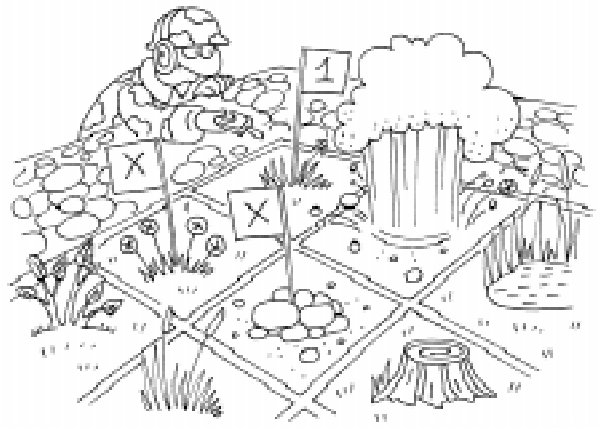
УЧЕБНАЯ МАСТЕРСКАЯ

Комаров Алексей Дмитриевич

ОСТОРОЖНО, МИНЫ! АЛГОРИТМ РЕШЕНИЯ ИГРЫ «САПЁР»

«Сапёр» – это популярная компьютерная логическая игра. Цель игры проста – отыскать все случайным образом расставленные мины на прямоугольном минном поле, разделенном на клетки. Каждая клетка или заминирована, или содержит информацию о количестве мин в восьми смежных клетках.

О популярности «Сапёра» говорят регулярно проводимые чемпионаты [1, 2], формируемые списки лучших игроков со всего мира [3, 4, 5], огромное количество модификаций и усовершенствований игры. Существует «Сапёр» на трехмерном поле, «Сапёр» с различной мощностью мин, с треугольными и шестиугольными клетками, со сценариями серии игр и с неслучайной расстановкой. Но нас интересует классическая игра, знакомая миллионам пользователей Windows. Попробуем выявить алгоритм решения.



В начале игры психологически проще безвреживать единицы...

В начале игры психологически проще обезвреживать единицы, а точнее те клетки, в которых стоит единица и есть смежная к ней только одна закрытая (рисунок 1). Вокруг клетки C3 может находиться только одна заминированная клетка, а так как и закрытая смежная клетка только одна (B2), то она и содержит эту мину.

Несложно расширить эти рассуждения на случай большего числа мин: если количество закрытых смежных клеток равно числу, стоящему в данной клетке, то все эти закрытые клетки содержат мины (рисунок 2).

Таким образом, простейший первоначальный алгоритм решения выглядит, как

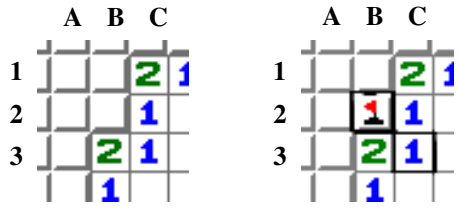


Рисунок 1.

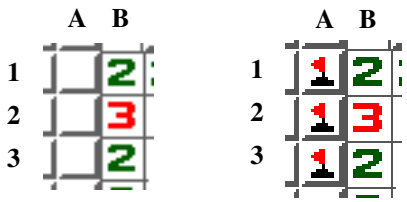


Рисунок 2.

Листинг 1. Первоначальный алгоритм

```
1.01  for i from 1 to n
1.02    for j from 1 to m
1.03      int closed = 0;
1.04      if field[i][j] > 0 then
1.05        проходим по смежным клеткам;
1.06          ищем закрытые клетки: closed++;
1.07        if field[i][j] == closed then
1.08          во всех закрытых клетках мины;
1.09        fi
1.10      fi
```

на листинге 1.

Он далек от совершенства: во-первых, вряд ли решит сколько-нибудь сложную задачу, во-вторых, если не считать заминированную клетку закрытой, то и вовсе будет ошибаться. Поэтому будем теперь учитывать уже известные заминированные поля. Понятно, что если вокруг клетки обнаружено столько мин, сколько требуется, то все остальные смежные клетки безопасны (рисунок 3). Или если суммарное количество смежных закрытых и заминированных клеток дают в точности необходимое число мин, то все такие закрытые клетки заминированы (рисунок 4). Также учтем тот факт, что закрытыми являются все те клетки, содержимое которых неизвестно, что дает возможность не рассматривать клетки, вокруг которых всё уже обезврежено. С учетом вышесказанного, имеем новый алгоритм (листинг 2), который запускается до тех пор, пока может что-то решить, то есть пока производит какие-то изменения.

Подобные пересчеты числа расставленных мин и закрытых клеток, несмотря на

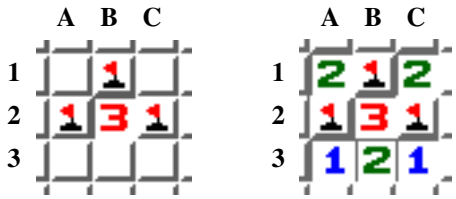


Рисунок 3.

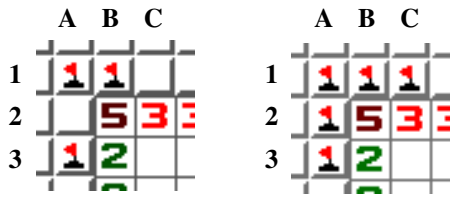


Рисунок 4.

Листинг 2. Усовершенствование

```
2.01  int closed = 0;
2.02  int mines = 0;
2.03  if field[i][j] > 0 then
2.04    проходим по смежным клеткам
2.05      ищем закрытые клетки: closed++;
2.06      ищем мины: mines++;
2.07    if closed > 0 then
2.08      if field[i][j] == mines then
2.09        все закрытые клетки безопасны
2.10      fi
2.11      if field[i][j] == mines + closed then
2.12        во всех закрытых клетках мины
2.13      fi
2.14    fi
2.15  fi
```

...если вокруг клетки обнаружено мин, сколько требуется, то все остальные смежные клетки безопасны...

простоту, довольно продуктивны: алгоритм приведёт к победе на младших уровнях сложности игры. «Профессионал» же из-за высокого относительного количества мин на клетку порождает интересные случаи (рисунок 5).

Понятно, что построенный алгоритм ничего не сможет открыть на таких полях. Тем не менее, видно, что некоторые клетки совершенно точно или заминированы, или безопасны. На рисунке 5 *а* мина для B1 может находиться в A1 или в A2 и нигде больше, а эти клетки влияют на B2: если в A1–A2 точно находится мина, то в A3 точно безопасно (рисунок 6). Похожим образом открываются клетки на рисунке 5 *б*; для C2 возможны только три варианта расстановки мин (B1–C1, C1–D1 и B1–D1), и только один вариант правильный, так как в остальных двух превышается число расставленных мин для клеток B2 и C2 (рисунок 7). И, наконец, на рисунке 5 *в* клетка B1 заминирована: единица из A3 позволяет ставить в B2–B3 только одну мину, а двойке на A2 нужна еще одна, место для которой остается лишь в B1.

Такие ситуации довольно часты, они формируют шаблоны расстановки мин на поле. Такими шаблонами почти бессознательно и мыслят профессиональные игроки в «Сапера», автоматически, даже не задумываясь, щёлкая по клеткам. Однако человеческий способ мышления отличается от компьютерных операций: гораздо проще и понятнее действовать методом предположений, дающим тот же результат. Предположим, что в данной закрытой клетке стоит мина (или её там *точно* нет). Нас интересует ситуация, приводящая к ошибке, тогда можно утверждать о ложности предположения и присвоить соответственный статус клетке: можно открыть или там мина. Схематичный алгоритм представлен на листинге 3.

Вообще, и такой алгоритм не решит «Сапёра» во всех случаях, в частности, потому, что есть такие неразрешимые ситуации, как на рисунок 9. Как правильно расставить здесь две мины, непонятно.

Все гениальное – просто, и в пользу гениальности этой простой на первый

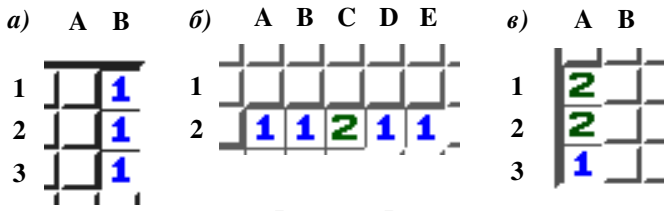


Рисунок 5.

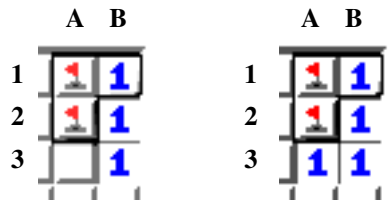


Рисунок 6.

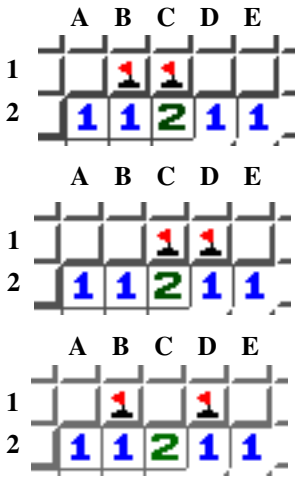


Рисунок 7.

взгляд игры говорит тот факт, что решение «Сапёра» является NP-полной задачей [6]. Из-за этого, а также из-за возможности рассмотренных выше неразрешимых случаев (и других, более сложных, но базирующихся на идее существования нескольких правильных расстановок мин [7]) создание алгоритма, дающего однозначный и правильный ответ, не представляется возможным [8]. К тому же только настоящие сапёры ошибаются один раз, а игроки, даже профессиональные, часто вынуждены действовать наугад, оценивая вероятность ошибки [9].

...гораздо проще и понятнее действовать методом предположений...

Как бы то ни было, угадывание выходит за границы компетенции программы. Остаётся вопрос о пользовательском интерфейсе или, точнее, о том, как связать игру в Windows с написанным алгоритмом.

Очевидное решение – вводить в некую матрицу цифры с поля «Сапёра», потом в игре открывать ставшие известными после прохода алгоритма клетки и снова вводить новую информацию. В некоторых случаях возложение этой обязанности на пользователя может быть выгодным, например, когда нужна лишь подсказка и нет необходимости переписывать все поле. Но гораздо лучше этот процесс скрыть и автоматизировать.

Средства Win32 API позволяют любому запущенному приложению получить информацию об окнах, формах, полях, кнопках и т. д. другого приложения. К сожалению, оказалось, что «Сапёр» защищён от подобного рода вторжений. Значит, нужно делать снимок экрана, обрабатывать полученное графическое изображение и формировать матрицу внутреннего представления.

Поле игры имеет четкую структуру, поэтому выявление количества клеток не представляет большой проблемы. Также выяснилось, что если наложить цифры из клеток друг на друга, то на их пересечении останется только два пикселя, по цвету которых и можно ставить соответствующее число в матрицу.

Если мы можем считать клетки из поля игры, то хотелось бы автоматически их и

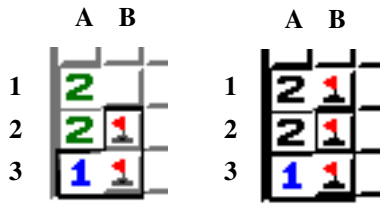


Рисунок 8.

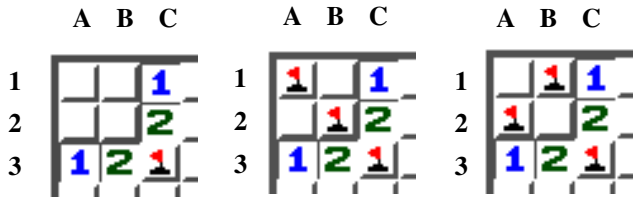
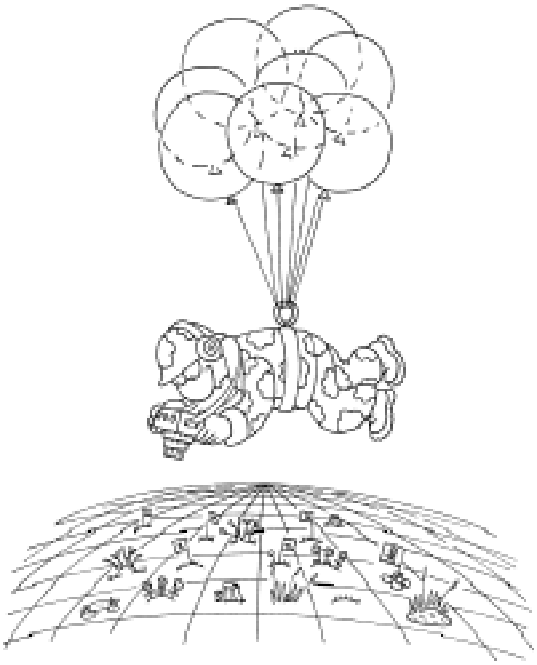


Рисунок 9.

Листинг 3. Алгоритм с предположениями

```
3.1  store(field);
3.2  change(field); //находим клетку для предположения
3.3  пытаемся решить задачу с таким измененным полем
3.4  restore(field);
3.5  if Error then
3.6    измененной в начале клетке приствиваем соответственный статус
3.7  fi
```



...нужно делать снимок экрана...

открывать. По ходу решения формируется список из найденных клеток, которые потом преобразуются в координаты на экране. В систему с заданной периодичностью посылаются сообщения о перемещении указателя в заданную точку и о щелчке мыши, что и представляет собой механизм открытия с различной скоростью. Конечно, удобнее и приятнее двигаться по контуру «островов» в минном поле, а не обходить всю матрицу в процессе решения несколько раз. Однако на интересующих нас сравнительно небольших размерах поля игры такой алгоритм даст весьма сомнительный прирост скорости, ведь фотографирование экрана – большое событие, и хотелось бы максимизировать выполняемую работу, то есть открывать каждый раз как можно больше. Вообще говоря, формирование сценария щелчков мыши является задачей о торговце, ведь без упорядочивания курсор бы беспорядочно прыгал по полю; в данном случае полное решение такой задачи совсем не нужно, достаточно ограничиться нахождением локально самой близкой следующей открываемой клетки.

В итоге получилась довольно забавная презентационная программа, автоматически и с различной скоростью решающая игру «Сапёр»: теперь кто-либо из друзей вряд ли побьет мои рекорды.

Комаров Алексей Дмитриевич,
студент 4 курса математико-механического
факультета СПбГУ,
кафедра информатики.