УЧЕБНАЯ МАСТЕРСКАЯ

Пестов Олег Александрович,
Шалыто Анатолий Абрамович

САПЕР, МИНЫ И АВТОМАТЫ

В условиях недостаточности данных допускается куда меньше ошибок, чем при полном отсутствии данных.
Чарльз Бэббидж

ВВЕДЕНИЕ

Игра «Сапер» [1, 2] уже несколько лет входит в стандартную поставку операционных систем фирмы *Microsoft*. Благодаря своей распространенности, а также простоте описания, она стала популярной, и это повлекло за собой работы по автоматизации процесса игры, в ходе которых ставилась задача написать программу, способную автоматически вычислять следующий ход в игре. Это позволяет использовать такую программу в качестве советчика. В настоящей работе изложен процесс создания указанной программы.

При этом отметим, что автоматическое вычисление следующего хода не имеет смысла использовать всегда. Например, в начале игры это ни к чему не приведет, так как информация о расположении мин отсутствует. Программа написана таким образом, что она либо со стопроцентной вероятностью вычисляет следующий ход, либо сообщает игроку, что имеющейся информации недостаточно. Во втором случае программа предлагает игроку сделать выбранный случайно ход, а игрок сам решает, как ему поступить.

Игроку в начале игры о расстановке мин на поле ничего не известно.

Опыт использования программы показал, что когда она не знает наверняка, куда ходить, то и авторы не имели полной уверенности в выборе следующего хода.

1. ОПИСАНИЕ ИГРЫ «САПЕР»

1.1. Правила игры

На прямоугольном поле размером $N \times M$ клеток случайным образом расставлены K мин. При этом в одной клетке может быть размещено не более одной мины. Игроку в начале игры о расстановке мин на поле ничего не известно. За один ход разрешается открыть одну клетку. Если клетка содержит мину, то игра проиграна, иначе в клетку выводится число от нуля до восьми – суммарное количество мин, расположенных в клетках, соседних с данной. Цель игры – открыть все пустые ячейки, не взорвавшись.

1.2. Пример игры

Пусть на поле размером 3×3 в клетках (3, 1) и (3, 2) расположены мины, от-

...каждая из неоткрытых клеток содержит мину...

меченные звездочками (рисунок 1), а поле перед игроком приведено на рисунке 2.

Опишем, как в данном случае мог бы играть человек, перечислив возможную последовательность ходов.

1. Пусть случайно открыта клетка (2, 3) и в ней появилась цифра 1 (рисунок 3). Следовательно, в одной из соседних пяти клеток находится ровно одна мина.

2. Полученной информации недостаточно для того, чтобы узнать наверняка расположение остальных клеток, которые не содержат мин. Еще раз случайно откроем клетку (рисунок 4).

3. Из рассмотрения рисунка 4 следует, что на одной из клеток (2, 2) или (3, 2) стоит мина. Поэтому клетки (1, 3) и (1, 2) свободны. Откроем их (рисунок 5).

4. Можно открыть еще три пустые клетки (рисунок 6).

5. Все пустые клетки открыты, и цель игры достигнута.

2. АЛГОРИТМ РЕШЕНИЯ

Задача вычисления следующего хода заключается в нахождении еще неизвестной пустой клетки. Отметим, что рассматриваемая задача является *NP*-полной [3, 4] и описываемый алгоритм эвристический.

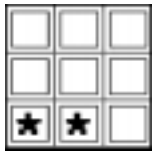


Рисунок 1.

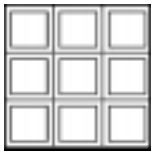


Рисунок 2.

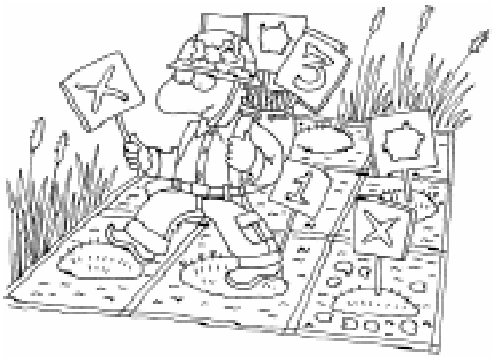
Идея решения основана на том, что можно получать информацию об игровом поле, рассматривая множества еще неоткрытых клеток и делая предположения о количестве мин в каждом из них.

Поясним это на примере, рассмотренном в разд. 1.2. Из рисунка 4 следует, что множество клеток $\{(1,2), (1,3), (2,2), (3,2)\}$ содержит одну мину так же, как и множество $\{(2,2), (3,2)\}$ Поэтому дополнение второго множества до первого (множество $\{(1,2), (1,3)\}$) не содержит мин.

Опишем предлагаемый алгоритм, использующий указанную идею. Во время игры множество всех клеток поля делится на три группы:

– неизвестные;
– мины;
– пустые (открытые).

Элементом назовем непустое множество еще неизвестных клеток вокруг уже открытой клетки. Так, например, для поля,



Во время игры множество всех клеток поля делится на три группы...

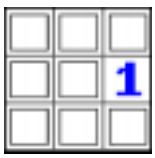


Рисунок 3.

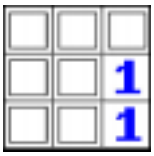


Рисунок 4.

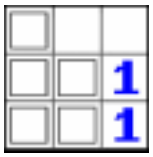


Рисунок 5.

Рисунок 6.

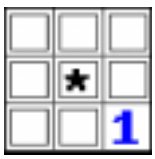


Рисунок 7.

представленного на рисунке 7, элементами являются только следующие три множества: $\{(2,3)\}$, $\{(3,2)\}$ и $\{(2,3), (3,2)\}$.

Каждому элементу сопоставим два числа – минимальное и максимальное количество мин (*состояние* элемента), которое может в нем находиться. Во время работы алгоритма будет происходить пошаговый процесс пересчета соответствующих значений в элементах. На каждом шаге уточняется минимальное и максимальное количество мин во всех элементах поля на основе информации, полученной на предыдущем шаге. Результатом работы является одна из следующих ситуаций.

1. Максимальное количество мин в одном из элементов равно нулю. Это означает, что все клетки, входящие в него пусты.

2. Минимальное количество мин в одном из элементов совпадает с количеством клеток в нем. Это означает, что все клетки, входящие в него – мины.

3. У всех элементов минимальное и максимальное количество мин не изменилось.

В первом и во втором случаях появилась информация о еще неизвестных клетках, и, следовательно, должно измениться множество элементов. Если найдена мина (случай 2), то после изменения множества элементов, следует снова запустить алгоритм, так как результатом его работы является нахождение неизвестной ранее пустой клетки. В третьем – требуется дополнительная информация.

Рассмотрим поочередно все возможные базовые клетки.

Этот алгоритм конечен. В самом деле, если алгоритм не заканчивает свою работу в результате возникновения одной из описанных выше ситуаций, то на каждом шаге либо увеличивается минимальное число мин, либо уменьшается максимальное число мин одного из элементов. Однако, в силу того, что число мин ограничено, данное изменение проходит конечное число раз. Поэтому, учитывая, что элементов также конечное число, приходим к конечности алгоритма.

Каким же образом происходит уточнение состояния элемента? Пусть A и C – элементы, причем $A \subset C$. Тогда, если $B = C \setminus A$ также является элементом, то справедливы следующие соотношения:

$$\mathrm{mn}(A) \le \mathrm{val}(A) \le \mathrm{mx}(A)$$
$$\mathrm{mn}(C) \le \mathrm{val}(A) + \mathrm{val}(B) \le \mathrm{mx}(C)$$

Они являются формальной записью того факта, что число мин в элементе (val) должно быть больше либо равно минимального (mn) числа мин и меньше либо равно максимального (mx) их числа. Приведем соотношение, получающееся в результате вычитания переменной $\mathrm{val}(B)$ из обеих частей второго соотношения:

$$\mathrm{mn}(A) \le \mathrm{val}(A) \le \mathrm{mx}(A)$$
$$\mathrm{mn}(C) - \mathrm{val}(B) \le \mathrm{val}(A) \le \mathrm{mx}(C) - \mathrm{val}(B)$$

Учитывая, что $\mathrm{mn}(B) \le \mathrm{val}(B) \le \mathrm{mx}(B)$, получим:

$$\mathrm{mn}(A) \le \mathrm{val}(A) \le \mathrm{mx}(A)$$
$$\mathrm{mn}(C) - \mathrm{mx}(B) \le \mathrm{val}(A) \le \mathrm{mx}(C) - \mathrm{mn}(B)$$

Таким образом,

$$\begin{cases} \mathrm{mn}(A) = \max(\mathrm{mn}(A), \mathrm{mn}(C) - \mathrm{mx}(B)) \\ \mathrm{mx}(A) = \min(\mathrm{mx}(A), \mathrm{mx}(C) - \mathrm{mn}(B)) \end{cases}$$

Полученные выражения можно преобразовать во временные выражения, определяя состояние элемента A в момент времени $t + 1$ через состояния элементов A, C и $B = C \setminus A$ в момент времени t:

$$\begin{cases} \mathrm{mn}(A_{t+1}) = \max(\mathrm{mn}(A_t), \mathrm{mn}(C_t) - \mathrm{mx}(B_t)) \\ \mathrm{mx}(A_{t+1}) = \min(\mathrm{mx}(A_t), \mathrm{mx}(C_t) - \mathrm{mn}(B_t)) \end{cases} \quad (1)$$

Начальное состояние каждого элемента определяется следующим образом. Все клетки элемента являются соседями какой-либо пустой (базовой) клетки. Рассмотрим поочередно все возможные базовые клетки. Пусть *free* – общее количество неизвестных клеток вокруг текущей базовой клетки, *state* – общее количество мин вокруг нее, а *mines* – количество уже известных мин вокруг нее. Тогда,

$$\begin{cases} \mathrm{mn}(A_0) = \max(0, state - mines - free + |A|) \\ \mathrm{mx}(A_0) = \min(|A|, state - mines) \end{cases} \quad (2)$$

Описанная структура является клеточным автоматом [5–7], в котором элементами являются клетки, а состояния – два числа (минимальное и максимальное число мин). У элементов есть соседи, от состояния которых зависят их состояния. Элементы A, B и C являются соседями, если $A \subset C$ и множество $B = C \setminus A$ также является элементом. Тогда соотношения (1) и (2) являются основой функционирования итеративного алгоритма как клеточного автомата.

В статье в качестве примера приведен фрагмент программы, позволяющей определить следующий ход на базе соотношений (1) (листинг 1).

3. ВНЕШНЕЕ ПРЕДСТАВЛЕНИЕ

Разработанная программа *Tips minesweeper* выполнена как *Windows*-приложение (рисунок 8), с той же функциональностью, что и программа *Minesweeper* фирмы *Microsoft*, но с одним ключевым отличием – возможностью вызова подсказки. При этом происходит выполнение описанного в разделе 2 алгоритма. Если удается найти еще одну пустую клетку, то она подсвечивается желтым цветом, иначе выводится сообщение «*I don't know*».

На рисунке 8 приведен интерфейс игры в момент использования меню. В рамках работы приложения игрок может:

– открыть еще неоткрытую клетку поля (нажатие левой кнопки мыши);

– пометить неоткрытую клетку с миной звездочкой, либо убрать существующую метку (нажатие правой кнопки мыши);

Рисунок 8.

– расставить мины заново, оставив основные характеристики поля (число строк, столбцов и мин) неизменными. Это обеспечивают пункт *New* в меню *Game* или нажатие кнопки F2;

– изменить параметры игрового поля (через меню *Game*). Возможны три варианта: *Beginner* (поле размером 8×8 и расставлено 10 мин), *Intermediate* (16×16, 40 мин) и *Expert* (16×30, 99 мин);

– попросить подсказку (пункт *Hint* в меню *Game* или нажатие кнопки F3).

...возможностью вызова подсказки.

Листинг 1.

```
/**
 * Пересчитывает состояние элемента b.
 */
private void changeState(Element a, Element b, Element c) throws
AlgorithmException
{
    State aState = (State) elementsState.get(a);
    State bState = (State) elementsState.get(b);
    State cState = (State) elementsState.get(c);

    if (bState.setMin(cState.getMin() - aState.getMax()) ||
        bState.setMax(cState.getMax() - aState.getMin()))
        touchedElements.add(b);
}

/**
 * Пересчитывает состояние элементов, соседних с данным.
 */
private void processElement(Element e) throws AlgorithmException
{
    Element o;
    for (Iterator i = ((HashSet) innerNeighbors.get(e)).iterator();
    i.hasNext();)
    {
        o = (Element) i.next();
        changeState(o, e.complement(o), e);
    }

    for (Iterator i = ((HashSet) outerNeighbors.get(e)).iterator();
    i.hasNext();)
    {
        o = (Element) i.next();
        changeState(e, o.complement(e), o);
    }
}

/**
 * Находит следующий элемент, состоящий только из мин или только из
 * пустых клеток.
 */
public Element predict() throws AlgorithmException
{
    while (!touchedElements.isEmpty())
    {
        Element e = (Element) touchedElements.iterator().next();
        touchedElements.remove(e);
        State s = (State) elementsState.get(e);
        // all cells in this element are mines.
        if (s.getMin() == s.getMax() && s.getMin() == e.size())
            return e;
        // all cells in this element are empty.
        else if (s.getMax() == 0)
            return e;
        else
            processElement(e);
    }

    return null;
}
```

ЗАКЛЮЧЕНИЕ

При использовании приложение продемонстрировало высокую эффективность предложенного алгоритма применительно к случайным минным полям. Применение клеточных автоматов позволило достаточно просто и формально описать алгоритм решения задачи. Правила функционирования алгоритма уместились в задании начальных условий и законах изменения состояния одного элемента в зависимости от состояния его соседей. Это упростило логику программы и значительно сократило временные затраты на написание и отладку приложения.

Исходные коды программы на языке *Java* и скомпилированное приложение доступны как на сайте http://is.ifmo.ru, раздел «Статьи», так и на диске к журналу.

Литература

1. *Minesweeper* (game) http://encyclopedia.thefreedictionary.com/Minesweeper%20(game)

2. *The Authoritative* Minesweeper Community http://metanoodle.com/minesweeper

3. *Richard Kaye's Minesweeper Pages* http://for.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.htm

4. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: Центр непрер. матем. образования, 2000.

5. *Тоффали Т., Марголус Н.* Машины клеточных автоматов. М.: Мир, 1991.

6. *Наумов Л., Шалыто А.* Клеточные автоматы – реализация и эксперименты // Мир ПК. № 8, 2003.

7. *Наумов Л.*, *Шалыто А.* «Цветные» клеточные автоматы, или клонирование Мона Лизы //Мир ПК. № 5, 2004.

Пестов Олег Александрович,
магистрант кафедры
«Компьютерные технологии»
Санкт-Петербургского
государственного университета
информационных технологий,
механики и оптики (СПбГУ ИТМО).

Шалыто Анатолий Абрамович
доктор технических наук,
заведующий кафедрой «Технологии
программирования» СПбГУ ИТМО.