Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет

К РЕАЛИЗАЦИИ РЕШЕНИЯ НА PROLOG
пасьянса «Свободная ячейка»

Студент _____ _____ Коваленко И.А.
(Группа) (Подпись, дата) (И.О. Фамилия)

Студент _____ _____ Кобаренков И.В.
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель проекта _____ Строганов Ю.В.
(Подпись, дата) (И.О. Фамилия)

2021 г.

Оглавление

Правила игры «Свободная ячейка»

Используется колода карт из n цветов и m карт. Раскладывается вся колода в количество столбцов n * 2. Также есть n ячеек, именуемых «домом», и n «свободных» ячеек. В начале игры все они пусты. Разрешено перекладывать одну карту из колонки или свободной ячейки:

- в любую другую колонку — на следующую по старшинству карту другого цвета;

  На рисунке 1 представлен пример данного шага.

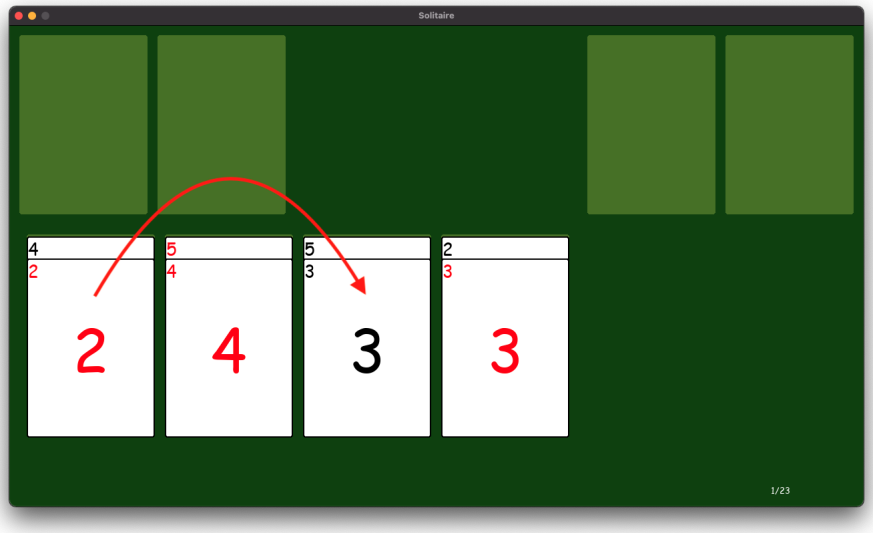

Рис. 1: пример перемещения карты в любую другую колонку — на следующую по старшинству карту другого цвета

- либо на свободную ячейку, если она пуста (таким образом, каждая из свободных ячеек может хранить только одну карту);

  На рисунке 2 представлен пример данного шага.

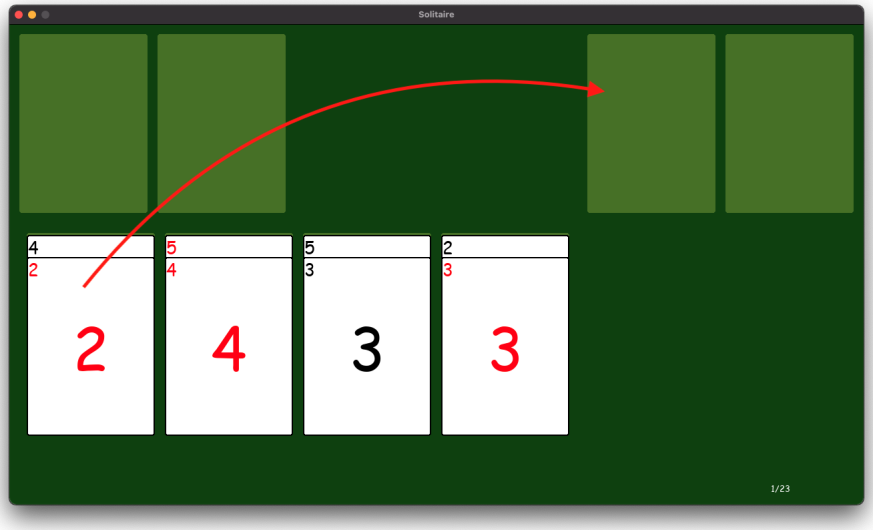

Рис. 2: пример перемещения карты на свободную ячейку, если она пуста

- либо в пустую колонку — без ограничений;

  На рисунке 3 представлен пример данного шага.

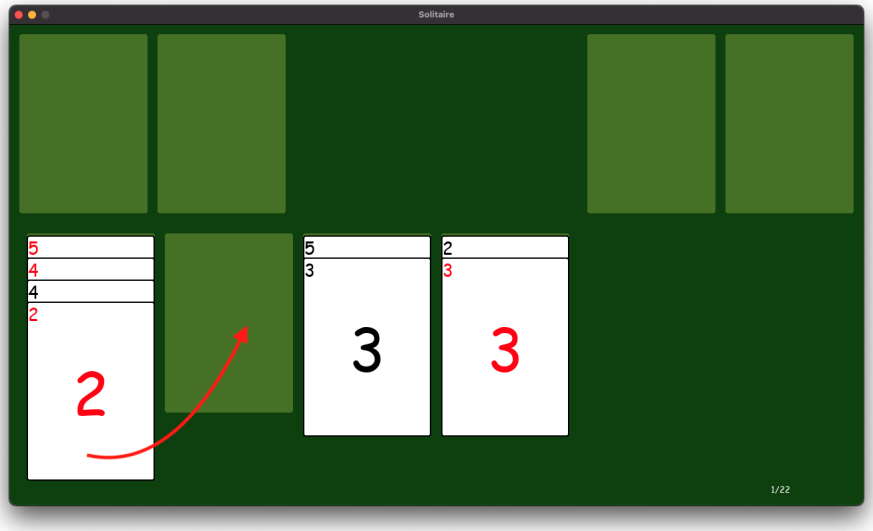

Рис. 3: пример перемещения карты в пустую колонку

- либо в «дом» — карты одной масти, начиная с самой старшей и дальше от младшей к старшей;

  На рисунке 4 представлен пример данного шага.

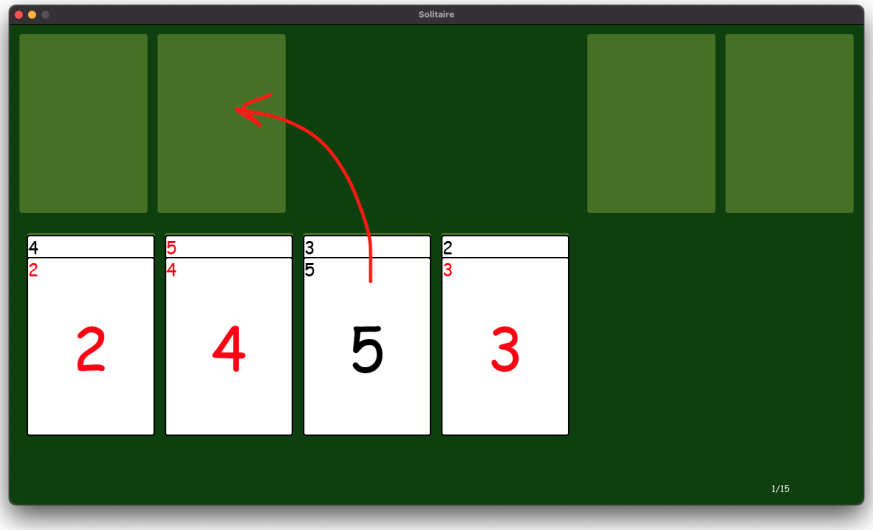

Рис. 4: пример перемещения карты в «дом»

- Если нужно перенести стопку карт, это можно сделать только по одной, используя пустые колонки и свободные ячейки.

Пасьянс сходится, если удаётся переместить всю колоду в «дом».

Цель проекта

Целью проекта является разработка программного обеспечения для получения решения пасьянса «Свободная ячейка» и визуализации найденного решения.

Для реализации поставленной задачи, необходимо выполнить следующие шаги:

- провести анализ существующих решений;
- разработать программу поиска решения на языке Prolog;
- разработать GUI для визуализации найденного решения.

1 Анализ существующих решений

В результате поиска существующих решений для пасьянса «Свободная ячейка», было обнаружено, что над данной проблемой работает только один разработчик.[1]

На данной проблемой он работает уже более 14 лет, им было внесено более 11500 изменений в репозиторий проекта.

В результате проделанной работы им была создана программа и библиотека на языке C, позволяющая решать пасьянс «Свободная ячейка» и похожие на него другие разновидности пасьянса.

2 Выбор языка программирования

Логический поиск решения было необходимо реализовать на языке логического программирования SWI-Prolog.

Для создания GUI было решено использовать Python, поскольку существует модуль для связи кода Python с кодом SWI-Prolog - Pyswip[4]. Для создания самого GUI использовался стандартный модуль Python - pygame[3]. Передача данных между Prolog'ом и Python'ом происходит в формате JSON.

3 Основная проблема

В ходе разработки было обнаружено, что часто программа зацикливается.

Было определено, что проблема в бесконечных повторении одних и тех же равноценных ходов.

Например, если одна карта будет постоянно перемещаться между двумя конкретными столбцами.

Для решения этой проблемы в программу было добавлено хранение набора состояний. После каждого нового сделанного хода, состояние полей игры сохраняется и поиск решения продолжается. Если же данное состояние уже добавлено в набор, ход не засчитывается и поиск продолжается дальше.

На рисунках 3.1-3.4 представлен пример ситуации зацикливания.

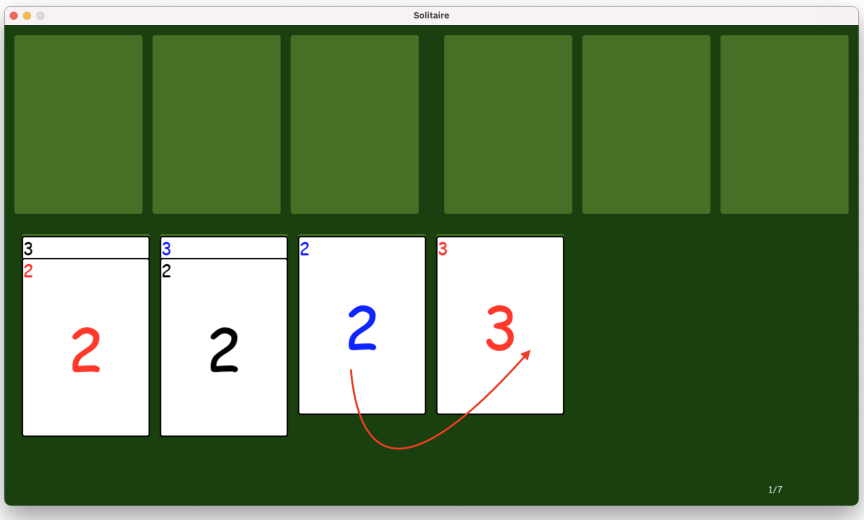

Рис. 3.1: перемещение синей двойки на красную тройку

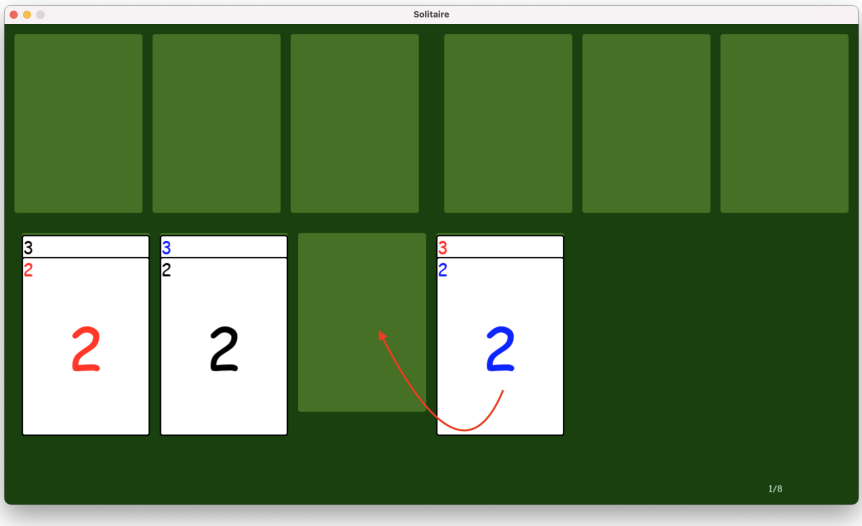

Рис. 3.2: перемещение синей двойки в пустой столбец

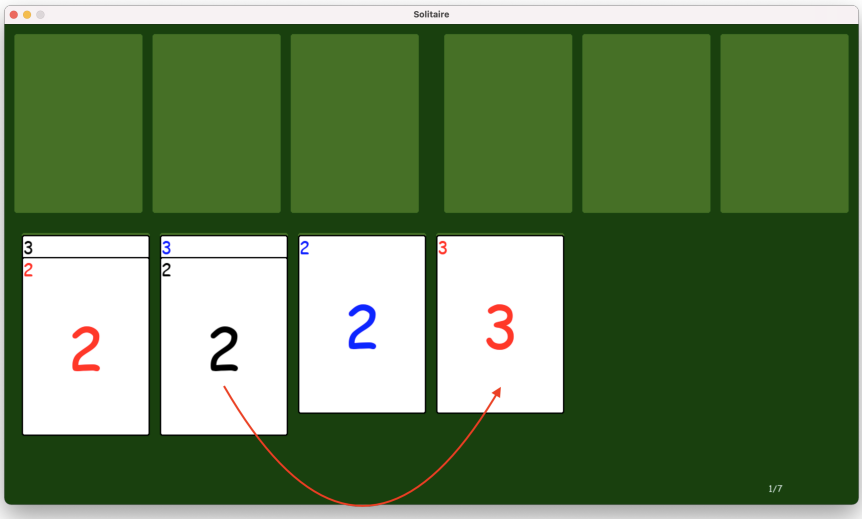

Рис. 3.3: перемещение черной двойки на красную тройку

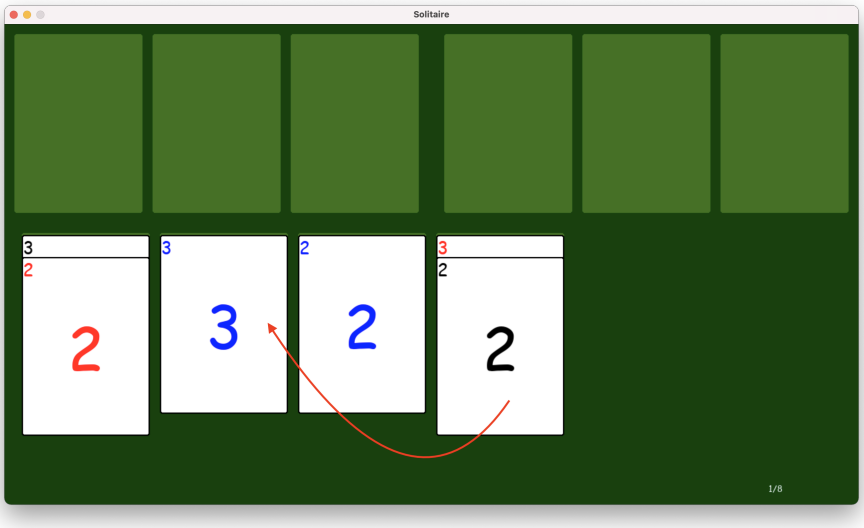

Рис. 3.4: перемещение черной двойки на синюю тройку

Благодаря хранению множества возможных к снятию карт, получилось решить данную проблему и оптимизировать решение.

На рисунках 3.5-3.6 представлен пример состояний, которые наша система считает идентичными.

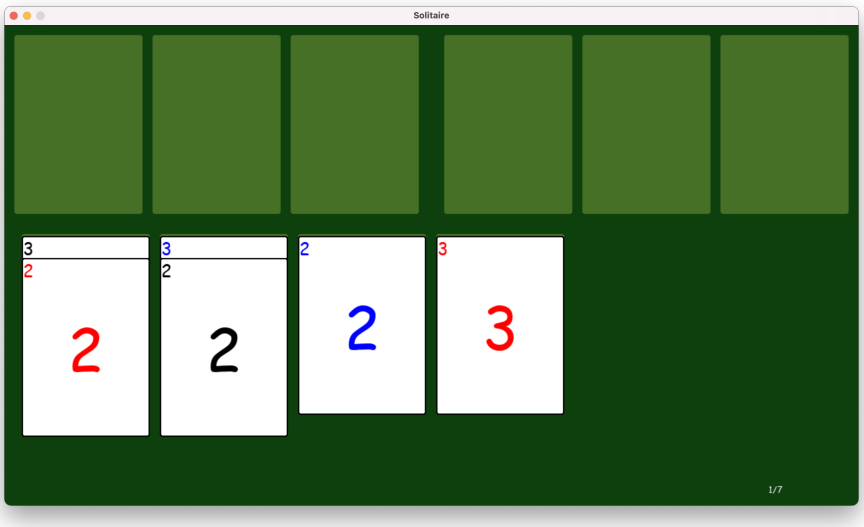

Рис. 3.5: перемещение синей двойки на красную тройку

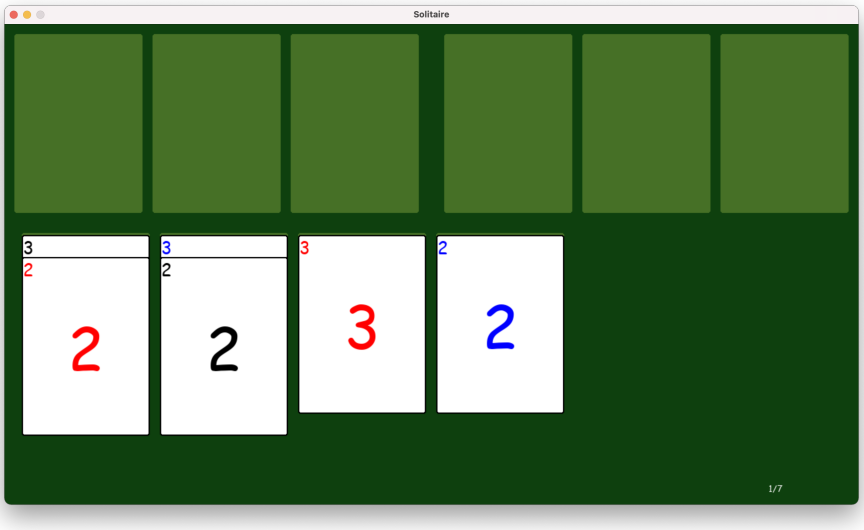

Рис. 3.6: перемещение синей двойки на красную тройку

4 IDEF0

На рисунках 4.1-4.3 представлена IDEF0 диаграмма решения:

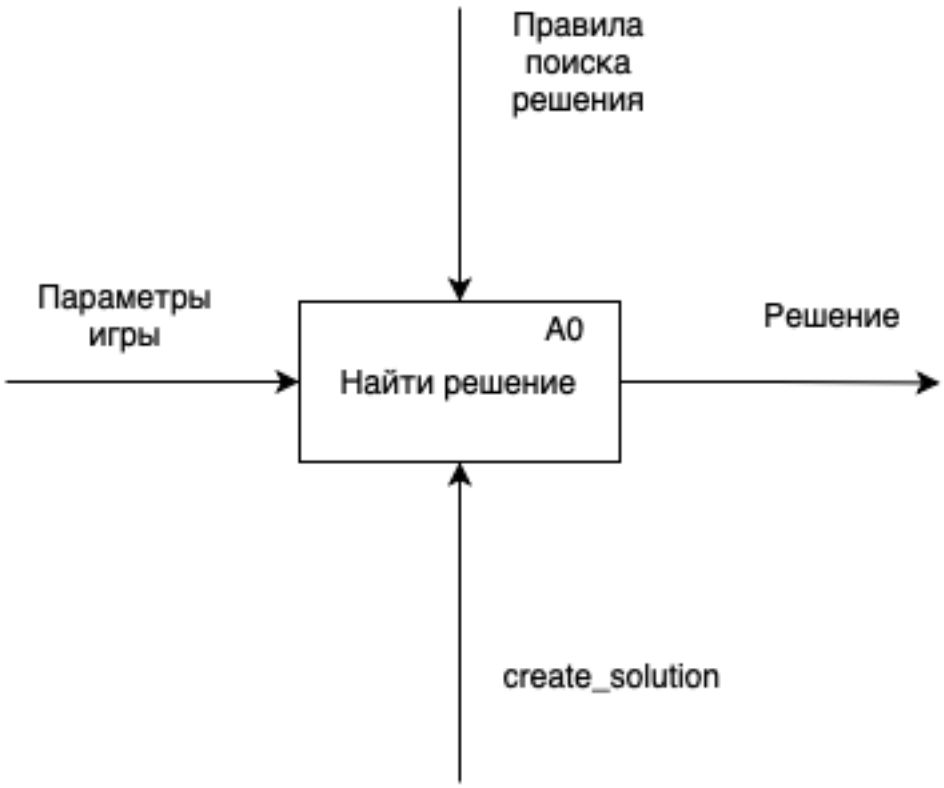

Рис. 4.1: IDEF0 A0

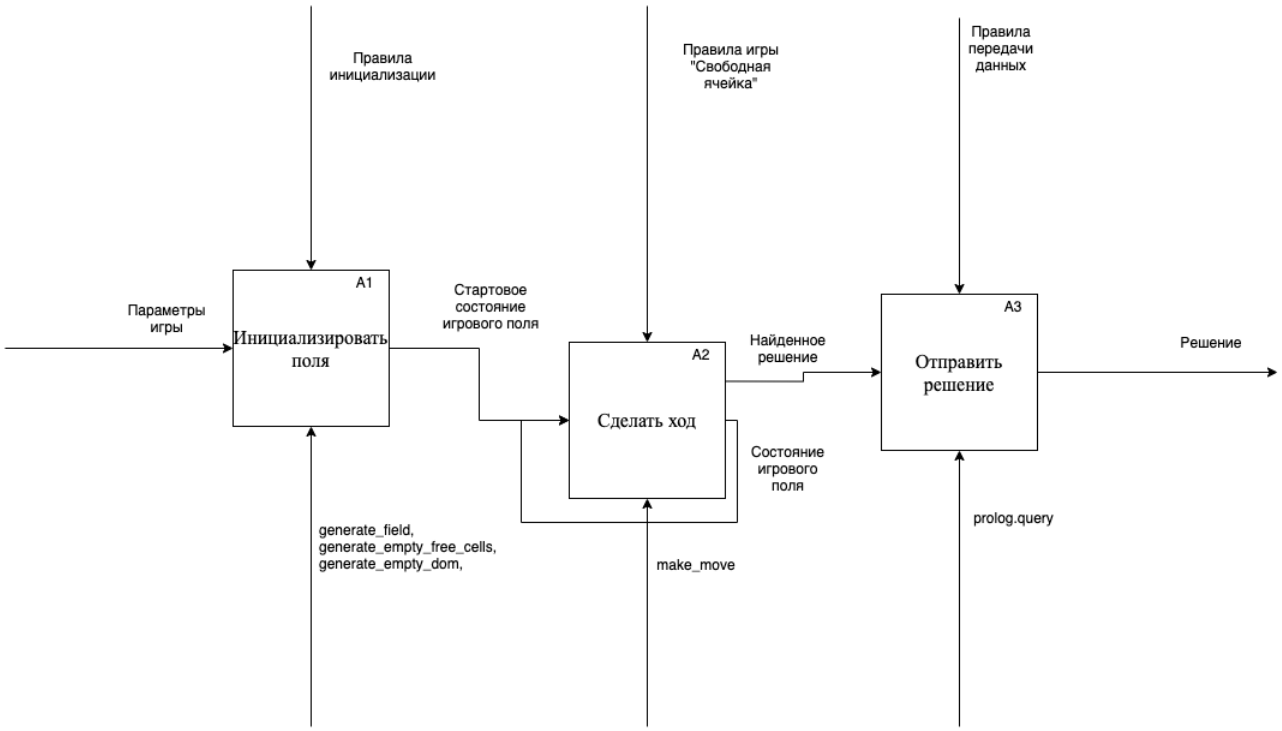

Рис. 4.2: IDEF0 A1-A3

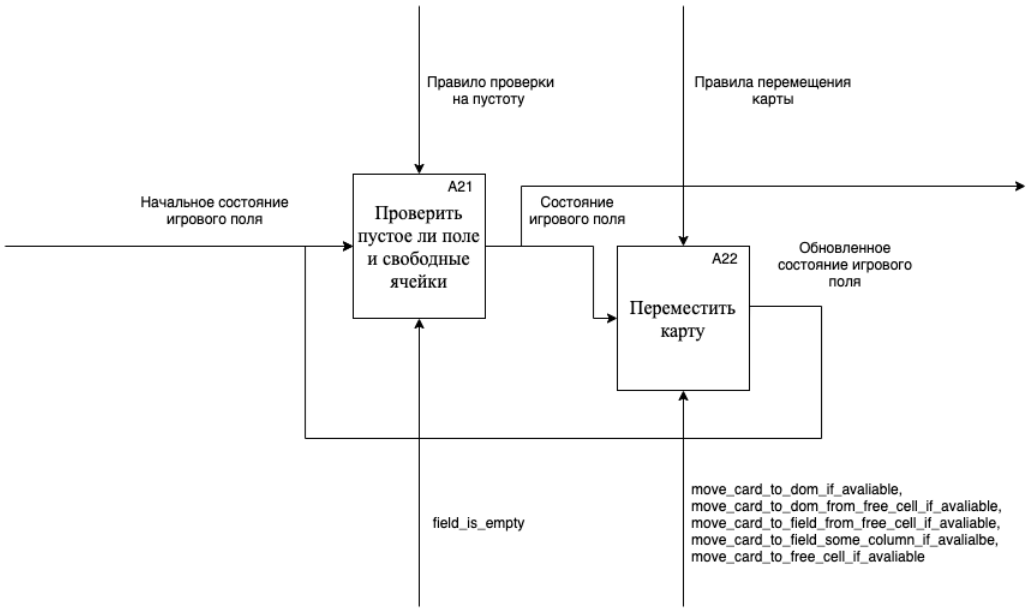

Рис. 4.3: IDEF0 A11-A12

5 Описание правил

В данном разделе описаны все правила, переменные, приведены примеры использования данных правил.

5.1 solve

Правило, с которого начинается работа программы. На основе MaxValue и ColorsCount определяются количество свободных ячеек, размеры поля и дома. Генерируются поле, пустые свободные ячейки и дом. Создаётся переменная State для хранения состояний. В конце проверяется правило, отвечающее за поиск решения.

Переменные:

- MaxValue – хранит количество карт каждого цвета;
- ColorsCount – хранит количество цветов карт;
- Solution – переменная, которая в результате работы программы получает набор состояний, являющийся последовательным решением задачи;
- States – переменная, для хранения набора состояний;
- Dom – переменная, в которой хранятся значения карт, находящихся в доме;
- FreeCells – переменная, в которой хранятся значения карт, находящихся в свободных ячейках;
- Field – переменная, в которой хранится набор карт, находящихся на поле;
- FreeCellsSize – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- ReservedCellsCount – хранит количество занятых свободных ячеек;
- NumberOfColumns – хранит количество столбцов игрового поля Field.

В листинге 5.1 представлена реализация правила solve

Листинг 5.1: реализация правила solve

```
solve(MaxValue, ColorsCount, Solution) :-
        FreeCellsSize is ColorsCount,
        ReservedCellsCount is 0,
        NumberOfColumns is ColorsCount * 2,
        generate_field(MaxValue, ColorsCount, Field),
        generate_empty_free_cells(FreeCells),
        generate_empty_dom(ColorsCount, Dom),
        stm_init(State),
        stm_append_value_if_not_in_stm(State, [Dom, FreeCells, Field], States),
        make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom,
   FreeCellsSize, ReservedCellsCount, States, Solution).
```

В листинге 5.2 представлен пример работы правила solve

Листинг 5.2: пример работы правила solve

```
?- solve(4, 2, Solution).
Solution = [[[[[0,0],[1,0]],[[0,1],[1,1]]],[],[[],[],[],[]]],
[[[[0,0],[1,0]],[[1,1]]],[],[[[0,1]],[],[],[]]],
[[[[0,0],[1,0]],[]],[],[[[0,1]],[],[],[[1,1]]]],
[[[[1,0]],[]],[],[[[0,1]],[],[[0,0]],[[1,1]]]],
[[[],[]],[],[[[0,1]],[[1,0]],[[0,0]],[[1,1]]]]].
```

5.2 make_move

make_move – это набор из шести равноценных правил, пять из которых проверяет возможность совершения одного из ходов в игре: положить карту из поля в дом, из свободных ячеек в дом, из свободных ячеек в поле, из поля в свободные ячейки и переложить карту между столбцами поля. Первое правило нужно для передачи результата в переменную Solution и завершения поиска решения. Если поле и свободные ячейки пусты, переменная Solution успешно унифицируется с переменной, хранящей набор состояний. В остальных правилах сначала создается переменная Card, хранящая значение одной карты, затем, в зависимости от правила, идёт проверка условий и совершение хода данного типа. Если данный набор состояний получен в первый раз, он добавляется в переменную NSSnapshot. В конце снова проверяется правило make_move, но уже с новыми значениями переменных.

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- MaxValue – хранит количество карт каждого цвета;
- NumberOfColumns – хранит количество столбцов поля Field;
- FreeCells – переменная, в которой хранятся значения карт, находящихся в свободных ячейках;

- Dom – переменная, в которой хранятся значения карт, находящихся в доме;
- FreeCellsSize – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- ReservedCellsCount – хранит количество занятых свободных ячеек;
- Snapshots – переменная, для хранения набора состояний;
- Solution – переменная, которая в результате работы программы получает набор состояний, являющийся последовательным решением задачи;
- Card – переменная для хранения значения одной карты;
- RowOfCard – переменная хранит длину столбца, наверху которого находится Card;
- FieldResult – переменная, в которой хранится набор карт, находящихся на поле, после совершения хода;
- DomResult – переменная, в которой хранятся значения карт, находящихся в доме, после совершения хода;
- FreeCellsResult – переменная, в которой хранятся значения карт, находящихся в свободных ячейках, после совершения хода;
- ReservedFreeCellsCountResult – переменная хранит количество занятых свободных ячеек, после совершения хода;
- NSSnapshot(Next Step Snapshot) – переменная, в которой хранится обновленный набор состояний, после совершения хода.

В листинге 5.3 представлена реализация make_move

Листинг 5.3: реализация make_move

```
make_move(Field, _, _, _, _, _, 0, Solution, Solution) :-
        field_is_empty(Field), !.

make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom, FreeCellsSize,
   ReservedFreeCells, Snapshots, Solution) :-
        get_top_card(Field, Card, RowOfCard),
        move_card_to_dom_if_avaliable(Card, Field, RowOfCard, Dom, MaxValue,
   FieldResult, DomResult),
        stm_append_value_if_not_in_stm(Snapshots, [DomResult, FreeCells,
   FieldResult], NSSnapshot),
        make_move(FieldResult, MaxValue, NumberOfColumns, FreeCells, DomResult,
    FreeCellsSize, ReservedFreeCells, NSSnapshot, Solution), !.

make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom, FreeCellsSize,
   ReservedFreeCells, Snapshots, Solution) :-
```

```
        process_free_cell_card(FreeCells, Card),
        move_card_to_dom_from_free_cell_if_avaliable(Card, Dom, FreeCells,
    ReservedFreeCells, DomResult, FreeCellsResult, ReservedFreeCellsCountResult,
     MaxValue),
        stm_append_value_if_not_in_stm(Snapshots, [DomResult, FreeCellsResult,
    Field], NSSnapshot),
        make_move(Field, MaxValue, NumberOfColumns, FreeCellsResult, DomResult,
     FreeCellsSize, ReservedFreeCellsCountResult, NSSnapshot, Solution), !.

make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom, FreeCellsSize,
    ReservedFreeCells, Snapshots, Solution) :-
        process_free_cell_card(FreeCells, Card),
        move_card_to_field_from_free_cell_if_avaliable(Card, Field, FreeCells,
    NumberOfColumns, ReservedFreeCells, FieldResult, FreeCellsResult,
    ReservedFreeCellsCountResult),
        stm_append_value_if_not_in_stm(Snapshots, [Dom, FreeCellsResult,
    FieldResult], NSSnapshot),
        make_move(FieldResult, MaxValue, NumberOfColumns, FreeCellsResult, Dom,
     FreeCellsSize, ReservedFreeCellsCountResult, NSSnapshot, Solution), !.

make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom, FreeCellsSize,
    ReservedFreeCells, Snapshots, Solution) :-
        get_top_card(Field, _, RowOfCard),
        move_card_to_field_some_column_if_avalialbe(Field, NumberOfColumns,
    RowOfCard, FieldResult),
        stm_append_value_if_not_in_stm(Snapshots, [Dom, FreeCells, FieldResult
    ], NSSnapshot),
        make_move(FieldResult, MaxValue, NumberOfColumns, FreeCells, Dom,
    FreeCellsSize, ReservedFreeCells, NSSnapshot, Solution), !.

make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom, FreeCellsSize,
    ReservedFreeCells, Snapshots, Solution) :-
        get_top_card(Field, Card, RowOfCard),
        move_card_to_free_cell_if_avaliable(Card, RowOfCard, Field, FreeCells,
    FreeCellsSize, ReservedFreeCells, FieldResult, FreeCellsResult,
    ReservedCellsCountResult),
        stm_append_value_if_not_in_stm(Snapshots, [Dom, FreeCellsResult,
    FieldResult], NSSnapshot),
        make_move(FieldResult, MaxValue, NumberOfColumns, FreeCellsResult, Dom,
     FreeCellsSize, ReservedCellsCountResult, NSSnapshot, Solution), !.
```

В листинге 5.4 представлен пример работы правила make_move

Листинг 5.4: пример работы правила make_move

```
?- make_move([[[0, 0], [2, 1]],
[[2, 0], [3, 0]],
[[1, 1], [3, 1]],
[[1, 0], [0, 1]]],
          4, 2,
          [],
          [[],[]],
          2, 0, [], Solution)

Solution = [[[[[2, 1], [1, 1], [0, 1], [3, 1]], [[2, 0], [1, 0], [0, 0], [3,
    0]]], [], [[], [], [], []]], [[[[2, 1], [1, 1], [0, 1], [3, 1]], [[1, 0],
    [0, 0], [3, 0]]], [[2, 0]], [[], [], [], []]], [[[[2, 1], [1, 1], [0, 1],
    [3, 1]], [[0, 0], [3, 0]]], [[1, 0], [2, 0]], [[], [], [], []]], [[[[2, 1],
    [1, 1], [0, 1], [3, 1]], [[3, 0]]], [[1, 0], [2, 0]], [[[0, 0]], [], [],
```

```
[]]], [[[[2, 1], [1, 1], [0, 1], [3, 1]], []], [[1, 0], [2, 0]], [[[0, 0]],
[[3, 0]], [], []]], [[[[2, 1], [1, 1], [0, 1], [3, 1]], []], [[1, 0]], [[[0,
 0]], [[2, 0], [3, 0]], [], []]], [[[[2, 1], [1, 1], [0, 1], [3, 1]], []],
[[0, 0], [1, 0]], [[], [[2, 0], [3, 0]], [], []]], [[[[2, 1], [1, 1], [0,
1], [3, 1]], []], [[0, 0]], [[[1, 0]], [[2, 0], [3, 0]], [], []]], [[[[1,
1], [0, 1], [3, 1]], []], [[2, 1], [0, 0]], [[[1, 0]], [[2, 0], [3, 0]], [],
 []]], [[[[0, 1], [3, 1]], []], [[2, 1], [0, 0]], [[[1, 0]], [[1, 1], [2,
0], [3, 0]], [], []]], [[[[0, 1], [3, 1]], []], [[2, 1]], [[[1, 0]], [[0,
0], [1, 1], [2, 0], [3, 0]], [], []]], [[[[0, 1], [3, 1]], []], [[1, 0], [2,
 1]], [[], [[0, 0], [1, 1], [2, 0], [3, 0]], [], []]], [[[[0, 1], [3, 1]],
[]], [[1, 0]], [[[2, 1]], [[0, 0], [1, 1], [2, 0], [3, 0]], [], []]], [[[[0,
 1], [3, 1]], []], [], [[[1, 0], [2, 1]], [[0, 0], [1, 1], [2, 0], [3, 0]],
[], []]], [[[[3, 1]], []], [], [[[1, 0], [2, 1]], [[0, 0], [1, 1], [2, 0],
[3, 0]], [], [[0, 1]]]], [[[[3, 1]], []], [], [[[2, 1]], [[0, 0], [1, 1],
[2, 0], [3, 0]], [], [[1, 0], [0, 1]]]], [[[[3, 1]], []], [], [[[0, 0], [2,
1]], [[1, 1], [2, 0], [3, 0]], [], [[1, 0], [0, 1]]]], [[[], []], [], [[[0,
0], [2, 1]], [[1, 1], [2, 0], [3, 0]], [[3, 1]], [[1, 0], [0, 1]]]]]
```

5.3 is_equal_sets

Правило проверяет эквивалентность двух списков List1 и List2, предварительно переведя их в множества Set1 и Set2 соответственно, после сортирует полученные множества и сохраняет результат в переменные SortedSet1 и SortedSet2 соответственно. Полученные отсортированные множества сравниваются на эквивалентность.

Переменные:

- List1 – список 1;
- List2 – список 2;
- Set1 – переменная, в которой хранится List1 без дубликатов значений;
- Set2 – переменная, в которой хранится List2 без дубликатов значений;
- SortedSet1 – переменная, в которой хранится отсортированная версия Set1;
- SortedSet2 – переменная, в которой хранится отсортированная версия Set2.

В листинге 5.5 представлена реализация правила is_equal_sets

Листинг 5.5: реализация правила is_equal_sets

```
is_equal_sets(List1, List2) :-
        list_to_set(List1, Set1),
        list_to_set(List2, Set2),
        sort(Set1, SortedSet1),
        sort(Set2, SortedSet2),
        is_equal(SortedSet1, SortedSet2).
```

В листинге 5.6 представлены примеры работ правила is_equal_sets

Листинг 5.6: примеры работ правила is_equal_sets

```
?- is_equal_sets([1,3,2,3,5],[3,1,2,5]).
true

?- is_equal_sets([1,3,2,3,2],[3,9,2,4]).
false
```

5.4 insert

Этот набор равноценных правил используется для добавления в хвост списка значения переменной Value.

Переменные:

- Value – переменная, значение которой помещается в хвост списка;
- H – переменная, обозначающая голову списка;
- T, T2 – переменные, обозначающие хвост списка.

В листинге 5.7 представлена реализация правила insert

Листинг 5.7: реализация правила insert

```
insert([H | T], [H | T2], Value) :-
        insert(T, T2, Value), !.

insert([H | []], [H, Value], Value) :- !.

insert([], [Value], Value) :- !.
```

В листинге 5.8 представлен пример работы правила insert

Листинг 5.8: пример работы правила insert

```
?- insert([1,2,3],Result, 4)
Result = [1,2,3,4]
```

5.5 insert_to_head

Этот набор равноценных правил используется для добавления в голову списка значения переменной Element.

Переменные:

- Element – переменная, значение которой помещается в хвост списка;
- H – переменная, обозначающая голову списка;
- T, T2 – переменные, обозначающие хвост списка.

В листинге 5.9 представлена реализация правила insert_to_head

Листинг 5.9: реализация правила insert_to_head

```
insert_to_head([H | T], Element, [Element, H | T]) :- !.

insert_to_head([], Element, [Element]).
```

В листинге 5.10 представлен пример работы правила insert_to_head

Листинг 5.10: пример работы правила insert_to_head

```
insert_to_head([1,2,3],4, Result)
Result = [4, 1, 2, 3]
```

5.6 set_value

Правило задаёт значение переменной.

В листинге 5.11 представлена реализация правила set_value

Листинг 5.11: реализация правила set_value

```
set_value(Hardcode, Hardcode).
```

В листинге 5.12 представлен пример работы правила set_value

Листинг 5.12: пример работы правила set_value

```
set_value(X, 12)
X = 12
```

5.7 is_equal

Правило используется для проверки эквивалентности значений двух переменных.

В листинге 5.13 представлена реализация правила is_equal

Листинг 5.13: реализация правила is_equal

```
is_equal(Value, Value).
```

В листинге 5.14 представлен пример работы правила is_equal

Листинг 5.14: пример работы правила is_equal

```
is_equal(12,12)
true

is_equal(12,13)
false
```

5.8 is_not_equal

Правило используется для проверки того, что значения переменных не эквивалентны.

В листинге 5.15 представлена реализация правила is_not_equal

Листинг 5.15: реализация правила is_not_equal

```
is_not_equal(Value, Value) :- !, false.

is_not_equal(_, _).
```

В листинге 5.16 представлены примеры работы правила is_not_equal

Листинг 5.16: пример работы правила is_not_equal

```
is_not_equal(12,12)
false

is_not_equal(12,13)
true
```

5.9 delete_nth_element_from_list

Правило используется для удаления N-ого элемента списка List, результат сохраняется в переменной ResultList.

Главное правило delete_nth_element_from_list использует рекурсивную версию правила с приставкой _internal. Внутреннее правило двигается по списку и передаёт значение головы списка списку ResultList. Как только количество итераций будет равно значению переменной N, списку ResultList будет передан оставшийся хвост начального списка без N-ого элемента, после чего дальнейшая проверка правила будет прервана.

Переменные:

- List – начальный список, из которого удаляют элемент;
- N – номер позиции в списке элемента, который нужно удалить;
- ResultList – список-результат, без n-ого элемента.

В листинге 5.17 представлена реализация правила delete_nth_element_from_list

Листинг 5.17: реализация правила delete_nth_element_from_list

```
delete_nth_element_from_list(List, N, ResultList) :-
        delete_nth_element_from_list_internal(List, N, ResultList, 0).

delete_nth_element_from_list_internal([_ | T], Iteration, T, Iteration) :- !.
```

```
delete_nth_element_from_list_internal([H | T1], N, [H | T2], Iteration) :-
        NSIteration is Iteration + 1,
        delete_nth_element_from_list_internal(T1, N, T2, NSIteration).
```

В листинге 5.18 представлен пример работы
правила delete_nth_element_from_list

Листинг 5.18: пример работы правила delete_nth_element_from_list

```
delete_nth_element_from_list([1,2,3],1,Res)
Res = [1, 3]
```

5.10 get_element

Правило позволяет получить определенный элемент списка по его индексу. Используется правило nth0, которое позволяет получить значение элемента по индексу.

Переменные:

- List – переменная, хранящая список;
- Index – переменная, хранящая индекс элемента, который нужно получить;
- Element – переменная, в которой сохраняется значение нужного элемента списка.

В листинге 5.19 представлена реализация правила get_element

Листинг 5.19: реализация правила get_element

```
get_element(List, Index, Element) :-
   nth0(Index, List, Element).

get_element([], _, []).

get_element(_, _, []).
```

В листинге 5.20 представлен пример работы правила get_element

Листинг 5.20: пример работы правила get_element

```
get_element([1,2,3,4,5],2,Res)
Res = 3
```

5.11 get_element_with_len

Правило возвращает элемент списка и его длину. Здесь также, как и в правиле get_element, используется правило nth0, после чего используется правило length для получения длинны списка, хранящегося в переменной Element.

Переменные:

- List – переменная, хранящая список;
- Index – переменная, хранящая индекс элемента, который нужно получить;
- Element – переменная, в которой сохраняется значение нужного элемента списка;
- Len – переменная для хранения длины списка Element.

В листинге 5.21 представлена реализация правила get_element_with_len

Листинг 5.21: реализация правила get_element_with_len

```
get_element_with_len(List, Index, Element, Len) :-
        nth0(Index, List, Element),
        length(Element, Len).

get_element_with_len([], _, [], _).

get_element_with_len(_, _, [], _).
```

В листинге 5.22 представлен пример работы правила get_element_with_len

Листинг 5.22: пример работы правила get_element_with_len

```
get_element_with_len([[1,2],[1,2,3],[1,2,3,4,5]],2,Res,ResLen)
Res = [1, 2, 3, 4, 5], ResLen = 5
```

5.12 cnt_mod

Переменная Res принимает значение остатка от деления переменной A на переменную B.

Переменные:

- A, B – численные переменные;
- Res – переменная, в которой сохраняется результат.

В листинге 5.23 представлена реализация правила cnt_mod

Листинг 5.23: реализация правила cnt_mod

```
cnt_mod(A, B, Res) :-
        Res is A mod B.
```

В листинге 5.24 представлен пример работы реализация правила cnt_mod

Листинг 5.24: пример работы правила cnt_mod

```
cnt_mod(6,3, Res)
Res = 0

cnt_mod(6,4, Res)
Res = 2
```

5.13 replace

Правило используется для того, чтобы менять значение элемента списка под номером Index на значение переменной Element. Если Index больше длины списка, значение переменной Element будет вставлено в хвост списка.

Переменные:

- List – переменная, хранящая список;
- Index – переменная, хранящая индекс элемента, который нужно заменить;
- Element – переменная хранящая значение, которое нужно поместить в список;
- Res – переменная возвращающая результат работы правила.

В листинге 5.25 представлена реализация правила replace

Листинг 5.25: реализация правила replace

```
replace(Index, List, Element, Result) :-
  nth0(Index, List, _, Res),
  nth0(Index, Result, Element, Res), !.

replace(Index, List, Element, Res) :-
        length(List, ListLength),
        Index >= ListLength,
        insert(List, Res, Element).
```

В листинге 5.26 представлен пример работы реализация правила replace

Листинг 5.26: реализация правила replace

```
replace(1, [1,2,3,4,5], 9, Result)
Result = [1, 9, 3, 4, 5]
```

5.14 state_inside_state_machine

Правило используется для проверки того находится ли данное [Dom, FreeCells, Field] состояние в наборе или нет. Дом и свободные ячейки проверяются по совпадению с значениями сохраненными в наборе. В поле же проверяются не все элементы, а лишь набор карт, доступных к снятию во всех столбцах. На основе списка этих карт с помощью правила is_equal_sets создаётся сет и уже он сравнивается с предыдущими версиями. Перевод списка в сет позволяет избежать лишних повторяемых ходов.

Переменные:

- Dom – переменная содержит информацию о состоянии дома;

- FreeCells – переменная содержит информацию о состоянии свободных ячеек;
- Field – переменная содержит информацию о состоянии поля;
- StateMachineList- переменная хранящая набор состояний;

В листинге 5.27 представлена реализация правила state_inside_state_machine

Листинг 5.27: реализация правила state_inside_state_machine

```
state_inside_state_machine(StateMachineList, [Dom, FreeCells, Field]) :-
        make_lite_field_downset_snapshot(Field, FieldSnapshot),
        state_inside_state_machine_internal(StateMachineList, [Dom, FreeCells,
   FieldSnapshot]).

state_inside_state_machine_internal([[Dom, FreeCells, FieldSTM] | _], [Dom,
   FreeCells, LiteFieldSnapshot]) :-
        make_lite_field_downset_snapshot(FieldSTM, FieldSnapshotSTM),
        is_equal_sets(FieldSnapshotSTM, LiteFieldSnapshot), !.

state_inside_state_machine_internal([_ | T], State) :-
        state_inside_state_machine_internal(T, State).

state_inside_state_machine_internal([], _) :- !, false.
```

5.15 stm_init

Правило используется для инициализации переменной, в которой будет храниться набор состояний.

Переменные:

- Dom – переменная содержит информацию о состоянии дома;
- FreeCells – переменная содержит информацию о состоянии свободных ячеек;
- Field – переменная содержит информацию о состоянии поля;
- StateMachineList- переменная хранящая набор состояний;

В листинге 5.28 представлена реализация правила stm_init

Листинг 5.28: реализация правила stm_init

```
stm_init(Result) :-
        set_value(Result, []).
```

В листинге 5.29 представлена реализация правила stm_init

Листинг 5.29: реализация правила stm_init

```
stm_init(Result)
Result= []
```

5.16 stm_append_value_if_not_in_stm

Правило добавляет состояние в набор состояний, если оно ещё не было добавлено. Сначала с помощью правила stm_append_value_if_not_in_stm проверяется есть ли, хранящееся в переменной State, состояние в наборе состояний. Если нет, оно добавляется в голову списка.

Переменные:

- Snapshot – текущий набор состояний;
- State – определенное состояние;
- SnapshotResult – обновленный набор состояний.

В листинге 5.30 представлена реализация правила stm_append_value_if_not_in_stm

Листинг 5.30: реализация правила stm_append_value_if_not_in_stm

```
stm_append_value_if_not_in_stm(Snapshot, State, SnapshotResult) :-
        not(state_inside_state_machine(Snapshot, State)), !,
        insert_to_head(Snapshot, State, SnapshotResult).
```

5.17 make_lite_field_downset_snapshot

Этот набор правил используется для создания списка содержащего доступные к снятию карты всех столбцов игрового поля. Делается это для того, чтобы затем в правиле state_inside_state_machine проверить было ли уже добавлено данное состояние или нет.

В листинге 5.31 представлена реализация правила make_lite_field_downset_snapshot

Листинг 5.31: реализация правила make_lite_field_downset_snapshot

```
make_lite_field_downset_snapshot(Field, FieldSnapshot) :-
   make_lite_field_downset_snapshot_internal(Field, FieldSnapshot, []).

make_lite_field_downset_snapshot_internal([[TopCard | ColumnTail] | FieldTail],
    FieldSnapshot, TempSnapshot) :-
        length([TopCard | ColumnTail], N),
        insert_to_head(TempSnapshot, [N, TopCard], NSTempSnapshot),
        make_lite_field_downset_snapshot_internal(FieldTail, FieldSnapshot,
   NSTempSnapshot).

make_lite_field_downset_snapshot_internal([[] | FieldTail], FieldSnapshot,
   TempSnapshot) :-
        insert_to_head(TempSnapshot, [0, []], NSTempSnapshot),
        make_lite_field_downset_snapshot_internal(FieldTail, FieldSnapshot,
   NSTempSnapshot).
```

```
make_lite_field_downset_snapshot_internal([], FieldSnapshot, FieldSnapshot) :-
    !.
```

В листинге 5.32 представлен пример работы правила make_lite_field_downset_snapshot

Листинг 5.32: реализация правила make_lite_field_downset_snapshot

```
make_lite_field_downset_snapshot([[[0, 0], [2, 1]],
[[2, 0], [3, 0]],
[[1, 1], [3, 1]],
[[1, 0], [0, 1]]],Result)
Result = [[2, [1, 0]], [2, [1, 1]], [2, [2, 0]], [2, [0, 0]]]
```

5.18 generate_sorted_array_of_color

Правило используется для создания сортированного списка цветов карт. В листинге 5.33 представлена реализация правила generate_sorted_array_of_color

Листинг 5.33: реализация правила generate_sorted_array_of_color

```
generate_sorted_array_of_color(MaxValue, Color, Result) :-
        generate_sorted_array_of_color_internal(MaxValue, Color, Result, [], 0)
    .

generate_sorted_array_of_color_internal(MaxValue, Color, Result, Temp,
    Iteration) :-
        Iteration < MaxValue,
        insert(Temp, NextTemp, [Iteration, Color]),
        NextIteration is Iteration + 1,
        generate_sorted_array_of_color_internal(MaxValue, Color, Result,
    NextTemp, NextIteration), !.

generate_sorted_array_of_color_internal(Iteration, _, Result, Result, Iteration
    ).
```

В листинге 5.34 представлен пример работы правила generate_sorted_array_of_color

Листинг 5.34: реализация правила generate_sorted_array_of_color

```
generate_sorted_array_of_color(4, 2, Res)
Res = [[0, 2], [1, 2], [2, 2], [3, 2]]
```

5.19 getAllElements

Правило используется для того, чтобы получить все элементы списка. В листинге 5.35 представлена реализация правила getAllElements

Листинг 5.35: реализация правила getAllElements

```
getAllElements([],[]).

getAllElements([H|T], ElementsList) :- getAllElements(T, NewElementsList),

    append(NewElementsList, H, ElementsList).
```

5.20 generate_field

Правило используется для создания поля с перемешанными картами. Сначала создаётся список всех карт, разложенных по порядку, затем карты перемешиваются и раскладываются по полю.

Переменные:

- MaxValue – хранит количество карт каждого цвета;
- NumColor – хранит количество цветов карт;
- AllCards – переменная, хранящая списки всех карт по цветам;
- AllCardsList – переменная, хранящая список всех карт;
- ShuffledCardList – переменная, хранящая список перемешанных карт;
- Field – переменная, хранящая набор всех карт лежащих на поле.

В листинге 5.36 представлена реализация правила generate_field

Листинг 5.36: реализация правила generate_field

```
generate_field(MaxValue, NumColor, Field) :-
        generate_field_internal(MaxValue, NumColor, AllCards, [], 0),
        getAllElements(AllCards, AllCardsList),
        random_permutation(AllCardsList, ShuffledCardList),
        RowsCount is NumColor * 2,
        TotalCards is NumColor * MaxValue,
        throw_cards(ShuffledCardList, RowsCount, TotalCards, Field).

generate_field_internal(MaxValue, NumColor, Field, Temp, Iteration) :-
        Iteration < NumColor,
        generate_sorted_array_of_color(MaxValue, Iteration, SortedResult),
        insert(Temp, NewTemp, SortedResult),
        NextIteration is Iteration + 1,
        generate_field_internal(MaxValue, NumColor, Field, NewTemp,
   NextIteration), !.

generate_field_internal(_, _, Field, Field, _).
```

В листинге 5.37 представлен пример работы правила generate_field

Листинг 5.37: реализация правила generate_field

```
generate_field(2,4,Field)
Field =  [[[0, 1 ]],  [[1, 1 ]],  [[1, 0 ]],  [[0, 2 ]],  [[0, 3 ]],  [[1, 2
   ]],  [[1, 3 ]],  [[0, 0 ]]]
```

5.21 throw_cards

Правило используется для того, чтобы перемешать карты в списке Cards. Переменные:

- Cards – переменная, хранящая список перемешанных карт;
- RowsCount – переменная, хранящая количество столбцов на поле;
- TotalCards – переменная, хранящая количество всех карт в игре.

В листинге 5.38 представлена реализация правила throw_cards

Листинг 5.38: реализация правила throw_cards

```
throw_cards(Cards, RowsCount, TotalCards, Field) :-
        throw_cards_internal(Cards, RowsCount, TotalCards, [], Field, 0).

throw_cards_internal([H | T], RowsCount, TotalCards, TempField, Field,
   Iteration) :-
        Iteration < TotalCards,
        cnt_mod(Iteration, RowsCount, Index),
        get_element(TempField, Index, Row),
        insert(Row, NewRow, H),
        replace(Index, TempField, NewRow, NewTempField),
        NextIteration is Iteration + 1,
        throw_cards_internal(T, RowsCount, TotalCards, NewTempField, Field,
   NextIteration), !.

throw_cards_internal(_, _, _, Field, Field, _).
```

5.22 generate_empty_free_cells

Правило создаёт пустую переменную для хранения свободных ячеек.
В листинге 5.39 представлена реализация правила generate_empty_free_cells

Листинг 5.39: реализация правила generate_empty_free_cells

```
generate_empty_free_cells(FreeCellsResult) :-
        set_value(FreeCellsResult, []).

generate_empty_free_cells_internal(NumberOfItems, TempArray, ResultArray,
   Iteration) :-
        Iteration < NumberOfItems,
        NSIteration is Iteration + 1,
        insert(TempArray, NSTempArray, []),
```

```
        generate_empty_free_cells_internal(NumberOfItems, NSTempArray,
    ResultArray, NSIteration), !.

generate_empty_free_cells_internal(NumberOfItems, ResultArray, ResultArray,
    NumberOfItems).
```

5.23 generate_empty_dom

Правило создаёт пустую переменную для хранения дома
В листинге 5.40 представлена реализация правила generate_empty_free_cells

Листинг 5.40: реализация правила generate_empty_free_cells

```
generate_empty_dom(NumberOfDomItems, DomResult) :-
        generate_empty_list_of_lists(NumberOfDomItems, DomResult).
```

5.24 generate_empty_list_of_lists

Правило позволяет создать переменную, хранящую список списков
Переменные:

- Field – переменная, хранящая набор карт, находящихся на поле;
- N – номер столбца;
- H – голова списка, то есть верхняя карта столбца.

В листинге 5.41 представлена реализация правила generate_empty_list_of_lists

Листинг 5.41: реализация правила generate_empty_list_of_lists

```
generate_empty_list_of_lists(NumberOfItems, ResultArray) :-
        generate_empty_free_cells_internal(NumberOfItems, [], ResultArray, 0).
```

5.25 get_top_card_from_nth_column

Правило позволяет получить значение верхней карты определенного столбца и длину данного столбца игрового поля
Переменные:

- Field – переменная, хранящая набор карт, находящихся на поле;
- N – номер столбца;
- H – голова списка, то есть верхняя карта столбца.

В листинге 5.42 представлена реализация правила get_top_card_from_nth_column

Листинг 5.42: реализация правила get_top_card_from_nth_column

```
get_top_card_from_nth_column(Field, N, H) :- get_element(Field, N, [H | _]).
```

В листинге 5.43 представлен пример работы правила get_top_card_from_nth_column

Листинг 5.43: пример работы правила get_top_card_from_nth_column

```
get_top_card_from_nth_column([[[0, 0], [2, 1]], [[2, 0], [3, 0]], [[1, 1], [3,
   1]], [[1, 0], [0, 1]]], 2, H)
H  =  [1, 1 ]
```

5.26 get_top_card_from_nth_column_with_length

Правило позволяет получить значение верхней карты определенного столбца и длину данного столбца игрового поля

Переменные:

- Field – переменная, хранящая набор карт, находящихся на поле;
- N – номер столбца;
- H – голова списка, то есть верхняя карта столбца;
- Len – длина столбца.

В листинге 5.44 представлена реализация правила get_top_card_from_nth_column_with_length

Листинг 5.44: реализация правила get_top_card_from_nth_column_with_length

```
get_top_card_from_nth_column_with_length(Field, N, H, Len) :-
    get_element_with_len(Field, N, [H | _], Len).
```

В листинге 5.45 представлена пример работы правила get_top_card_from_nth_column_with_length

Листинг 5.45: пример работы правила get_top_card_from_nth_column_with_length

```
get_top_card_from_nth_column_with_length([[[0, 0], [2, 1]], [[2, 0], [3, 0]],
   [[1, 1], [3, 1]], [[1, 0], [0, 1]]], 2, H, Len)
H  =  [1, 1 ],   Len   = 2
```

5.27 pop_first_element

Правило удаляет из списка первый элемент.
Переменные:

- Field – переменная, хранящая набор карт, находящихся на поле;
- N – номер столбца;
- H – голова списка, то есть верхняя карта столбца;
- Len – длина столбца.

В листинге 5.46 представлена реализация правила pop_first_element

Листинг 5.46: реализация правила pop_first_element

```
pop_first_element([_ | T], T) :- !.

pop_first_element([], []) :- !.
```

В листинге 5.47 представлен пример работы правила pop_first_element

Листинг 5.47: пример работы правила pop_first_element

```
pop_first_element([1,2,3],Res)
Res =  [2, 3 ]
```

5.28 remove_top_card_from_nth_column

Правило удаляет из списка первый элемент.
Переменные:

- Field – переменная, хранящая набор карт, находящихся на поле;
- N – номер столбца;
- H – голова списка, то есть верхняя карта столбца;
- Len – длина столбца.

В листинге 5.48 представлена реализация правила remove_top_card_from_nth_column

Листинг 5.48: реализация правила remove_top_card_from_nth_column

```
remove_top_card_from_nth_column(Field, N, FieldResult) :-
        get_element(Field, N, Column),
        pop_first_element(Column, ColumnResult),
        replace(N, Field, ColumnResult, FieldResult).
```

В листинге 5.49 представлен пример работы правила remove_top_card_from_nth_column

Листинг 5.49: пример работы правила remove_top_card_from_nth_column

```
remove_top_card_from_nth_column([[[0, 0], [2, 1]], [[2, 0], [3, 0]], [[1, 1],
    [3, 1]], [[1, 0], [0, 1]]], 2, Res)
Res =  [[[0, 0 ], [2, 1 ]], [[2, 0 ], [3, 0 ]], [[3, 1 ]], [[1, 0 ], [0,
    1 ]]]
```

5.29 avalialbe_to_push_card_to_free_cells

Правило проверяет можно ли положить карту в свободные ячейки
Переменные:

- FreeCellsCount – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- ReservedCellsCount – хранит количество занятых свободных ячеек.

В листинге 5.50 представлена реализация правила avalialbe_to_push_card_to_free_cells

Листинг 5.50: реализация правила avalialbe_to_push_card_to_free_cells

```
avalialbe_to_push_card_to_free_cells(ReservedFreeCellsCount, FreeCellsCount) :-
    ReservedFreeCellsCount < FreeCellsCount.
```

В листинге 5.51 представлен пример работы правила avalialbe_to_push_card_to_free_cells

Листинг 5.51: пример работы правила avalialbe_to_push_card_to_free_cells

```
avalialbe_to_push_card_to_free_cells(1, 3)
true
```

5.30 move_card_to_free_cell_if_avaliable

Правило используется для перемещения карты из поля в свободные ячейки. Сначала проверяется есть ли не занятые ячейки. Если есть, карта помещается в ячейку, удаляется из столбца, а количество занятых ячеек увеличивается.
Переменные:

- Card – переменная хранит значение перемещаемой карты;
- RowOfCard – переменная хранит длину столбца, наверху которой находится Card;

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- FreeCells – переменная хранит значения карт, находящихся в свободных ячейках;
- FreeCellsSize – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- ReservedCellsCount – хранит количество занятых свободных ячеек;
- FieldResult, FreeCellsResult, ReservedCellsCountResu – переменные, хранящие обновленные значения предыдущего состояния.

В листинге 5.52 представлена реализация правила move_card_to_free_cell_if_avaliable

Листинг 5.52: реализация правила move_card_to_free_cell_if_avaliable

```
move_card_to_free_cell_if_avaliable(Card, RowOfCard, Field, FreeCells,
   FreeCellsSize, ReservedCellsCount, FieldResult, FreeCellsResult,
   ReservedCellsCountResult) :-
        avalialbe_to_push_card_to_free_cells(ReservedCellsCount, FreeCellsSize)
   ,
        insert(FreeCells, FreeCellsResult, Card),
        remove_top_card_from_nth_column(Field, RowOfCard, FieldResult), !,
        ReservedCellsCountResult is ReservedCellsCount + 1.
```

В листинге 5.53 представлен пример работы правила move_card_to_free_cell_if_avaliable

Листинг 5.53: пример работы правила move_card_to_free_cell_if_avaliable

```
move_card_to_free_cell_if_avaliable([1,1],2,[[[0, 0], [2, 1]], [[2, 0], [3,
   0]], [[1, 1], [3, 1]], [[1, 0], [0, 1]]], [], 4, 0, FieldResult,
   FreeCellsResult, ReservedCellsCountResult)
FieldResult =  [[[0, 0 ],  [2, 1 ]],  [[2, 0 ],  [3, 0 ]],  [[3, 1 ]],  [[1,
    0 ],  [0, 1 ]]],
FreeCellsResult =  [[1, 1 ]],
ReservedCellsCountResult = 1
```

5.31 avaliable_to_push_card_to_dom_column

Правило проверяет можно ли переложить карту в дом

В листинге 5.54 представлена реализация правила avaliable_to_push_card_to_dom_column

Листинг 5.54: реализация правила avaliable_to_push_card_to_dom_column

```
avaliable_to_push_card_to_dom_column(Card, Column, MaxValue) :-
        MaxAvaliableValue is MaxValue - 1,
        avaliable_to_push_card_to_dom_column_internal(Card, Column,
   MaxAvaliableValue).
```

```
avaliable_to_push_card_to_dom_column_internal([MaxAvaliableValue, _], [],
   MaxAvaliableValue) :- !.

avaliable_to_push_card_to_dom_column_internal([0, Color], [[MaxAvaliableValue,
   Color] | []], MaxAvaliableValue) :- !.

avaliable_to_push_card_to_dom_column_internal([Value, Color], [[TopCardValue,
   Color] | _], _) :-
        ValueValidaesTolayOver is TopCardValue + 1,
        is_equal(Value, ValueValidaesTolayOver).
```

5.32 move_card_to_dom_if_avaliable

Правило используется для перемещения карты из поля в дом. Сначала проверяется, можно ли переложить карту в один из столбцов дома. Если можно, карта добавляется в новый столбец и удаляется из старого столбца поля. Иначе проверяется следующий столбец дома. Переменные:

- Card – переменная хранит значение перемещаемой карты;
- RowOfCard – переменная хранит длину столбца, наверху которого находится Card;
- Field – переменная, в которой хранится набор карт, находящихся на поле;
- Dom – переменная, в которой хранится набор карт, находящихся в доме;
- FieldResult, DomResult – переменные, хранящие обновленные значения предыдущего состояния;
- MaxValue - переменная, хранящая максимально количество карт одного цвета.

В листинге 5.55 представлена реализация правила move_card_to_dom_if_avaliable

Листинг 5.55: реализация правила move_card_to_dom_if_avaliable

```
move_card_to_dom_if_avaliable_internal(Card, Field, RowOfCard, Dom, [H | _],
   MaxValue, FieldResult, DomResult, Iteration) :-
        avaliable_to_push_card_to_dom_column(Card, H, MaxValue),
        insert_to_head(H, Card, ColumnRes),
        replace(Iteration, Dom, ColumnRes, DomResult),
        remove_top_card_from_nth_column(Field, RowOfCard, FieldResult), !.

move_card_to_dom_if_avaliable_internal(Card, Field, RowOfCard, Dom, [_ | T],
   MaxValue, FieldResult, DomResult, Iteration) :-
        NSIteration is Iteration + 1,
        move_card_to_dom_if_avaliable_internal(Card, Field, RowOfCard, Dom, T,
   MaxValue, FieldResult, DomResult, NSIteration), !.
```

```
move_card_to_dom_if_avaliable_internal(_, Field, _, Dom, [], _, Field, Dom, _)
    :- false.
```

5.33 move_card_to_field_from_free_cell_if_avaliable

Правило используется для перемещения карты из свободных ячеек в поле. Берётся карта из свободных ячеек и проверяется возможность переместить её

на поле. Если условие соблюдено, карта удаляется из ячейки и добавляется в столбец на поле. Иначе проверяется следующая карта, лежащая в свободных ячейках

Переменные:

- Card – переменная хранит значение перемещаемой карты;
- RowOfCard – переменная хранит длину столбца, наверху которого находится Card;
- NumberOfColumns – хранит количество столбцов игрового поля Field;
- Field – переменная, в которой хранится набор карт, находящихся на поле;
- FreeCells – переменная хранит значения карт, находящихся в свободных ячейках;
- FreeCellsSize – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- ReservedCellsCount – хранит количество занятых свободных ячеек;
- FieldResult, FreeCellsResult, ReservedCellsCountResu – переменные, хранящие обновленные значения предыдущего состояния;

В листинге 5.56 представлена реализация правила move_card_to_field_from_free_cell_if_avaliable

Листинг 5.56: реализация правила move_card_to_field_from_free_cell_if_avaliable

```
move_card_to_field_from_free_cell_if_avaliable(Card, Field, FreeCells,
    NumberOfColumns, ReservedFreeCellsCount, FieldResult, FreeCellsResult,
    ReservedFreeCellsCountResult) :-
        move_card_to_field_some_column_if_avalialbe_card(Field, Card,
    NumberOfColumns, FieldResult),
        delete(FreeCells, Card, FreeCellsResult),
        ReservedFreeCellsCountResult is ReservedFreeCellsCount - 1.
```

5.34 move_card_to_dom_from_free_cell_if_avaliable

Правило используется для перемещения карты из свободных ячеек в дом. Проверяется возможность положить карту в дом. Если можно, карта добавляется в нужный столбец дома и удаляется из свободных ячеек. Иначе проверяется следующая карата из свободных ячеек.

Переменные:

- Card – переменная хранит значение перемещаемой карты;
- FreeCells – переменная хранит значения карт, находящихся в свободных ячейках;
- FreeCellsSize – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- Dom – переменная, в которой хранится набор карт, находящихся в доме;
- MaxValue - переменная, хранящая максимально количество карт одного цвета;
- ReservedCellsCount – хранит количество занятых свободных ячеек;
- DomResult, FreeCellsResult, ReservedCellsCountResu – переменные, хранящие обновленные значения предыдущего состояния;

В листинге 5.57 представлена реализация правила move_card_to_dom_from_free_cell_if_avaliable

Листинг 5.57: реализация правила move_card_to_dom_from_free_cell_if_avaliable

```
        move_card_to_dom_from_free_cell_if_avaliable_internal(Card, FreeCells,
    Dom, Dom, MaxValue, FreeCellsResult, DomResult, ReservedFreeCellsCount,
    ReservedFreeCellsCountResult, 0).

move_card_to_dom_from_free_cell_if_avaliable_internal(Card, FreeCells, Dom, [H
    | _], MaxValue, FreeCellsResult, DomResult, ReservedFreeCellsCount,
    ReservedFreeCellsCountResult, Iteration) :-
        avaliable_to_push_card_to_dom_column(Card, H, MaxValue),
        insert_to_head(H, Card, ColumnRes),
        replace(Iteration, Dom, ColumnRes, DomResult),
        delete(FreeCells, Card, FreeCellsResult),
        ReservedFreeCellsCountResult is ReservedFreeCellsCount - 1, !.

move_card_to_dom_from_free_cell_if_avaliable_internal(Card, FreeCells, Dom, [_
    | T], MaxValue, FreeCellsResult, DomResult, ReservedFreeCellsCount,
    ReservedFreeCellsCountResult, Iteration) :-
        NSIteration is Iteration + 1,
        move_card_to_dom_from_free_cell_if_avaliable_internal(Card, FreeCells,
    Dom, T, MaxValue, FreeCellsResult, DomResult, ReservedFreeCellsCount,
    ReservedFreeCellsCountResult, NSIteration), !.
```

```
move_card_to_dom_from_free_cell_if_avaliable_internal(_, Field, Dom, [], _,
    Field, Dom, _, _) :- false.
```

В листинге 5.58 представлен пример работы правила move_card_to_dom_from_free_cell_if_avaliable

Листинг 5.58: пример работы правила move_card_to_dom_from_free_cell_if_avaliable

```
move_card_to_field_from_free_cell_if_avaliable([2,0],
                                                [[[0,0],[2,1]],
[[3,1]],
[[1,1],[3,0]],
[[1,0],[0,1]]],
[[2,0]], 4, 1,FieldResult, FreeCellsResult, ReservedFreeCellsCountResult)
FieldResult =  [[[0, 0 ],  [2, 1 ]],  [[2, 0 ],  [3, 1 ]],  [[1, 1 ],  [3, 0
    ]],  [[1, 0 ],  [0, 1 ]]],
FreeCellsResult =  [],
ReservedFreeCellsCountResult = 0
```

5.35 avaliable_to_push_card_to_field_column

Правило проверят можно ли переместить карту между двумя столбцами. Сначала определяется значение столбца игрового поля. Затем получается значение карты, находящейся на верху своего столбца. Дальше проверяется можно ли положить карту на верх выбранного столбца. Проверяются цвета карт (они должны быть разными) и старшинство карты, которую перекладываем (оно должно быть на единицу меньше старшинства карты наверху столбца). Также, если столбцы, между которыми происходит перемещение, пустые, не считая карты, которая перемещается, ход не происходит. Это правило используется для проверки перемещения между столбцами игрового поля.

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- Card – переменная хранит значение перемещаемой карты;
- CardN – переменная хранит номер столбца, из которого надо взять верхнюю карту;
- ColumnN – переменная хранит номер столбца, верхняя карта которого сравнивается с перемещаемой картой;
- CardColumnLen – переменная хранит длину столбца, где хранится перемещаемая карта;

- ColumnLen – переменная хранит длину столбца, в который пробуем переместить карту.

В листинге 5.59 представлена реализация правила avaliable_to_push_card_to_field_column

Листинг 5.59: реализация правила avaliable_to_push_card_to_field_column

```
avaliable_to_push_card_to_field_column(_, N, N) :- !, false.

avaliable_to_push_card_to_field_column(Field, CardN, ColumnN) :-
        get_element(Field, ColumnN, Column),
        length(Column, ColumnLen),
        get_top_card_from_nth_column_with_length(Field, CardN, Card,
   CardColumnLen), !,
        avaliable_to_push_card_to_field_column_internal_with_len(Card, Column,
   CardColumnLen, ColumnLen).

avaliable_to_push_card_to_field_column_card(Field, Card, ColumnN) :-
        get_element(Field, ColumnN, Column), !,
        avaliable_to_push_card_to_field_column_internal(Card, Column).

avaliable_to_push_card_to_field_column_internal(_, []) :-  !.

avaliable_to_push_card_to_field_column_internal([Value, Color], [[TopCardValue,
    TopCardColor] | _]) :-
        is_not_equal(Color, TopCardColor),
        ValueValidaesTolayOver is TopCardValue - 1,
        is_equal(Value, ValueValidaesTolayOver).

avaliable_to_push_card_to_field_column_internal_with_len([Value, Color], [[
   TopCardValue, TopCardColor] | _], CardColumnLen, ColumnLen) :-
        columns_empty(CardColumnLen, ColumnLen),
        is_not_equal(Color, TopCardColor),
        ValueValidaesTolayOver is TopCardValue - 1,
        is_equal(Value, ValueValidaesTolayOver).
```

В листинге 5.60 представлен пример работы правила avaliable_to_push_card_to_field_column

Листинг 5.60: пример работы правила avaliable_to_push_card_to_field_column

```
avaliable_to_push_card_to_field_column([[[2,0],[2,1]],
                                        [[3,1]],
                                        [[1,1],[3,0]],
                                        [[1,0],[0,1]]], 0, 1)
true
```

5.36 avaliable_to_push_card_to_field_column_card

Правило похожее на предыдущее. Правило проверяет можно ли переложить карту из свободной ячейки в столбец игрового поля. Сначала получается значение столбца, затем сравниваются карта из ячейки и верхняя карта столбца. Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- Card – значение перемещаемой карты;
- ColumnN – переменная хранит номер столбца игрового поля, верхняя карта которого сравнивается с перемещаемой картой;
- Column – столбец игрового поля;

В листинге 5.61 представлена реализация правила avaliable_to_push_card_to_field_column_card

Листинг 5.61: реализация правила avaliable_to_push_card_to_field_column_card

```
avaliable_to_push_card_to_field_column_card(Field, Card, ColumnN) :-
        get_element(Field, ColumnN, Column), !,
        avaliable_to_push_card_to_field_column_internal(Card, Column).
```

В листинге 5.62 представлен пример работы правила avaliable_to_push_card_to_field_column_card

Листинг 5.62: пример работы правила avaliable_to_push_card_to_field_column_card

```
avaliable_to_push_card_to_field_column_card([[[4,0],[2,1]],
                                             [[3,1]],
                                             [[1,1],[3,0]],
                                             [[1,0],[0,1]]], [2,0], 1)
true
```

5.37 move_card_to_field_column

Правило используется для перемещения карты в столбец игрового поля Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- N – номер столбца, из которого берётся карта;
- Card – значение перемещаемой карты;
- ColumnN – номер столбца, в который перемещаем карту;
- FieldResult – обновленный набор карт, находящихся на поле;
- UpdateColumn – обновленный столбец игрового поля.

В листинге 5.63 представлена реализация правила move_card_to_field_column

Листинг 5.63: реализация правила move_card_to_field_column

```
        get_top_card_from_nth_column(Field, N, Card),
        remove_top_card_from_nth_column(Field, N, TempFieldResult),
        get_element(TempFieldResult, ColumnN, Column),
           insert_to_head(Column, Card, UpdatedColum),
        replace(ColumnN, TempFieldResult, UpdatedColum, FieldResult), !.
```

В листинге 5.64 представлен пример работы правила move_card_to_field_column

Листинг 5.64: реализация правила move_card_to_field_column

```
move_card_to_field_column([[[2,0],[2,1]],
                                        [[3,1]],
                                        [[1,1],[3,0]],
                                        [[1,0],[0,1]]], 0, 1, FieldResult)
FieldResult =  [[[2, 1 ]],  [[2, 0 ],  [3, 1 ]],  [[1, 1 ],  [3, 0 ]],  [[1,
    0 ],  [0, 1 ]]]
```

5.38 add_card_to_field_column

Правило используется для добавление уже известной карты в столбец игрового поля

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- Card – значение перемещаемой карты;
- ColumnN – номер столбца, в который перемещаем карту;
- FieldResult – обновленный набор карт, находящихся на поле;
- ColumnResult – обновленный столбец игрового поля.

В листинге 5.65 представлена реализация правила add_card_to_field_column

Листинг 5.65: реализация правила add_card_to_field_column

```
        get_element(Field, ColumnN, Column),
        insert_to_head(Column, Card, ColumnResult),
        replace(ColumnN, Field, ColumnResult, FieldResult), !.
```

В листинге 5.66 представлен пример работы правила add_card_to_field_column

Листинг 5.66: пример работы правила add_card_to_field_column

```
add_card_to_field_column([[[2,0],[2,1]]], [1,1], 0, FieldResult)
FieldResult =  [[[1, 1 ],  [2, 0 ],  [2, 1 ]]]
```

5.39 find_avaliable_column_to_push

Правило используется для поиска столбца игрового поля, в который можно переместить карту, находящуюся в другом столбце.

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- NumberOfColumns – хранит количество столбцов поля Field;
- N – номер столбца, из которого берётся карта;
- FoundedColumn – найденный столбец, в который можно переложить карту.

В листинге 5.67 представлена реализация правила find_avaliable_column_to_push

Листинг 5.67: реализация правила find_avaliable_column_to_push

```
        find_avaliable_column_to_push_internal(Field, NumberOfColumns, N,
    FoundedColumn, 0).

find_avaliable_column_to_push_internal(_, NumberOfColumns, _, _,
    NumberOfColumns) :-
        !, false.

find_avaliable_column_to_push_internal(Field, _, N, Iteration, Iteration) :-
        avaliable_to_push_card_to_field_column(Field, N, Iteration).

find_avaliable_column_to_push_internal(Field, NumberOfColumns, N, FoundedItem,
    Iteration) :-
        NSIteration is Iteration + 1,
        find_avaliable_column_to_push_internal(Field, NumberOfColumns, N,
    FoundedItem, NSIteration).
```

В листинге 5.68 представлен пример работы правила find_avaliable_column_to_push

Листинг 5.68: реализация правила find_avaliable_column_to_push

```
find_avaliable_column_to_push([[[2,0],[2,1]], [[3,1]], [[1,1],[3,0]],
    [[1,0],[0,1]]], 4, 0, FoundedColumn)
FoundedColumn = 1
```

5.40 find_avaliable_column_to_push_card

Правило используется для поиска столбца игрового поля, в который можно переместить карту, значение которой уже известно.

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;

- NumberOfColumns – хранит количество столбцов поля Field;
- Card – значение перемещаемой карты;
- FoundedColumn – найденный столбец, в который можно переложить карту.

В листинге 5.69 представлена реализация правила find_avaliable_column_to_push_card

Листинг 5.69: реализация правила find_avaliable_column_to_push_card

```
        find_avaliable_column_to_push_internal_card(Field, NumberOfColumns,
    Card, FoundedColumn, 0).

find_avaliable_column_to_push_internal_card(_, NumberOfColumns, _, _,
    NumberOfColumns) :-
        !, false.

find_avaliable_column_to_push_internal_card(Field, _, Card, Iteration,
    Iteration) :-
        avaliable_to_push_card_to_field_column_card(Field, Card, Iteration).

find_avaliable_column_to_push_internal_card(Field, NumberOfColumns, Card,
    FoundedItem, Iteration) :-
        NSIteration is Iteration + 1,
        find_avaliable_column_to_push_internal_card(Field, NumberOfColumns,
    Card, FoundedItem, NSIteration).
```

В листинге 5.70 представлен пример работы правила find_avaliable_column_to_push_card

Листинг 5.70: реализация правила find_avaliable_column_to_push_card

```
find_avaliable_column_to_push_card([[[2,1]], [[3,1]], [[1,1],[3,0]],
    [[1,0],[0,1]]], 4, [2,0],FoundedColumn)
FoundedColumn = 1
```

5.41 move_card_to_field_some_column_if_avalialbe

Правило используется для перемещения карты между двумя столбцами игрового поля. В правиле find_avaliable_column_to_push определяется столбец, в который можно переложить карту. Затем с помощью правила move_card_to_field_column происходит перенос карты из одного столбца в другой.

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- NumberOfColumns – хранит количество столбцов поля Field;
- N – номер столбца, из которого берётся карта;

- FieldRes – обновлённый набор карт, находящихся на поле.

В листинге 5.71 представлена реализация правила move_card_to_field_some_column_if_avalialbe

Листинг 5.71: реализация правила move_card_to_field_some_column_if_avalialbe

```
move_card_to_field_some_column_if_avalialbe(Field, NumberOfColumns, N,
   FieldResult) :-
        find_avaliable_column_to_push(Field, NumberOfColumns, N, ColumnNumber),
    !,
        move_card_to_field_column(Field, N, ColumnNumber, FieldResult).
```

В листинге 5.72 представлен пример работы правила move_card_to_field_some_column_if_avalialbe

Листинг 5.72: реализация правила move_card_to_field_some_column_if_avalialbe

```
move_card_to_field_some_column_if_avalialbe([[[2,0],[2,1]], [[3,1]],
   [[1,1],[3,0]], [[1,0],[0,1]]], 4, 0,FieldResult)
FieldResult =  [[[2, 1 ]],  [[2, 0 ],  [3, 1 ]],  [[1, 1 ],  [3, 0 ]],  [[1,
    0 ],  [0, 1 ]]]
```

5.42 move_card_to_field_some_column_if_avalialbe_card

Правило используется для перемещения заранее известной карты в столбец игрового поля. В правиле find_avaliable_column_to_push_card определяется столбец, в который можно переложить карту. Затем с помощью правила add_card_to_field_column происходит добавление карты в столбец

Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- NumberOfColumns – хранит количество столбцов поля Field;
- Card – значение перемещаемой карты;
- FieldResult – обновлённый набор карт, находящихся на поле.

В листинге 5.73 представлена реализация правила move_card_to_field_some_column_if_avalialbe_card

Листинг 5.73: реализация правила move_card_to_field_some_column_if_avalialbe_card

```
move_card_to_field_some_column_if_avalialbe(Field, NumberOfColumns, N,
   FieldResult) :-
        find_avaliable_column_to_push(Field, NumberOfColumns, N, ColumnNumber),
    !,
        move_card_to_field_column(Field, N, ColumnNumber, FieldResult).
```

В листинге 5.74 представлен пример работы правила move_card_to_field_some_

Листинг 5.74: реализация правила
move_card_to_field_some_column_if_avalialbe_card

```
move_card_to_field_some_column_if_avalialbe_card([[[2,1]], [[3,1]],
   [[1,1],[3,0]], [[1,0],[0,1]]], [2,0], 4, FieldResult)
FieldResult =  [[[2, 1 ]],  [[2, 0 ],  [3, 1 ]],  [[1, 1 ],  [3, 0 ]],  [[1,
    0 ],  [0, 1 ]]]
```

5.43 solve_mock

Правило используется для альтернативного старта программы. Вместо генерации поле задаётся через правило set_value, а значение переменных меняется в коде. Это правило удобно использовать для тестирования на конкретных случаях.

Переменные:

- MaxValue – хранит количество карт каждого цвета;
- ColorsCount – хранит количество цветов карт;
- Solution – переменная, которая в результате работы программы получает набор состояний, являющийся последовательным решением задачи;
- States – переменная, для хранения набора состояний;
- Dom – переменная, в которой хранятся значения карт, находящихся в доме;
- FreeCells – переменная, в которой хранятся значения карт, находящихся в свободных ячейках;
- Field – переменная, в которой хранится набор карт, находящихся на поле;
- FreeCellsSize – хранит размер переменной FreeCells, то есть количество свободных ячеек в игре;
- ReservedCellsCount – хранит количество занятых свободных ячеек;
- NumberOfColumns – хранит количество столбцов игрового поля Field;

В листинге 5.75 представлена реализация правила solve_mock

Листинг 5.75: реализация правила solve_mock

```
solve_mock(Solution) :-
        MaxValue is 10,
        ColorsCount is 1,
        FreeCellsSize is ColorsCount,
        ReservedCellsCount is 0,
        NumberOfColumns is ColorsCount * 2,
```

```
        set_value(Field,
        [[[3, 0], [2, 0], [1, 0], [9, 0], [4, 0]], [[8, 0], [7, 0], [5, 0], [0,
    0], [6, 0]]]
        ),
        generate_empty_free_cells(FreeCells),
        generate_empty_dom(ColorsCount, Dom),
        stm_init(State),
        stm_append_value_if_not_in_stm(State, [Dom, FreeCells, Field], States),
        make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom,
   FreeCellsSize, ReservedCellsCount, States, Solution).
```

5.44 create_solution

Это правило используется для альтернативного старта программы. Отличие в том, что значение переменной Field известно заранее. Это удобно, если нужно протестировать программу на определённом примере

Переменные:

- MaxValue – хранит количество карт каждого цвета;
- NumberOfColors – хранит количество цветов карт;
- Solution – переменная, которая в результате работы программы получает набор состояний, являющийся последовательным решением задачи;
- States – переменная, для хранения набора состояний;
- Dom – переменная, в которой хранятся значения карт, находящихся в доме;
- FreeCells – переменная, в которой хранятся значения карт, находящихся в свободных ячейках;
- Field – переменная, в которой хранится набор карт, находящихся на поле;
- NumberOfColumns – хранит количество столбцов игрового поля Field.

В листинге 5.76 представлена реализация правила create_solution

Листинг 5.76: реализация правила create_solution

```
create_solution(Field, MaxValue, NumberOfColors, Solution) :-
        generate_empty_free_cells(FreeCells),
        NumberOfColumns is NumberOfColors * 2,
        generate_empty_dom(NumberOfColumns, Dom),
        stm_init(State),
        stm_append_value_if_not_in_stm(State, [Dom, FreeCells, Field], States),
        make_move(Field, MaxValue, NumberOfColumns, FreeCells, Dom,
   NumberOfColors, 0, States, Solution).
```

5.45 get_top_card

Правило используется для получения верхней карты столбца.
Переменные:

- Field – переменная, в которой хранится набор карт, находящихся на поле;
- TopCard – верхняя карта;
- CardRow – номер столбца, где находится карта.

В листинге 5.77 представлена реализация правила get_top_card

Листинг 5.77: реализация правила get_top_card

```
get_top_card(Field, TopCard, CardRow) :-
        get_top_card_internal(Field, TopCard, CardRow, 0).

get_top_card_internal([[TopCard | _] | _], TopCard, Itaration, Itaration).

get_top_card_internal([ _ | T], TopCard, CardRow, Itaration) :-
        NSIteration is Itaration + 1,
        get_top_card_internal(T, TopCard, CardRow, NSIteration).
```

В листинге 5.78 представлен пример работы правила get_top_card

Листинг 5.78: пример работы правила get_top_card

```
get_top_card([ [[0, 0], [2, 1]], [[2, 0], [3, 0]], [[1, 1], [3, 1]], [[1, 0],
   [0, 1]] ], TopCard, CardRow)
CardRow = 0 ,   TopCard   =  [0, 0 ]
Next
CardRow = 1 ,   TopCard   =  [2, 0 ]
```

5.46 process_free_cell_card

Правило используется для получения карты из набора свободных ячеек
Переменные:

- FreeCells – переменная, в которой хранятся значения карт, находящихся в свободных ячейках
- Card – значение найденной карты

В листинге 5.79 представлена реализация правила process_free_cell_card

Листинг 5.79: реализация правила process_free_cell_card

```
process_free_cell_card(FreeCells, Card) :-
        process_free_cell_card_internal(FreeCells, Card).

process_free_cell_card_internal([H | _], H).
```

```
process_free_cell_card_internal([_ | T], Card) :-
        process_free_cell_card_internal(T, Card).
```

В листинге 5.80 представлен работы правила правила process_free_cell_card

Листинг 5.80: пример работы правила process_free_cell_card

```
process_free_cell_card([[2,0],[1,1],[0,0]], Card)
Card =  [2, 0 ]
Next
Card =  [1, 1 ]
Next
Card =  [0, 0 ]
```

5.47 field_is_empty

Правило используется для проверки поля на наличие карт
В листинге 5.81 представлена реализация правила field_is_empty

Листинг 5.81: реализация правила field_is_empty

```
field_is_empty([[] | T]) :-
        field_is_empty(T), !.

field_is_empty([[]]).
```

В листинге 5.82 представлена реализация правила field_is_empty

Листинг 5.82: пример работы правила field_is_empty

```
field_is_empty([[],[],[]])
true
field_is_empty([[],[2,1],[]])
false
```

Заключение

Задача проекта успешно решена. Было разработано программное обеспечение, предоставляющее возможность получать решение пасьяна «Свободная ячейка». Решение получено на языке логического программирования Prolog. Для визуализации полученного решения был разработан GUI на языке программирования Python.

Литература

[1] Shlomi Fish's Homepage. [Электронный ресурс] – Режим доступа: https://www.shlomifish.org, свободный – (дата обращения: 20.08.21)

[2] SWI-Prolog documentation. [Электронный ресурс] – Режим доступа: https://www.swi-prolog.org/pldoc/index.html, свободный – (дата обращения: 25.08.21)

[3] Pygame Front Page. [Электронный ресурс] – Режим доступа: https://www.pygame.org/docs/, свободный – (дата обращения: 15.09.21)

[4] yuce/pyswip [Электронный ресурс] – Режим доступа: https://github.com/yuce/pyswip, свободный – (дата обращения: 02.09.21)