка по индексу

delete\_element\_list([\_|T], 0, T).

delete\_element\_list([H|T], N, [H|T1]) :-

N1 is (N - 1),

delete\_element\_list(T, N1, T1).

p([]," ").

p([H|T],S):-

p(T,SS),

concat(" ",SS,SSS),

concat(H,SSS,S).

convert([], []).

convert([H|T], [Elm|L]) :-

atom\_number(H, Elm),

convert(T,L).

%Сортировка пузырьком

%Если список пуст

bubble\_sort([], []).

%Если список состоит из 1-го элемента

bubble\_sort([H], [H]).

%Запуск сортировки методом пузырька

bubble\_sort(SortList, SortList):-

move\_max\_to\_end(SortList, DoubleSortList),

SortList = DoubleSortList, !. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

bubble\_sort(List, SortList):-

move\_max\_to\_end(List, SortPart),

bubble\_sort(SortPart, SortList).

move\_max\_to\_end([], []):-!. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

move\_max\_to\_end([Head], [Head]):-!. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

move\_max\_to\_end([First, Second|Tail], [Second|ListWithMaxEnd]):-

First > Second, !, % Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

move\_max\_to\_end([First|Tail], ListWithMaxEnd).

move\_max\_to\_end([First, Second|Tail], [First|ListWithMaxEnd]):-

move\_max\_to\_end([Second|Tail], ListWithMaxEnd).

answer([HeadAnswer|TailAnswer]) :-

list\_number\_1\_10(L),

permutation(L, [HeadAnswer|TailAnswer]),

check\_answer([HeadAnswer|TailAnswer], HeadAnswer).

%Список чисел от 1 до 10

list\_number\_1\_10([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).

%Проверка на то, что суммы соседних чисел не делятся ни на 3, ни на 5, ни на 7.

check\_answer([A], B):-

SumAB is A + B,

X is SumAB mod 3, X > 0,

Y is SumAB mod 5, Y > 0,

Z is SumAB mod 7, Z > 0.

check\_answer([A, B|Tail], К):-

SumAB is A + B,

X is SumAB mod 3, X > 0,

Y is SumAB mod 5, Y > 0,

Z is SumAB mod 7, Z > 0,

check\_answer([B|Tail], К).

%Если на входе пустой список, то добавлять ничего не надо.

ins([], \_, \_, []):- !.%Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Если длина списка меньше, чем указанный I, то добавлять ничего не нужно

ins(List, I, \_, List) :-

length(List, Length),

Length < I.

/\*Вставка в список элементов, находящихся на (i \* n) – х местах, где i задается в качестве аргумента, а в n = 1, 2, 3, … элемента Element\*/

ins(List, I, Element, Result) :-

length(List, Length),

get\_number\_insert(Length, I, 1, ListInsert),

insert\_list\_elem(List, ListInsert, Element, Result).

%Получение списка индексов для вставки элемента

get\_number\_insert(Length, I, N, ListInsert) :-

Length >= (I \* N), !, % Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

Value is (I \* N - 1),

NewN is (N + 1),

get\_number\_insert(Length, I, NewN, ListInsertTail),

ListInsert = [Value|ListInsertTail];

ListInsert = [].

%Вставить элементы по списку индексов

insert\_list\_elem(Result, [], \_, Result).

insert\_list\_elem(List, [H|T], Element, Result) :-

insert\_elem\_list(List, Element, H, TempList),

insert\_list\_elem(TempList, T, Element, Result).

insert\_elem\_list([], Elem, 0, [Elem]).

insert\_elem\_list([H|T], Elem, 0, [Elem|[H|T]]).

insert\_elem\_list([H|T], Elem, N, [H|T1]) :-

N1 is (N - 1),

insert\_elem\_list(T, Elem, N1, T1).

**Приложение 2**

Код программы:

game():-

initial\_state(State),

next\_move(State, 0).

%Вывод поля игры

show(State) :-

get\_element(State, 0, SymbolA),

get\_element(State, 1, SymbolB),

get\_element(State, 2, SymbolC),

get\_element(State, 3, SymbolD),

get\_element(State, 4, SymbolE),

get\_element(State, 5, SymbolF),

get\_element(State, 6, SymbolG),

get\_element(State, 7, SymbolH),

get\_element(State, 8, SymbolI),

symbol\_to\_number(SymbolA, NumberA),

symbol\_to\_number(SymbolB, NumberB),

symbol\_to\_number(SymbolC, NumberC),

symbol\_to\_number(SymbolD, NumberD),

symbol\_to\_number(SymbolE, NumberE),

symbol\_to\_number(SymbolF, NumberF),

symbol\_to\_number(SymbolG, NumberG),

symbol\_to\_number(SymbolH, NumberH),

symbol\_to\_number(SymbolI, NumberI),

write("|"),

write(NumberA),

write("|"),

write(NumberB),

write("|"),

write(NumberC),

writeln("|"),

write("|"),

write(NumberD),

write("|"),

write(NumberE),

write("|"),

write(NumberF),

writeln("|"),

write("|"),

write(NumberG),

write("|"),

write(NumberH),

write("|"),

write(NumberI),

writeln("|").

%Начальное состояние

initial\_state([0,0,0,0,0,0,0,0,0]).

%0 - Незанятое поле

symbol\_to\_number(0, ' ').

%1 - крестик

symbol\_to\_number(1, 'x').

%2 - нолик

symbol\_to\_number(2, 'o').

%Проверка того, что данное поле свободно

is\_free(X, Y, State) :-

N is (Y \* 3 + X),

get\_element(State, N, A),

A = 0.

%Выполнение хода

make\_move(X, Y, State, Symbol, State\_next) :-

is\_free(X, Y, State),

N is (Y \* 3 + X),

delete\_element(State, N, State1),

insert\_element(State1, Symbol, N, State\_next).

%Проверка того, что победили крестики

check\_state(State) :-

state\_win\_x(StateX),

member(State, [StateX]),

writeln("Крестики победили(Победа игрока)"), !. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Проверка того, что победили нолики

check\_state(State) :-

state\_win\_o(StateO),

member(State, [StateO]),

writeln("Нолики победили(Проигрыш игрока)"), !. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Проверка того, что игра закончилачь ничьей

check\_state(State) :-

not(member(0, State)),

writeln("Ничья"), !. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Ход игрока

next\_move(State, N) :-

I is (N mod 2), I = 0,

Symbol is 1,

show(State),

writeln("Введите номер столбца Вашего хода (0 1 2)"),

read(X),

writeln("Введите номер строки Вашего хода (0 1 2)"),

read(Y),

make\_move(X, Y, State, Symbol, NewState), !, % Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

not(check\_state(NewState)),

N1 is (N + 1),

next\_move(NewState, N1).

%Следующий ход.

next\_move(State, N) :-

I is (N mod 2), I = 1,

Symbol is 2,

next\_move\_o(State, X, Y),

make\_move(X, Y, State, Symbol, NewState), !, % Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

not(check\_state(NewState)),

N1 is (N + 1),

next\_move(NewState, N1).

%Варианты ситуаций игры, когда побеждают крестики (Победа игрока)

state\_win\_x([1,1,1,\_,\_,\_,\_,\_,\_]).

state\_win\_x([\_,\_,\_,1,1,1,\_,\_,\_]).

state\_win\_x([\_,\_,\_,\_,\_,\_,1,1,1]).

state\_win\_x([1,\_,\_,1,\_,\_,1,\_,\_]).

state\_win\_x([\_,1,\_,\_,1,\_,\_,1,\_]).

state\_win\_x([\_,\_,1,\_,\_,1,\_,\_,1]).

state\_win\_x([1,\_,\_,\_,1,\_,\_,\_,1]).

state\_win\_x([\_,\_,1,\_,1,\_,1,\_,\_]).

%Варианты ситуаций игры, когда побеждают нолики (Проигрыш игрока)

state\_win\_o([2,2,2,\_,\_,\_,\_,\_,\_]).

state\_win\_o([\_,\_,\_,2,2,2,\_,\_,\_]).

state\_win\_o([\_,\_,\_,\_,\_,\_,2,2,2]).

state\_win\_o([2,\_,\_,2,\_,\_,2,\_,\_]).

state\_win\_o([\_,2,\_,\_,2,\_,\_,2,\_]).

state\_win\_o([\_,\_,2,\_,\_,2,\_,\_,2]).

state\_win\_o([2,\_,\_,\_,2,\_,\_,\_,2]).

state\_win\_o([\_,\_,2,\_,2,\_,2,\_,\_]).

%Получение элемента по индексу

get\_element([Head|\_], 0, Head).

get\_element([\_|[Head1|Tail]], N, Element) :-

N1 is (N - 1),

get\_element([Head1|Tail], N1, Element).

%Вставка элемента по индексу

insert\_element([], Element, 0, [Element]).

insert\_element([Head|Tail], Element, 0, [Element|[Head|Tail]]).

insert\_element([Head|Tail], Element, N, [Head|Tail1]) :-

N1 is (N-1),

insert\_element(Tail, Element, N1, Tail1) .

%Удаление элемента по индексу

delete\_element([\_|Tail], 0, Tail).

delete\_element([Head|Tail], N, [Head|Tail1]) :-

N1 is (N - 1),

delete\_element(Tail, N1, Tail1).

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%Все следующие отсечения используются для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%Если центр не занят, то занимаем его. (Уровень важности ход: 1) Чем ниже уровень, тем он важнее

next\_move\_o([\_, \_,\_, \_, 0, \_, \_, \_, \_], 1, 1) :- !.

%Возможные ситуации, при которых, данный ход является выигрышным,тогда занимаем победную позицию. (Уровень важности ход: 2)

next\_move\_o([0,2,2,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

next\_move\_o([2,0,2,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

next\_move\_o([2,2,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,\_,0,2,2,\_,\_,\_], 0, 1) :- !.

next\_move\_o([\_,\_,\_,2,0,2,\_,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,\_,2,2,0,\_,\_,\_], 2, 1) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,0,2,2], 0, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,2,0,2], 1, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,2,2,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,2,\_,\_,2,\_,\_], 0, 0) :- !.

next\_move\_o([2,\_,\_,0,\_,\_,2,\_,\_], 0, 1) :- !.

next\_move\_o([2,\_,\_,2,\_,\_,0,\_,\_], 0, 2) :- !.

next\_move\_o([\_,0,\_,\_,2,\_,\_,2,\_], 1, 0) :- !.

next\_move\_o([\_,2,\_,\_,0,\_,\_,2,\_], 1, 1) :- !.

next\_move\_o([\_,2,\_,\_,2,\_,\_,0,\_], 1, 2) :- !.

next\_move\_o([\_,\_,0,\_,\_,2,\_,\_,2], 2, 0) :- !.

next\_move\_o([\_,\_,2,\_,\_,0,\_,\_,2], 2, 1) :- !.

next\_move\_o([\_,\_,2,\_,\_,2,\_,\_,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,\_,2,\_,\_,\_,2], 0, 0) :- !.

next\_move\_o([2,\_,\_,\_,0,\_,\_,\_,2], 1, 1) :- !.

next\_move\_o([2,\_,\_,\_,2,\_,\_,\_,0], 2, 2) :- !.

next\_move\_o([\_,\_,0,\_,2,\_,2,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,2,\_,0,\_,2,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,2,\_,2,\_,0,\_,\_], 0, 2) :- !.

%Возможные ситуации, в которой нужно предотвратить выигрыш противника (Уровень важности ход: 3)

next\_move\_o([0,1,1,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

next\_move\_o([1,0,1,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

next\_move\_o([1,1,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,\_,0,1,1,\_,\_,\_], 0, 1) :- !.

next\_move\_o([\_,\_,\_,1,0,1,\_,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,\_,1,1,0,\_,\_,\_], 2, 1) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,0,1,1], 0, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,1,0,1], 1, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,1,1,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,1,\_,\_,1,\_,\_], 0, 0) :- !.

next\_move\_o([1,\_,\_,0,\_,\_,1,\_,\_], 0, 1) :- !.

next\_move\_o([1,\_,\_,1,\_,\_,0,\_,\_], 0, 2) :- !.

next\_move\_o([\_,0,\_,\_,1,\_,\_,1,\_], 1, 0) :- !.

next\_move\_o([\_,1,\_,\_,0,\_,\_,1,\_], 1, 1) :- !.

next\_move\_o([\_,1,\_,\_,1,\_,\_,0,\_], 1, 2) :- !.

next\_move\_o([\_,\_,0,\_,\_,1,\_,\_,1], 2, 0) :- !.

next\_move\_o([\_,\_,1,\_,\_,0,\_,\_,1], 2, 1) :- !.

next\_move\_o([\_,\_,1,\_,\_,1,\_,\_,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,\_,1,\_,\_,\_,1], 0, 0) :- !.

next\_move\_o([1,\_,\_,\_,0,\_,\_,\_,1], 1, 1) :- !.

next\_move\_o([1,\_,\_,\_,1,\_,\_,\_,0], 2, 2) :- !.

next\_move\_o([\_,\_,0,\_,1,\_,1,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,1,\_,0,\_,1,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,1,\_,1,\_,0,\_,\_], 0, 2) :- !.

%Если центр занят, то занимает левый верхний угол. (Уровень важности ход: 4)

next\_move\_o([0, \_,\_, \_, 1, \_, \_, \_, \_], 0, 0):- !.

%Если центр заняли мы (o), а игрок занял одну из угловых позиций, то занять следующию позицию (Уровень важности ход: 5)

next\_move\_o([0, \_, \_, \_, 2, \_, \_, \_, 1], 0, 0):- !.

next\_move\_o([\_, \_, 1, \_, 2, \_, 0, \_, \_], 0, 2):- !.

next\_move\_o([\_, \_, 0, \_, 2, \_, 1, \_, \_], 2, 0):- !.

next\_move\_o([1, \_, \_, \_, 2, \_, \_, \_, 0], 2, 2):- !.

%В остальных случаях возможны следующие ходы (Уровень важности ход: 6)

next\_move\_o(\_, 0, 0).

next\_move\_o(\_, 0, 2).

next\_move\_o(\_, 2, 0).

next\_move\_o(\_, 2, 2).

next\_move\_o(\_, 0, 1).

next\_move\_o(\_, 1, 0).

next\_move\_o(\_, 1, 1).

next\_move\_o(\_, 1, 2).

next\_move\_o(\_, 2, 1).

%Если центр заняли мы (o), а игрок занял одну из угловых позиций (Уровень важности ход: 5)

next\_move\_o([0, \_, \_, \_, 2, \_, \_, \_, 1], 0, 0):- !.

next\_move\_o([\_, \_, 1, \_, 2, \_, 0, \_, \_], 0, 2):- !.

next\_move\_o([\_, \_, 0, \_, 2, \_, 1, \_, \_], 2, 0):- !.

next\_move\_o([1, \_, \_, \_, 2, \_, \_, \_, 0], 2, 2):- !.

next\_move\_o([\_, 0,\_, \_, 2, \_, \_, \_, 1], 1, 0) :- !.

next\_move\_o([\_, 0,1, \_, 2, \_, \_, \_, \_], 1, 0) :- !.

next\_move\_o([\_, 0,\_, \_, 2, \_, 1, \_, \_], 1, 0) :- !.

next\_move\_o([1, 0,\_, \_, 2, \_, \_, \_, \_], 1, 0) :- !.

%В остальных случаях возможны следующие ходы (Уровень важности ход: 6)

next\_move\_o(\_, 0, 0).

next\_move\_o(\_, 1, 0).

next\_move\_o(\_, 2, 0).

next\_move\_o(\_, 1, 0).

next\_move\_o(\_, 1, 1).

next\_move\_o(\_, 1, 2).

next\_move\_o(\_, 2, 0).

next\_move\_o(\_, 2, 1).

next\_move\_o(\_, 2, 2).

**Приложение 3**

Код программы:

:- use\_module(library(pce)).

%Окно старта игры

game\_menu():-

new(Main, dialog('Крестики-нолики')),

new(Start,button('Начать игру', message(@prolog, game))),

new(Exit,button('Выход', message(Main, destroy))),

send(Main, append, Start),

send(Main, append, Exit, below),

send(Main, open).

%Окно игры

game() :-

new(Game, dialog('Игра')),

new(Field1, button('X')),

new(Field2, button('X')),

new(Field3, button('X')),

new(Field4, button('X')),

new(Field5, button('X')),

new(Field6, button('X')),

new(Field7, button('X')),

new(Field8, button('X')),

new(Field9, button('X')),

new(Click1,button('1', message(@prolog, change, Field1, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click2,button('2', message(@prolog, change, Field2, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click3,button('3', message(@prolog, change, Field3, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click4,button('4', message(@prolog, change, Field4, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click5,button('5', message(@prolog, change, Field5, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click6,button('6', message(@prolog, change, Field6, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click7,button('7', message(@prolog, change, Field7, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click8,button('8', message(@prolog, change, Field8, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

new(Click9,button('9', message(@prolog, change, Field9, Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9))),

send(Game, append, Field1),

send(Game, append, Field2, right),

send(Game, append, Field3, right),

send(Game, append, Click1, below),

send(Game, append, Click2, right),

send(Game, append, Click3, right),

send(Game, append, Field4, below),

send(Game, append, Field5, right),

send(Game, append, Field6, right),

send(Game, append, Click4, below),

send(Game, append, Click5, right),

send(Game, append, Click6, right),

send(Game, append, Field7, below),

send(Game, append, Field8, right),

send(Game, append, Field9, right),

send(Game, append, Click7, below),

send(Game, append, Click8, right),

send(Game, append, Click9, right),

send(Game, open),

game1([Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8, Field9]).

%Начально состояние

game1(ListField) :-

change\_field(ListField, [0,0,0,0,0,0,0,0,0]).

%0 - Незанятое поле

symbol\_to\_number(0, ' ').

%1 - крестик

symbol\_to\_number(1, 'X').

%2 - нолик

symbol\_to\_number(2, 'O').

%Изменение состояния полей FieldN

change\_field(ListField, State) :-

get\_element(State, 0, SymbolA),

get\_element(State, 1, SymbolB),

get\_element(State, 2, SymbolC),

get\_element(State, 3, SymbolD),

get\_element(State, 4, SymbolE),

get\_element(State, 5, SymbolF),

get\_element(State, 6, SymbolG),

get\_element(State, 7, SymbolH),

get\_element(State, 8, SymbolI),

symbol\_to\_number(SymbolA, NumberA),

symbol\_to\_number(SymbolB, NumberB),

symbol\_to\_number(SymbolC, NumberC),

symbol\_to\_number(SymbolD, NumberD),

symbol\_to\_number(SymbolE, NumberE),

symbol\_to\_number(SymbolF, NumberF),

symbol\_to\_number(SymbolG, NumberG),

symbol\_to\_number(SymbolH, NumberH),

symbol\_to\_number(SymbolI, NumberI),

get\_element(ListField, 0, A),

get\_element(ListField, 1, B),

get\_element(ListField, 2, C),

get\_element(ListField, 3, D),

get\_element(ListField, 4, E),

get\_element(ListField, 5, F),

get\_element(ListField, 6, G),

get\_element(ListField, 7, H),

get\_element(ListField, 8, I),

send(A, name, NumberA),

send(B, name, NumberB),

send(C, name, NumberC),

send(D, name, NumberD),

send(E, name, NumberE),

send(F, name, NumberF),

send(G, name, NumberG),

send(H, name, NumberH),

send(I, name, NumberI).

change(X, A, B, C, D, E, F, J, H, I) :-

get(X, name, X1),

symbol\_to\_number(N, X1),

N = 0,

send(X, name, 'X'),

get\_state([A, B, C, D, E, F, J, H, I], State), !, % Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

not(check(State)),

Symbol is 2,

next\_move\_o(State, X0, Y0),

make\_move(X0, Y0, State, Symbol, NewState), !, % Здесь отсечение используется для прекращения процесса порождения и проверки возможных решений.

change\_field([A, B, C, D, E, F, J, H, I], NewState),

not(check(NewState)).

%Проверка того, что поле свободно.

is\_free(X, Y, State) :-

N is (Y \* 3 + X),

get\_element(State, N, A),

A = 0.

%Выполнение хода

make\_move(X, Y, State, Symbol, State\_next) :-

is\_free(X, Y, State),

N is Y\*3+X,

delete\_element(State, N, State1),

insert\_element(State1, Symbol, N, State\_next).

%Проверка того, что победили крестики

check(State) :-

state\_win\_x(StateX),

member(State, [StateX]),

new(Dialog, dialog("Результат")),

new(F, text("Крестики победили(Победа игрока)")),

send(Dialog, append, F),

send(Dialog, open), !. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Проверка того, что победили нолики

check(State) :-

state\_win\_o(StateO),

member(State, [StateO]),

new(Dialog, dialog("Результат")),

new(F, text("Нолики победили(Проигрыш игрока)")),

send(Dialog, append, F),

send(Dialog, open), !. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Проверка того, что игра закончилась ничьей

check(State) :-

not(member(0, State)),

new(Dialog, dialog("Ничья")),

new(F, text("Ничья")),

send(Dialog, append, F),

send(Dialog, open),!. %Отсечение используется для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%Варианты ситуаций игры, когда побеждают крестики (Победа игрока)

state\_win\_x([1,1,1,\_,\_,\_,\_,\_,\_]).

state\_win\_x([\_,\_,\_,1,1,1,\_,\_,\_]).

state\_win\_x([\_,\_,\_,\_,\_,\_,1,1,1]).

state\_win\_x([1,\_,\_,1,\_,\_,1,\_,\_]).

state\_win\_x([\_,1,\_,\_,1,\_,\_,1,\_]).

state\_win\_x([\_,\_,1,\_,\_,1,\_,\_,1]).

state\_win\_x([1,\_,\_,\_,1,\_,\_,\_,1]).

state\_win\_x([\_,\_,1,\_,1,\_,1,\_,\_]).

%Варианты ситуаций игры, когда побеждают нолики (Проигрыш игрока)

state\_win\_o([2,2,2,\_,\_,\_,\_,\_,\_]).

state\_win\_o([\_,\_,\_,2,2,2,\_,\_,\_]).

state\_win\_o([\_,\_,\_,\_,\_,\_,2,2,2]).

state\_win\_o([2,\_,\_,2,\_,\_,2,\_,\_]).

state\_win\_o([\_,2,\_,\_,2,\_,\_,2,\_]).

state\_win\_o([\_,\_,2,\_,\_,2,\_,\_,2]).

state\_win\_o([2,\_,\_,\_,2,\_,\_,\_,2]).

state\_win\_o([\_,\_,2,\_,2,\_,2,\_,\_]).

%Получение состояния

get\_state(ListField, [A, B, C, D, E, F, G, H, I]) :-

get\_element(ListField, 0, A2),

get\_element(ListField, 1, B2),

get\_element(ListField, 2, C2),

get\_element(ListField, 3, D2),

get\_element(ListField, 4, E2),

get\_element(ListField, 5, F2),

get\_element(ListField, 6, G2),

get\_element(ListField, 7, H2),

get\_element(ListField, 8, I2),

get(A2, name, A1),

get(B2, name, B1),

get(C2, name, C1),

get(D2, name, D1),

get(E2, name, E1),

get(F2, name, F1),

get(G2, name, G1),

get(H2, name, H1),

get(I2, name, I1),

symbol\_to\_number(A, A1),

symbol\_to\_number(B, B1),

symbol\_to\_number(C, C1),

symbol\_to\_number(D, D1),

symbol\_to\_number(E, E1),

symbol\_to\_number(F, F1),

symbol\_to\_number(G, G1),

symbol\_to\_number(H, H1),

symbol\_to\_number(I, I1).

%Получение элемента по индексу

get\_element([Head|\_], 0, Head).

get\_element([\_|[Head1|Tail]], N, Element) :-

N1 is (N - 1),

get\_element([Head1|Tail], N1, Element).

%Вставка элемента по индексу

insert\_element([], Element, 0, [Element]).

insert\_element([Head|Tail], Element, 0, [Element|[Head|Tail]]).

insert\_element([Head|Tail], Element, N, [Head|Tail1]) :-

(N1 is N - 1),

insert\_element(Tail, Element, N1, Tail1).

%Удаление элемента по индексу

delete\_element([\_|Tail], 0, Tail).

delete\_element([Head|Tail], N, [Head|Tail1]) :-

N1 is (N - 1),

delete\_element(Tail, N1, Tail1).

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%Все следующие отсечения используются для указания Прологу на то, что найдено нужное правило для заданного целевого утверждения;

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%Если центр не занят, то занимаем его. (Уровень важности ход: 1) Чем ниже уровень, тем он важнее

next\_move\_o([\_, \_,\_, \_, 0, \_, \_, \_, \_], 1, 1) :- !.

%Возможные ситуации, при которых, данный ход является выигрышным,тогда занимаем победную позицию. (Уровень важности ход: 2)

next\_move\_o([0,2,2,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

next\_move\_o([2,0,2,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

next\_move\_o([2,2,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,\_,0,2,2,\_,\_,\_], 0, 1) :- !.

next\_move\_o([\_,\_,\_,2,0,2,\_,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,\_,2,2,0,\_,\_,\_], 2, 1) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,0,2,2], 0, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,2,0,2], 1, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,2,2,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,2,\_,\_,2,\_,\_], 0, 0) :- !.

next\_move\_o([2,\_,\_,0,\_,\_,2,\_,\_], 0, 1) :- !.

next\_move\_o([2,\_,\_,2,\_,\_,0,\_,\_], 0, 2) :- !.

next\_move\_o([\_,0,\_,\_,2,\_,\_,2,\_], 1, 0) :- !.

next\_move\_o([\_,2,\_,\_,0,\_,\_,2,\_], 1, 1) :- !.

next\_move\_o([\_,2,\_,\_,2,\_,\_,0,\_], 1, 2) :- !.

next\_move\_o([\_,\_,0,\_,\_,2,\_,\_,2], 2, 0) :- !.

next\_move\_o([\_,\_,2,\_,\_,0,\_,\_,2], 2, 1) :- !.

next\_move\_o([\_,\_,2,\_,\_,2,\_,\_,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,\_,2,\_,\_,\_,2], 0, 0) :- !.

next\_move\_o([2,\_,\_,\_,0,\_,\_,\_,2], 1, 1) :- !.

next\_move\_o([2,\_,\_,\_,2,\_,\_,\_,0], 2, 2) :- !.

next\_move\_o([\_,\_,0,\_,2,\_,2,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,2,\_,0,\_,2,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,2,\_,2,\_,0,\_,\_], 0, 2) :- !.

%Возможные ситуации, в которой нужно предотвратить выигрыш противника (Уровень важности ход: 3)

next\_move\_o([0,1,1,\_,\_,\_,\_,\_,\_], 0, 0) :- !.

next\_move\_o([1,0,1,\_,\_,\_,\_,\_,\_], 1, 0) :- !.

next\_move\_o([1,1,0,\_,\_,\_,\_,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,\_,0,1,1,\_,\_,\_], 0, 1) :- !.

next\_move\_o([\_,\_,\_,1,0,1,\_,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,\_,1,1,0,\_,\_,\_], 2, 1) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,0,1,1], 0, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,1,0,1], 1, 2) :- !.

next\_move\_o([\_,\_,\_,\_,\_,\_,1,1,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,1,\_,\_,1,\_,\_], 0, 0) :- !.

next\_move\_o([1,\_,\_,0,\_,\_,1,\_,\_], 0, 1) :- !.

next\_move\_o([1,\_,\_,1,\_,\_,0,\_,\_], 0, 2) :- !.

next\_move\_o([\_,0,\_,\_,1,\_,\_,1,\_], 1, 0) :- !.

next\_move\_o([\_,1,\_,\_,0,\_,\_,1,\_], 1, 1) :- !.

next\_move\_o([\_,1,\_,\_,1,\_,\_,0,\_], 1, 2) :- !.

next\_move\_o([\_,\_,0,\_,\_,1,\_,\_,1], 2, 0) :- !.

next\_move\_o([\_,\_,1,\_,\_,0,\_,\_,1], 2, 1) :- !.

next\_move\_o([\_,\_,1,\_,\_,1,\_,\_,0], 2, 2) :- !.

next\_move\_o([0,\_,\_,\_,1,\_,\_,\_,1], 0, 0) :- !.

next\_move\_o([1,\_,\_,\_,0,\_,\_,\_,1], 1, 1) :- !.

next\_move\_o([1,\_,\_,\_,1,\_,\_,\_,0], 2, 2) :- !.

next\_move\_o([\_,\_,0,\_,1,\_,1,\_,\_], 2, 0) :- !.

next\_move\_o([\_,\_,1,\_,0,\_,1,\_,\_], 1, 1) :- !.

next\_move\_o([\_,\_,1,\_,1,\_,0,\_,\_], 0, 2) :- !.

%Если центр занят, то занимает левый верхний угол. (Уровень важности ход: 4)

next\_move\_o([0, \_,\_, \_, 1, \_, \_, \_, \_], 0, 0):- !.

%Если центр заняли мы (o), а игрок занял одну из угловых позиций, то занять следующию позицию (Уровень важности ход: 5)

next\_move\_o([0, \_, \_, \_, 2, \_, \_, \_, 1], 0, 0):- !.

next\_move\_o([\_, \_, 1, \_, 2, \_, 0, \_, \_], 0, 2):- !.

next\_move\_o([\_, \_, 0, \_, 2, \_, 1, \_, \_], 2, 0):- !.

next\_move\_o([1, \_, \_, \_, 2, \_, \_, \_, 0], 2, 2):- !.

%В остальных случаях возможны следующие ходы (Уровень важности ход: 6)

next\_move\_o(\_, 0, 0).

next\_move\_o(\_, 0, 2).

next\_move\_o(\_, 2, 0).

next\_move\_o(\_, 2, 2).

next\_move\_o(\_, 0, 1).

next\_move\_o(\_, 1, 0).

next\_move\_o(\_, 1, 1).

next\_move\_o(\_, 1, 2).

next\_move\_o(\_, 2, 1).