

Институт ИТКН

Кафедра инженерной кибернетики

Направление подготовки: 09.04.03 Прикладная информатика

Квалификация (степень): магистр

Группа: МПИ-20-4-2


ОТЧЕТ

ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

на тему: «Математическое и программное обеспечение для решения казуально-логических игр с использованием технологий самообучения»

III семестр 2021 – 2022 у. г.

Студент:

 /Новицкий Д. А./

подпись Фамилия И.О.

Руководитель НИР:

 /доцент, к.т.н. Кожаринов А. С./

подпись должность, уч. степ. Фамилия И.О.

Оценка:

Дата защиты:

27.12.21

Утвердил:

Председатель комиссии

_____/_____

подпись

Фамилия И.О.

Москва 2021

Оглавление

СПИСОК ИСПОЛЬЗУЕМЫХ ОСНОВНЫХ СОКРАЩЕНИЙ	4
ВВЕДЕНИЕ	5
Цель работы	5
1 АНАЛИТИЧЕСКИЙ ОБЗОР	6
1.1 Обзор логических задач	6
1.1.1 Виды головоломок.....	7
1.1.2 Выбор логической задачи для самообучающейся системы	8
1.2 Алгоритмы построения самообучающихся систем	9
1.2.1 Гибридные модели анализа ситуаций	9
1.2.2 Разработанный метод построения самообучающейся системы.....	11
1.2.3 Анализ алгоритмов построения самообучающихся систем.....	15
2 СПЕЦИАЛЬНАЯ ЧАСТЬ	17
2.2 Содержательная постановка задачи.....	18
2.3 Математическая постановка задачи	19
2.4 Описание алгоритма.....	23
2.4.1 Описание метода 1.....	23
2.4.2 Описание метода 2.....	23
2.4.3 Описание метода 3.....	23
2.4.4 Описание применения схем.....	23
2.5 Теоретическое доказательство применимости разработанного алгоритма для нахождения решения любого поля игры «Сапёр» при заданных ограничениях	23
2.6 Описание программной реализации	23
2.7 Результаты работы программы	23
ВЫВОДЫ	24
ТЕЗАУРУС.....	25
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	27
ПРИЛОЖЕНИЕ А. Правила игры «Minesweeper»/ «Сапёр»	28

СПИСОК ИСПОЛЬЗУЕМЫХ ОСНОВНЫХ СОКРАЩЕНИЙ

- ИНС – искусственные нейронные сети;
- КЛИ – казуально-логическая игра.

ВВЕДЕНИЕ

В последние десятилетия в научно-техническом обществе большое внимание уделяется развитию такой отрасли, как машинное обучение. Оно применяется во многих сферах деятельности людей, например, при поиске информации в интернете, при подборе музыки по предпочтениям, при доступе к банковским данным с применением биометрии, и это далеко не полный список, где используются методы машинного обучения. На данный момент эти методы демонстрируют высокую эффективность и точность, однако, и у методов машинного обучения есть свои недостатки. Так, например, известны случаи, когда в Китае житель смог разблокировать чужой телефон с использованием своих биометрических данных. Но, пожалуй, самая главная проблема машинного обучения, которая относится к ИНС – это невозможность определения, почему ИНС обучилась именно таким образом? Эта проблема не позволяет ИНС эффективно решать задачи, преимущественно связанные с логической обработкой данных. В связи с этим, возникает потребность в создании самообучающихся систем, не связанных с ИНС для эффективного решения задач, связанных с логической обработкой данных.

В данной работе представлены алгоритмы для построения самообучающихся систем, которые позволят решать определённый набор логических задач, произведен анализ данных алгоритмов и произведена их оценка, выбран набор заданий, которые данная самообучающаяся система будет способна решать, а также описаны цели, задачи и используемый математический аппарат.

Цель работы

Целью данной работы является анализ методов построения самообучающихся систем (в том числе самостоятельно разработанных), их оценка, построение самообучающейся системы с дальнейшим тестированием системы и проверкой эффективности обучения на реальных задачах.

1 АНАЛИТИЧЕСКИЙ ОБЗОР

1.1 Обзор логических задач

Всего существует огромное множество логических задач и важно определить круг задач, которые будут решаться разрабатываемой самообучающейся системой. Для того, чтобы выбрать круг логических задач, необходимо определить критерии, по которым будет происходить их выбор. Это такие критерии, как:

- сложность;
- формализуемость;
- наличие однозначного решения;

Поясним необходимость данных критериев. Сложность логической задачи – это основополагающий критерий выбора. Хотя данный критерий можно считать субъективным, поскольку нет единого стандарта оценки сложности той или иной логической задачи, в данном случае под сложностью будем подразумевать, количество входных данных, уровень однотипности логических операций и «объём» логических операций, необходимых для решения задачи. Если задача будет сложной, то и самообучающийся алгоритм также должен быть сложным, а, поскольку в данной работе предполагается разработка самообучающейся системы не трудных логических задач, в приоритете будут задачи с низким и средним уровнями сложности.

От уровня формализуемости зависит сложность представления логической задачи в математических терминах, а, чем выше сложность представления логической задачи в математических терминах, тем выше вероятность запутаться в данных, установить зависимости между ними, не учесть тот или иной элемент задачи. Таким образом, будем также ориентироваться на задачи с низким и средним уровнями формализуемости.

Наличие однозначного решения также важно при выборе логической задачи для самообучающейся системы. Задачи, имеющие однозначные решения, проще решать, поскольку такие задачи могут решаться по шаблону, которому самообучающейся системе необходимо научиться. Задачи, не имеющие однозначного решения, хоть и могут также решаться по шаблону, но такой метод не всегда может быть эффективным, поскольку в некоторых ситуациях необходим творческий подход к решению задачи, которому пока обучить машину не получилось. В данной работе будем отдавать предпочтение логическим задачам, которые имеют однозначное решение.

Головоломки – это распространённые и интересные логические задачи, решение которых зависит не от специальных знаний и творческих умений, а от сообразительности,

а, говоря на языке машины, зависит от алгоритма, определяющие порядок применения логических операций к тем или иным элементам задачи. Рассмотрим, какие бывают виды головоломок.

1.1.1 Виды головоломок

Ещё никто не придумал общей классификации для задач на логику, однако, исходя из их смысла, можно выделить четыре категории [1]:

- головоломки с предметами;
- механические головоломки;
- печатные головоломки;
- устные головоломки;
- компьютерные игры-головоломки.

Рассмотрим подробнее каждую из этих категорий.

1.1.1.1 Головоломки с предметами

Не обязательно покупать головоломку, порой бывает достаточно предметов, которые имеются в каждом доме. Одной из таких, которая обрела невероятную популярность, является головоломка со спичками. Её суть сводится к тому, что нужно переставить определённое количество спичек, чтобы получилась другая фигура. Ещё одна разновидность: переставить спички так, чтобы вышло верное равенство (речь идёт о цифрах). Наподобие этого есть головоломка с монетками.

1.1.1.2 Механические головоломки

Всемирно известный пример – кубик Рубика, автор которого также придумал «змейку». Также здесь можно вспомнить любимую игру детей – пятнашки. Все эти головоломки объединяет одно – это предметы, которые созданы для решения поставленных задач. Описанные выше игры найдутся далеко не в каждом доме, но есть и такая головоломка, в которую играли, пожалуй, все – это пазлы.

1.1.1.3 Печатные головоломки

К этому виду относятся головоломки, которые напечатаны на бумаге, а для их решения понадобится ручка. Примерами такой зарядки для ума являются:

- сканворды;
- кроссворды;
- ребусы;
- японские кроссворды;
- sudoku и другие.

1.1.1.4 Устные головоломки

К данному типу игр относятся загадки, игра в «да или нет», а также шарады.

1.1.1.5 Компьютерные игры-головоломки

К этой категории головоломок можно отнести наиболее известные игры, выпущенные как встроенные приложения в операционную систему Windows ещё в 90-е годы:

- сапёр;
- пинбол;
- пасьянс;
- червы;
- косынка;
- солитер;
- маджонг и другие.

1.1.2 Выбор логической задачи для самообучающейся системы

Итак, из большого количества логических игр выберем ту, для которой будем разрабатывать самообучающуюся систему.

Головоломка со спичками имеет достаточно узкий круг возможных задач, притом, данная задача может решаться с помощью полного перебора вариантов решения, поэтому данная задача не подходит.

Кубик Рубика является достаточно сложной задачей, хотя и достаточно популярной с имеющимся алгоритмом решения. Основная проблема при решении данной задачи – возможность визуализации процесса решения, поэтому данная задача также не подходит.

Сканворды, кроссворды и ребусы основаны на поиске информации по заданным данным, а также на игре слов, и при их решении практически не используются логические элементы, а sudoku уже подходит по критериям, поскольку данная логическая задача имеет средний уровень сложности, хорошо формализуема, а также имеет однозначное решение.

Карточные игры, наподобие игр «Пасьянс», «Червы», «Косынка», «Солитер», а также «Маджонг» не всегда могут иметь однозначное решение, «Пинбол» достаточно трудно формализуемая задача, а «Сапёр» подходит по рассматриваемым критериям.

Таким образом, были выбраны две логические игры для построения самообучающейся системы: sudoku и сапёр.

1.2 Алгоритмы построения самообучающихся систем

Рассмотрим теперь существующие алгоритмы построения самообучающихся систем.

1.2.1 Гибридные модели анализа ситуаций

Наиболее частым подходом к построению современных интеллектуальных систем является использование гибридных моделей анализа ситуаций [2].

Рассмотрим обучаемую модель игры в составе следующих блоков:

1. Создание поля. Создание игрового поля состоит из двух этапов:

- с помощью функции генерации случайных чисел размещается заданное число единиц - «мин» по полю заданного размера в матрице **A**;
- для каждой клетки матрицы **A** вычисляется сумма окружающих ее «мин», значение записывается в соответствующую ячейку матрицы **B** того же размера.

При решении задачи «сапера» пользователь или программа обращаются к матрице **B** во время открытия неизвестных ячеек; с помощью матрицы **A** проверяется количество оставшихся «мин» и условия проигрыша и победы. В матрице **C** того же размера ведется учет вероятностей нахождения «мин» в той или иной клетке. Матрица **D** содержит описание текущей видимой ситуации. В начальный момент все $d_{i,j} = 10$, т. е. не опознаны. Для клеток с «миной» будем использовать маркер $d_{i,j} = 9$, открытые клетки содержат маркеры от $d_{i,j} = 0$ («мин» рядом нет) до $d_{i,j} = 8$, (вокруг только «мины»).

2. Модуль обработки жесткой логики содержит следующие правила:

- если количество «мин» вокруг данной клетки соответствует ее числу, можно открыть все остальные ячейки вокруг нее, т. е. вероятность нахождения «мины» устанавливается равной 0;
- если количество неизвестных «мин» вокруг данной клетки равно числу свободных клеток вокруг нее, то можно поставить там «мины», т. е. вероятность нахождения «мины» устанавливается равной 1;

3. Модуль принятия решения содержит следующие правила:

- при неоднозначной расстановке «мин» следует выбрать наиболее вероятные точки путем составления матрицы вероятных положений с учетом обучения и расположения открытых клеток;
- при равнозначном выборе в режиме обучения запросить помощь пользователя. Описать решение пользователя в виде модели размера $K(n)_{5 \times 5}$ извлеченной из матрицы **D**. На основании его решений изменить параметры вероятности соответствующей клетки и проверить качество принятого решения: «удача\ошибка». Если решение было удачным, то вероятность отсутствия «мины» для клеток, соответствующих области типа $K(n)_{5 \times 5}$ увеличить, иначе следует ее уменьшить. В начальный момент считаем, что $P(K(n)_{5 \times 5} - \text{"мина"}) = 1$, т. е. для всех образцов «мины» есть;
- в режиме игры при равнозначном выборе полагаться на выбор. На основании получившегося результата изменить параметры вероятности;

4. Модуль изменения параметров вероятности. Работа с матрицей вероятностей **C** включает в себя следующие правила:

- свободные закрытые клетки имеют собственную вероятность нулевого уровня, являющуюся суммой вероятностей нахождения «мины» около открытых ячеек. Соответствующие слагаемые вычисляются как отношение количества недостающих «мин» к количеству свободных закрытых клеток вокруг открытых ячеек, расположенных вокруг текущей клетки;
- изменение вероятности при поощрении решения вычисляется следующим образом: $P_{t+1} = P_t + \Delta$;
- изменение вероятности при наказании решения вычисляется как: $P_{t+1} = P_t - \Delta$.

Таким образом, картина вероятности для каждой клетки, неоткрытой в текущий момент, обуславливается опытом удачных или неудачных решений в схожей ситуации и, с увеличением объема накопленных результатов, изменяется на соответствующую величину.

Результаты применения данного алгоритма представлены в таблице 1 [3].

	Без анализа вероятностей			С анализом вероятностей		
число мин	игр	побед	%	игр	побед	%
$q = 10$	50	38	76,0	49	34	69,4
$q = 15$	50	6	12,0	49	23	46,9
$q=20$	50	0,5	1,0	72	6	8,3

Табл. 1. Результаты применения алгоритма

1.2.2 Разработанный метод построения самообучающейся системы

Основная идея данного метода заключается в синтезе новых (производных) правил решения задачи на основе базовых правил.

Синтез новых правил будет происходить при помощи обучения на мини-полях размером 3*3, 4*4, 5*5 и 6*6 клеток. В данных полях для каждой клетки генерируются значения, которые могут находиться в клетках (это цифры [0; 8], мина (М), стена (С), закрытая клетка (З)). С помощью полного перебора значений в клетках и с помощью применения базовых правил выявляются:

- недопустимые (запретные) комбинации;
- единственно возможные комбинации (для прогнозирования значений в закрытых клетках).

Все выявленные комбинации записываются в отдельный список правил, которые являются производными правилами от базовых правил логической задачи.

Затем полученные правила анализируются и выявляются зависимости (для определения основных свойств отношений, применяемых в дискретной математике: рефлексивность, симметричность, транзитивность и др.). Это позволит компактнее сформировать уже синтезированные правила, а также провести синтез новых правил, таким образом сформировав набор шаблонов, по которым можно будет выявлять, есть ли в той или клетке мина.

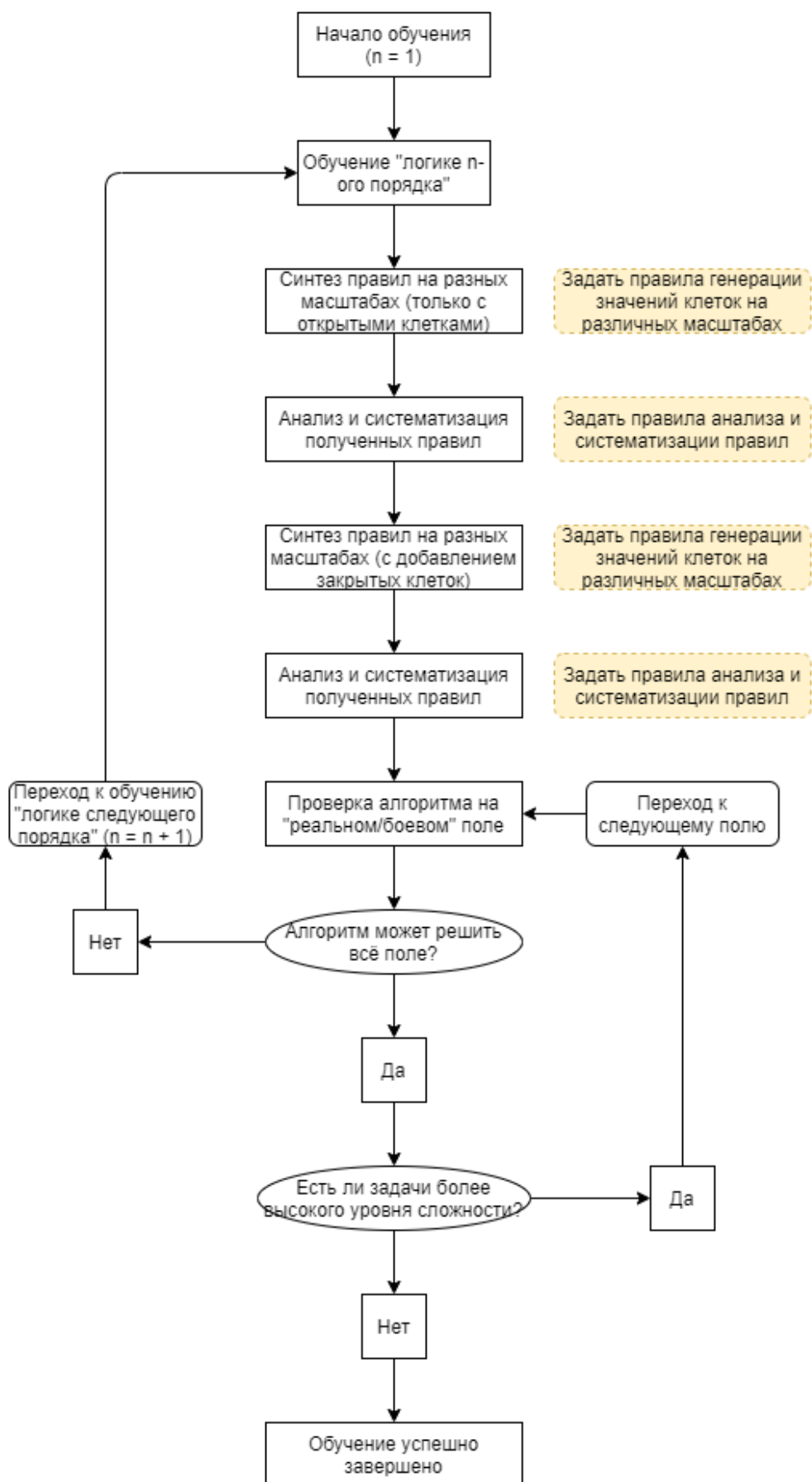


Рис. 1.Блок-схема разработанного алгоритма

Выявлять новые правила при помощи мини-полей будем итеративно. На первой итерации будут применяться только базовые правила игры. На последующих итерациях будут применяться правила, основанные на принципе «что, если», то есть, сначала, по определённым правилам, будет выдвигаться предположение для закрытой клетки (например: предположим, что в данной клетке мина) и, исходя из данного предположения, будут проверяться уже имеющиеся правила и синтезироваться новые.

Блок-схема данного метода представлена на рисунке 1.

Рассмотрим подробнее описание применения логики n-ого порядка.

Логика 1-го порядка основана на правиле поскольку в клетке 1 значение x , то в клетке 2 значение y . Пример логики 1-го порядка приведён на рисунке 2.

Пример логики 1-ого порядка				
	0	1	2	3
0	0	0	0	0
1	0	М	0	0
2	0	0	0	0
3	0	0	0	0

В клетке (1; 1) не может быть "мины", поскольку в клетке (0; 0) находится цифра 0

Рис. 2. Пример логики 1-го порядка

Логика 2-го порядка основана на следующем правиле. Исходя из базовых правил и правил, полученных в ходе применения логики 1-го порядка, установлено, что мина может быть в клетке 1 или в клетке 2. Тогда можно выделить 2 пункта для синтеза новых правил:

- Если мина находится в клетке 1, то, исходя из данного предположения, в соседних клетках будет определённый набор значений (назовём его *набор 1*). Если мина находится в клетке 2, то, исходя из данного предположения, в соседних клетках будет другой набор значений (назовём его *набор 2*). Тогда к набору 1 и набору 2, а, точнее, к соответствующим закрытым значениям

клеток *набора 1* и *набора 2* можно применить логическую операцию **И**, чтобы выявить схожие значения.

- Для *набора 1* и *набора 2* производится применение правил низших порядков для выявления ошибочных значений.

Таким образом, если в первом пункте будут найдены схожие значения для закрытых клеток двух полей и/или во втором пункте будут найдены ошибки при применении правил низших порядков, полученные схемы будут занесены в список правил 2-го порядка.

Пример применения первого пункта логики 2-го порядка представлен на рисунке 3.

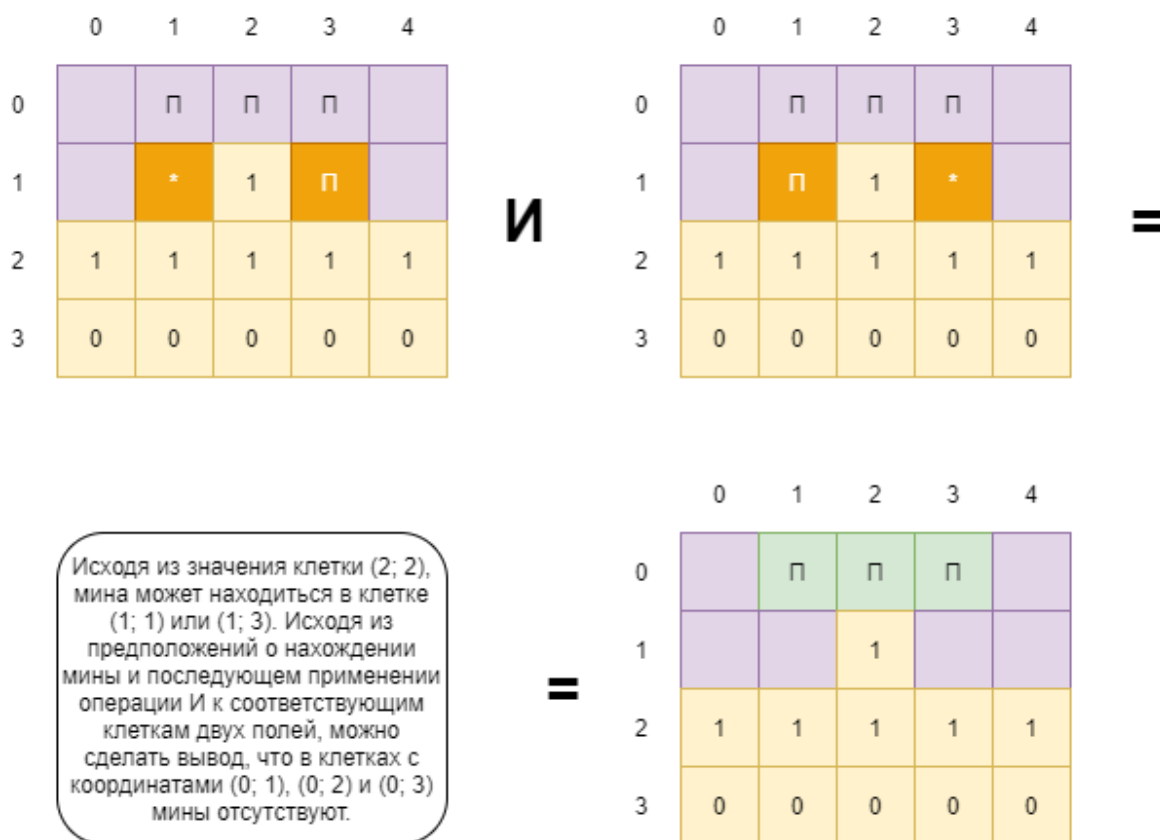


Рис. 3. Пример применения первого пункта логики 2-го порядка

Пример применения второго пункта логики 2-го порядка представлен на рисунке 4.

Пример логики 1-ого порядка				
	0	1	2	3
0	0	0	0	0
1	0	М	0	0
2	0	0	0	0
3	0	0	0	0
В клетке (1; 1) не может быть "мины", поскольку в клетке (0; 0) находится цифра 0				

Рис. 4. Пример применения второго пункта логики 2-го порядка

1.2.3 Анализ алгоритмов построения самообучающихся систем

Для определения предпочтительного метода для разработки самообучающегося алгоритма для решения логической задачи необходимо чётко определить критерии выбора.

Главным критерием является точность решения задачи. Задача, которая была выбрана в качестве задачи для реализации самообучающейся системы, имеет однозначное решение, поэтому для достижения максимальной эффективности важно, чтобы каждое поле головоломки решалось безошибочно.

Первый рассмотренный подход к разработке самообучающегося алгоритма основан на использовании вероятностной обучаемой стратегии. Данный алгоритм хоть и демонстрирует повышение эффективности при решении логической задачи «Сапёр», тем не менее, алгоритм не гарантирует безошибочность решения поля. Более того, данный алгоритм показывает невысокие значения доли побед для полей с числом мин q , равным 10 и 15 (0,694 и 0,469 соответственно), а для полей, с числом мин q , равным 20 алгоритм демонстрирует низкое значение доли побед (0,083). При заданных ограничениях применение данного подхода к разработке самообучающегося алгоритма недопустимо.

Второй рассмотренный подход основан на «запоминании» схем (шаблонов), которые увеличивают вероятность нахождения подходящего шаблона для того, чтобы найти хотя бы одну такую клетку, в которой отсутствует мина. Важным фактом является

то, что в случае, если самообучающаяся система не сможет найти хотя бы одну такую клетку, в которой отсутствует мина, то самообучающаяся система «уйдёт на дообучение», то есть будет тренироваться для нахождения зависимостей с применением «логики более высоких порядков» и, получив новые схемы, система продолжит пытаться найти решения поля. Таким образом, худшим вариантом при нахождении решения логической задачи может стать заикливание алгоритма, который с каждым разом будет искать шаблоны, используя логику и более и более высоких порядков. При этом возникать таких ситуаций, в которых поле решено с ошибкой, не должно.

Исходя из представленного анализа, можно сделать вывод о том, что разработанный метод для решения логической задачи является более подходящим, чем гибридная модель анализа ситуаций. Тем не менее, разработанный метод необходимо доработать, чтобы избежать заикливания при самообучении на полях, не подходящих под требования к исходным данным.

2 СПЕЦИАЛЬНАЯ ЧАСТЬ

Цель работы – разработка комплекса алгоритмов с элементами самообучения и их программной реализации для независимого от человека поиска эффективного решения казуально-логических игр (на примере игры «Minesweeper»/«Сапёр»).

Разработка специальной части начинается с описания правил игры «Сапёр». Далее, исходя из правил, будут представлены содержательная и математическая постановки задачи исследования. Исходя из них, будет разработана теоретическая часть, которая будет описывать связь входных и выходных данных. На основе теоретической части будет разработан алгоритм поиска эффективных решений, который впоследствии будет разделён на части (блоки), где далее для части блоков будет описан самообучающийся алгоритм, который будет выполнять функции данного блока. В завершении будет описано и представлено программное обеспечение, реализующее разработанный алгоритм с элементами самообучения, а также будут представлены результаты работы программы. Блок-схема специальной части представлена на рисунке 5.

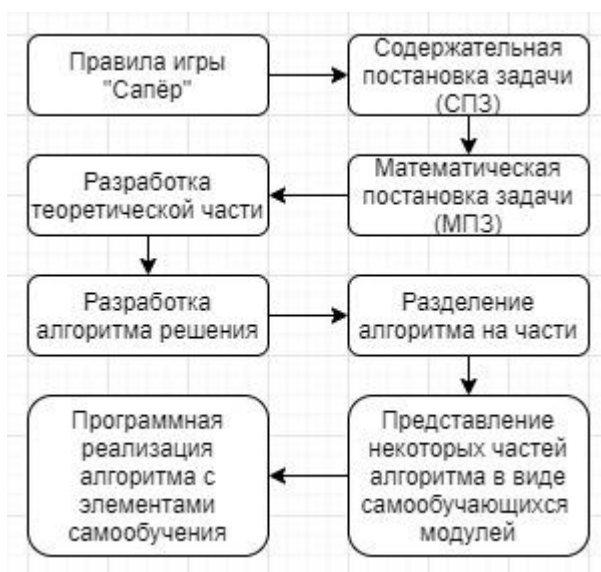


Рисунок 5. Блок-схема специальной части

Правила игры «Сапёр» представлены в приложении А.

2.2 Содержательная постановка задачи

Содержательная постановка задачи представляет из себя выжимку и краткое содержание правил игры. Таким образом, содержательная постановка задачи – это точная и небольшая по объёму формулировка, понятная читателю.

Основным элементом игры является поле с заданными значениями длины и ширины. Элементами поля являются клетки. Клетка может быть открыта или закрыта. В открытой клетке хранится целое значение $[0; 8]$ или мина. Значения $[0; 8]$ в клетке означает количество мин, находящихся в соседних клетках. В закрытой клетке находится флаг мины, знак вопроса или не находится ничего – пользователь сам решает, что будет находиться в закрытой клетке. Если клетка закрыта, то неизвестно, что в ней находится, в противном случае – известно.

На вход подаётся поле, большая часть клеток которого (или все клетки) закрыты, а также число, означающее общее количество мин, находящихся на поле. Пользователь может открыть закрытую клетку, чтобы узнать значение, которое в ней хранится. Если это значение $[0; 8]$, то данное значение отображается в открываемой клетке. Если в открываемой клетке находится мина, то она также отображается в открываемой клетке, но при этом игра заканчивается поражением. Для победы необходимо открыть все клетки поля, в которых отсутствуют мины. Цель: победить.

Таким образом, исходными данными для решения поставленной задачи являются:

- поле с заданными значениями длины и ширины, состоящее из клеток;
- клетка может быть открыта или закрыта;
- если клетка закрыта, то неизвестно, что в ней находится, в противном случае – известно;
- в открытой клетке хранится целое значение $[0; 8]$ или мина;
- в закрытой клетке находится флаг мины, знак вопроса или не находится ничего – пользователь сам решает, что будет находиться в закрытой клетке;
- пользователь может открыть клетку, чтобы узнать значение в ней;
- общее количество мин, которое находится на поле;
- набор условий, определяющие правила игры, а именно:
 - цифра в клетке определяет количество мин в соседних клетках;
 - условие победы (цель игры): открыть все клетки поля, не содержащие мины;
 - условие поражения: открыть клетку с миной.

2.3 Математическая постановка задачи

Математическая постановка задачи представляет из себя переработанную содержательную постановку задачи, представленную с помощью математических терминов. Большинство терминов, которые представлены в данном разделе можно найти в тезаурусе, но, для лучшего понимания, определение некоторых терминов будет приведено сразу.

Основным элементом игры является множество F (field – поле), мощность которого равна $l*w$ (length – длина и width – ширина соответственно). Определим нумерацию множества F как всякое отображение n множества C_o на множество F , где множество C_o (coordinates – координаты) получено при декартовом (прямом) произведении множества целых чисел $[0; l]$ и множества целых чисел $[0; w]$. Пара $F_c = (F, n)$, где n – некоторая нумерация множества F будет называться нумерованным множеством. Для множества F_c обозначим элемент с номером (i, j) как $F_c[i, j]$.

Элементами множества F являются множества C (cell – клетка), состоящие из 4-х элементов:

$$\{cv \in CCV, ov \in OCV, v \in CCV \cup OCV, s \in St\},$$

где s – (status – состояние) – элемент множества St , которое отвечает за состояние клетки – открыта она или закрыта,

cv – (close value – закрытое значение) – элемент множества CCV , которое отвечает за значение, находящееся в закрытой клетке,

ov (open value – открытое значение) – элемент множества OCV , которое отвечает за значение, находящееся в открытой клетке,

v (value – значение) – элемент множества $CCV \cup OCV$, означающий значение, находящееся в клетке.

Множество CCV (close cell values – значения закрытых клеток) состоит из значений MF (mine flag – флаг мины), Q (question – вопрос/сомнение), E (emptiness – пустота).

Множество OCV (open cell values – значения открытых клеток) состоит из множества целых чисел $[0; 8]$ и элемента M (mine – мина).

Множество St (status – состояние) состоит из двух элементов: C (close – закрыта) и O (open – открыта).

Изначально любой элемент cv множества C имеет значение E . Изначально, если элемент s множества C имеет значение C , то значение элемента v равно значению элемента cv . Если элемент s множества C имеет значение O , то значение элемента v равно значению элемента ov . Также пользователь не имеет доступ к элементам cv и ov множества C .

Зададим несколько функций над множеством F_c :

$$sign_M = f(x, y): F_c \rightarrow \{0, 1\} = \begin{cases} 1, \text{ если } ov \in F_c[x, y] = M \\ 0, \text{ если } ov \in F_c[x, y] \neq M \end{cases}$$

$$sign_{oc} = f(x, y): F_c \rightarrow \{0, 1\} = \begin{cases} 1, \text{ если } st \in F_c[x, y] = O \\ 0, \text{ если } st \in F_c[x, y] = C \end{cases}$$

Определим множество Fac (adjacent cells – соседние клетки) мощностью $l*w$. Определим нумерацию множества Fac как всякое отображение n множества C_o на множество Fac , где множество C_o (coordinates – координаты) получено при декартовом (прямом) произведении множества целых чисел $[0; l]$ и множества целых чисел $[0; w]$. Пара $Fac_c = (Fac, n)$, где n – некоторая нумерация множества Fac будет называться нумерованным множеством. Для множества Fac_c обозначим элемент с номером (i, j) как $Fac_c[i, j]$. Элементы множества Fac_c определим следующим образом:

$$Fac_c[x, y] = \left\{ \begin{array}{l} \{F_c[x-1, y-1], F_c[x-1, y], F_c[x-1, y+1], F_c[x, y-1], F_c[x, y+1], F_c[x+1, y-1], \\ F_c[x+1, y], F_c[x+1, y+1]\}, \text{ если } 0 < x < l, 0 < y < w, ov \in F_c[x, y] \neq M \\ \{F_c[x, y-1], F_c[x, y+1], F_c[x+1, y-1], F_c[x+1, y], F_c[x+1, y+1]\}, \\ \text{если } x = 0, 0 < y < w, ov \in F_c[x, y] \neq M \\ \{F_c[x-1, y-1], F_c[x-1, y], F_c[x-1, y+1], F_c[x, y-1], F_c[x, y+1]\}, \\ \text{если } x = l, 0 < y < w, ov \in F_c[x, y] \neq M \\ \{F_c[x-1, y], F_c[x-1, y+1], F_c[x, y+1], F_c[x+1, y], F_c[x+1, y+1]\}, \\ \text{если } 0 < x < l, y = 0, ov \in F_c[x, y] \neq M \\ \{F_c[x-1, y-1], F_c[x-1, y], F_c[x, y-1], F_c[x+1, y-1], F_c[x+1, y]\}, \\ \text{если } 0 < x < l, y = w, ov \in F_c[x, y] \neq M \\ \{F_c[x, y+1], F_c[x+1, y], F_c[x+1, y+1]\}, \text{ если } x = 0, y = 0, ov \in F_c[x, y] \neq M \\ \{F_c[x, y-1], F_c[x+1, y+1], F_c[x+1, y]\}, \text{ если } x = 0, y = w, ov \in F_c[x, y] \neq M \\ \{F_c[x-1, y], F_c[x-1, y+1], F_c[x, y+1]\}, \text{ если } x = l, y = 0, ov \in F_c[x, y] \neq M \\ \{F_c[x-1, y-1], F_c[x-1, y], F_c[x, y-1]\}, \text{ если } x = l, y = w, ov \in F_c[x, y] \neq M \\ \{M\}, \text{ если } ov \in F_c[x, y] = M \end{array} \right.$$

Определим функцию над множеством Fac_c :

$$sign_{M2}[x, y] = f(x_1): Fac_c[x, y] \rightarrow \{0, 1\} = \begin{cases} 1, \text{ если } ov \in sonc[x, y]_c[x_1] = M \\ M, \text{ если } ov \in sonc[x, y]_c[x_1] \neq M \end{cases}$$

Теперь можно определить тождественное отображение R множества F_c в себя же:

$$R = f(x, y): F_c \rightarrow F_c = \begin{cases} \sum_{i=0}^{|Fac_c[x, y]|} sign_{M2}[x, y][i], \text{ если } Fac_c[x, y] \neq \{M\} \\ M, \text{ если } Fac_c[x, y] = \{M\} \end{cases}$$

Также задаётся множество состояний игры GS (game status – статус/состояние игры), состоящее из 3-ёх элементов: V (victory – победа), D (defeat – поражение) и N/O (not over – игра не окончена). Определим переменную gs (game status – статус/состояние игры), значение которой принадлежит множеству GS . В начале игры переменная gs имеет

значение N/O. Игра считается завершённой, когда значение переменной *gs* изменится на V (в таком случае считается, что игра завершилась победой) или на D (в таком случае считается, что игра завершилась поражением).

Общее количество мин на поле задаётся параметром *tm* (total mines – общее количество мин), значение которого рассчитывается по следующей формуле:

$$tm = \sum_{i=0}^l \sum_{j=0}^w sign_M(i, j)$$

Зададим параметр *cwm* (cells without mines – клеток без мин) и зададим переменную *occ* (open cell current – текущее количество открытых клеток). Параметр *cwm* рассчитывается по следующей формуле:

$$cwm = l * w - tm$$

Начальное значение переменной *occ* рассчитывается по следующей формуле:

$$occ = \sum_{i=0}^l \sum_{j=0}^w sign_oc(i, j)$$

Также определим множество пользовательских допустимых изменений состояний (переходов) UT (user transition – пользовательский переход) и множество условных переходов CT (conditional transition – условный переход). Будем называть пользовательским переходом такое изменение переменной и/или элемента или элементов тех или иных множеств, которое может осуществлять пользователь. Данное множество будет состоять из следующих элементов:

- Для множества $F_c[x, y]: v \neq MF, s: C \rightarrow O; v, s \in F_c[x, y]$,
- Для множества $F_c[x, y]: s = C, cv: E \rightarrow MF; s, cv \in F_c[x, y]$,
- Для множества $F_c[x, y]: s = C, cv: E \rightarrow Q; s, cv \in F_c[x, y]$,
- Для множества $F_c[x, y]: s = C, cv: MF \rightarrow E; s, cv \in F_c[x, y]$,
- Для множества $F_c[x, y]: s = C, cv: MF \rightarrow Q; s, cv \in F_c[x, y]$,
- Для множества $F_c[x, y]: s = C, cv: Q \rightarrow E; s, cv \in F_c[x, y]$,
- Для множества $F_c[x, y]: s = C, cv: Q \rightarrow MF; s, cv \in F_c[x, y]$.

Назовём условным переходом такое изменение переменной и/или элемента или элементов тех или иных множеств, которое происходят автоматически, исходя из изменения переменной и/или элемента или элементов множеств после пользовательского перехода. Условный переход состоит из двух частей: условной (которое, в свою очередь является пользовательским переходом) и действенной части (определяет переход, который осуществляется при условии выполнения указанного пользовательского перехода). Данное множество будет состоять из следующих элементов:

- ЕСЛИ для множества $F_c[x, y]: v \neq MF, s: C \rightarrow O; v, s \in F_c[x, y]$,
ТО для множества $F_c[x, y]: v: v \rightarrow ov$,
- ЕСЛИ для множества $F_c[x, y]: s = C, cv: E \rightarrow MF; s, cv \in F_c[x, y]$,
ТО для множества $F_c[x, y]: cv: cv \rightarrow v$,
- ЕСЛИ для множества $F_c[x, y]: s = C, cv: E \rightarrow Q; s, cv \in F_c[x, y]$,
ТО для множества $F_c[x, y]: cv: cv \rightarrow v$,
- ЕСЛИ для множества $F_c[x, y]: s = C, cv: MF \rightarrow E; s, cv \in F_c[x, y]$,
ТО для множества $F_c[x, y]: cv: cv \rightarrow v$,
- ЕСЛИ для множества $F_c[x, y]: s = C, cv: MF \rightarrow Q; s, cv \in F_c[x, y]$,
ТО для множества $F_c[x, y]: cv: cv \rightarrow v$,
- ЕСЛИ для множества $F_c[x, y]: s = C, cv: Q \rightarrow E; s, cv \in F_c[x, y]$,
ТО для множества $F_c[x, y]: cv: cv \rightarrow v$,
- ЕСЛИ для множества $F_c[x, y]: s = C, cv: Q \rightarrow MF; s, cv \in F_c[x, y]$,
ТО для множества $F_c[x, y]: cv: cv \rightarrow v$,
- ЕСЛИ для множества $F_c[x, y]: s: C \rightarrow O, ov = M; s, ov \in F_c[x, y]$,
ТО переменная $gs: N/O \rightarrow D$,
- ЕСЛИ для множества $F_c[x, y]: s: C \rightarrow O, ov \neq M; s, ov \in F_c[x, y]$,
ТО переменная $oss: oss \rightarrow oss + 1$,
- ЕСЛИ для переменной $oss: oss \rightarrow oss + 1, oss + 1 = oct$,
ТО переменная $gs: N/O \rightarrow V$,
- ЕСЛИ для множества $F_c[x, y]: ov = 0, s: C \rightarrow O; ov, s \in F_c[x, y]$,
ТО для каждого элемента подмножества $Fac_c[x, y]$ множества $F_c: v: \rightarrow ov, s: C \rightarrow O$.

Итак, входными данными являются:

- множество F_c , мощностью $l*w$,
- значение параметра tm .

Успешным выполнением задачи является достижение переменной gs значения V .

Неуспешным выполнением задачи является достижение переменной gs значения D .

2.4 Описание алгоритма

2.4.1 Описание метода 1

2.4.2 Описание метода 2

2.4.3 Описание метода 3

2.4.4 Описание применения схем

2.5 Теоретическое доказательство применимости разработанного алгоритма для нахождения решения любого поля игры «Сапёр» при заданных ограничениях

2.6 Описание программной реализации

2.7 Результаты работы программы

ВЫВОДЫ

В данной работе был произведён поиск и анализ логических задач, которые подходили по заданным критериям для разработки самообучающейся системы. Из большого набора представленных логических задач – головоломок, была выбрана задача поиска решения в игре сапёр.

Для выбранной логической задачи был найден и описан подход к разработке самообучающегося алгоритма, а также описан собственный алгоритм для самообучающейся системы, произведён сравнительный анализ представленного подхода и алгоритма и определён более предпочтительный вариант для реализации.

Также в данной работе описана содержательная постановка задачи исследования и представлен математический аппарат, с помощью которого будет реализован самообучающийся алгоритм.

В дальнейшем планируется разработка самообучающейся системы, основанной на предложенном алгоритме, тестирование данной системы и проведение сравнительного анализа работы разработанной самообучающейся системы и других алгоритмов решения поставленной задачи, в том числе выявления достоинств и недостатков в работе программы.

В случае успеха разработки самообучающейся системы планируется модифицировать её для решения других логических задач, таких как sudoku и «Бэнг!».

ТЕЗАУРУС

- **Клетка** – это ячейка с определёнными координатами $(x; y)$, состоянием (O/C – открыта/закрыта из множества **S**), в которой хранится значение из множества **TCV**.
- **Множество S** (status – состояние) – множество статусов клетки, состоящее из двух элементов: C (close – закрыта), O (open – открыта).
- **Закрытая клетка** – клетка с состоянием C (закрыта).
- **Открытая клетка** – клетка с состоянием O (открыта).
- **Множество TCV** (total cell values – общие значения ячеек) – множество, содержащее целые числа $[0; 8]$, элемент **M** (мина – mine), элемент **MF** (флаг мины – mine flag), элемент **Q** (вопрос/сомнение – question), элемент **E** (пустота – emptiness).
- **Истинная клетка** – это ячейка с определёнными координатами $(x; y)$, в которой хранится значение из множества **CV** (cell values).
- **Множество CV** (cell values – значения ячеек) – множество, содержащее целые числа $[0; 8]$, а также элемент **M** (мина – mine).
- **Мина** – это элемент **M** множеств **CV** и **TCV**. Словосочетание «в клетке находится мина» означает, что в заданной клетке хранится значение **M** множества **CV/TCV**. Словосочетание «в клетке отсутствует мина» означает, что в заданной клетке хранится значение, отличающееся от значения **M** множества **CV/TCV**.
- **Поле** – множество клеток/истинных клеток, размером $l*w$, в котором координаты каждой клетки уникальны и находятся в диапазоне $(0 \leq x < l; 0 \leq y < w)$.
- **Соседняя клетка** – это такая клетка поля, координаты $(x_1; y_1)$ которой отличаются от координат заданной клетки $(x_2; y_2)$ не более, чем на 1 (то есть $(x_2 - 1 \leq x_1 \leq x_2 + 1; y_2 - 1 \leq y_1 \leq y_2 + 1)$). При этом, $x_1 \geq 0, y_1 \geq 0, x_2 \geq 0, y_2 \geq 0$.
- **Изолированная клетка** – это такая клетка поля, для которой все соседние клетки – закрытые.
- **Связанные клетки** – это множество закрытых клеток поля, в котором суммарное количество значений **M** элементов поля больше 0.
- **Гипотеза** – это предположение, что в выбранной закрытой клетке хранится/хранятся заданное/заданные значение.
- **Множество G** (hypothesis - гипотеза) – это множество гипотез, состоящее из двух элементов: (клетка хранит одно из значений подмножеств **CV/TCV** - целые числа $[0; 8]$; клетка хранит значение подмножеств **CV/TCV** – **M** (мина)).

- **Противоположные гипотезы** – это две гипотезы из множества гипотез G .
- **Проверка гипотезы** – подтверждение или опровержение гипотезы практическим путём.
- **Фокусная клетка** – при проверке гипотезы это закрытая клетка, для которой проверяется выполнение гипотезы.
- **Схема** – это множество пар $\{клетка; состояние\}$, для которых выполняется условие:
ЕСЛИ $\{клетка_1; 0\}$ И $\{клетка_2; 0\}$ И ... И $\{клетка_n; 0\}$, ТО $\{клетка_n+1; 1\}$.
- **Применение схемы** (к полю) –
- **Правила** – набор исходных логических условий, определяющие допустимые и недопустимые ситуации.
- **Недопустимая ситуация** – это такая ситуация, выявление которой приводит к проигрышу.
- **Допустимая ситуация** – это ситуация, являющаяся необходимым условием для победы.
- **Некорректная ситуация** – это такая ситуация (относительно выбранной открытой клетки), при которой как минимум одно значение в соседних клетках вычислено неверно.
- **Корректная ситуация** – это такая ситуация (относительно выбранной открытой клетки), при которой выполняются базовые правила для выбранной открытой клетки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Виды головоломок. Саморазвитие 2.0. URL: <http://pruslin.ru/vidy-golovolomok/> (дата обращения: 26.12.21).
2. Е. Ю. Корлякова, М. О. Корлякова. Подход к разработке самообучающегося алгоритма игры в «Сапёр». Научно-технические технологии в приборостроении и машиностроении и развитие инновационной деятельности в вузе. Материалы Всероссийской научно-технической конференции. Том 2. Изд. КФ МГТУ им. Баумана, Калуга, 2016. С. 23-24.
3. Е. Ю. Корлякова. Подход к разработке самообучающегося алгоритма игры в «Сапёр». Презентация к докладу. Калужский филиал МГТУ им. Баумана, Калуга, 2016.

ПРИЛОЖЕНИЕ А. Правила игры «Minesweeper»/ «Сапёр»

Игра имеет три уровня сложности: Новичок, Любитель и Профессионал. Уровни различаются размером игрового поля, а также общим количеством мин на поле.

На уровне Новичок размер поля равен 8x8, однако в операционной системе Windows XP он равен 9x9.

На уровне Любитель размер поля равен 16x16; на уровне Профессионал размер поля равен 30x16.

Суть игры заключается в том, что некоторые клетки поля свободны, а некоторые заминированы. Каждая клетка поля либо свободна, либо заминирована; третьего не дано.

Перед началом игры точно известно, сколько всего мин будет установлено на поле. По умолчанию, на уровне Новичок устанавливается 10 мин; на уровне Любитель — 40 мин; на уровне Профессионал — 99 мин. Это число можно изменить. Количество обнаруженных мин указывается в левом верхнем углу.

Однако при этом неизвестно, какие именно клетки заминированы. До того, как игрок не откроет клетку или не закончится игра, неизвестно, какие именно клетки свободны, а какие заминированы.

Если игрок считает, что некая клетка свободна, он может нажать её левой кнопкой мыши. При этом клетка откроется. Возможны три варианта:

1) Если открываемая клетка была заминирована, то игра показывает это. В той клетке, которая была открыта, появляется мина, залитая красным. Также при этом открываются все остальные клетки с минами: появляется рисунок мины, но уже без красной заливки. Игра немедленно заканчивается, игрок умер (проиграл).

2) Если открываемая клетка была свободна, но хотя бы одна из соседних клеток содержит мину, то в открываемой клетке появляется цифра, указывающая общее количество мин в соседних клетках. 1 — одна соседняя клетка заминирована; 2 — две соседние клетки заминированы, и так далее. Соседними в Сапёре считаются клетки, которые имеют либо общую сторону (соприкасаются по горизонтали или по вертикали), либо имеют общий угол (соприкасаются в углу по диагонали). Очевидно, что в клетке может появляться одна из цифр от 1 до 8. Каждая цифра имеет свой цвет: 1 — голубой, 2 — зелёный, 3 — красный, 4 — синий, 5 — коричневый, и так далее. Игра продолжается.

3) Если открываемая клетка была свободна и все соседние клетки тоже свободны, то все соседние клетки, как и открываемая клетка, автоматически разминируются и помечаются как свободные (окрасятся в серый цвет). Если какая-либо из разминированных клеток также будет иметь только свободных соседей, то процесс автоматического разминирования на

этом же ходу продолжится далее. И так далее, пока не обнаружатся клетки, имеющие заминированных соседей, и не появятся цифры. Игра продолжается.

Если игрок считает, что в клетке находится мина, он может нажать её правой кнопкой мыши. При этом клетка пометится как потенциально заминированная: в этой клетке появится своеобразный флажок. Кроме того, количество обнаруженных мин при этом уменьшится на единицу. Подчёркиваю, что флажок означает мнение игрока о наличии мины, но вовсе не означает фактическую мину.

Нельзя открывать (нажимать левой кнопкой) клетки, помеченные флажком. Нельзя помечать флажками разминированные клетки — то есть серые либо помеченные цветными цифрами.

Но если игрок сомневается, то он может нажать на помеченную флажком клетку правой кнопкой мыши вторично. При этом флажок поменяется на вопросительный знак. Вопросительный знак в клетке означает, что клетка имеет неопределённый статус и заслуживает внимания. Эту клетку можно оставить на время и вернуться к ней позже.

Можно нажать на клетку со знаком вопроса правой кнопкой мыши в третий раз. При этом знак вопроса исчезнет, и клетка вернётся к своему изначальному состоянию — неопределённое состояние, отсутствие изображения.

Игра продолжается до тех пор, пока каждая клетка поля не будет иметь один и только один из следующих двух статусов:

- 1) разминирована (либо серая, либо содержит цветную цифру);
- 2) правильно помечена флажком как содержащая мину.

Если игрок пройдёт всё поле — он выиграл.

Очевидно, что часто игрок в любой момент может наткнуться на мину, это означает, что он проиграл. При этом часто бывает, что одна или несколько клеток помечены игроком неверно. В этом случае игра помечает такие клетки перечёркнутой миной, что означает, что мины там не было, несмотря на то, что игрок поставил там флажок.