Статья опубликована в журнале «Мир ПК», 2004, № 5, с.64-71.

«Цветные» клеточные автоматы, или клонирование Мона Лизы

Лев Наумов, Анатолий Шалыто
levnaumov@mail.ru

Настоящая работа является продолжением статьи [1]. В ней рассматривались автоматы с клетками, каждая из которых может быть в одном из двух состояний («черно-белые» клеточные автоматы). Такие автоматы, в частности, обеспечивают самовоспроизведение. Этот и другие эффекты будут продемонстрированы на автоматах из клеток с большим числом состояний, каждому из которых может быть поставлен в соответствие цвет спектра. Автоматы этого типа назовем «цветными».

- Белый – это цвет?
- Да.
- Черный – это цвет?
- Да.
- Почему ты тогда говоришь,
  что это – не цветной телевизор.

Фольклор

Наряду с клеточными автоматами, рассмотренными в работе [1], существуют автоматы [2], клетки которых могут находиться более чем в двух состояниях. При этом каждой клетке соответствует целочисленная переменная, кодирующая ее состояние в данный момент времени. В свою очередь, в компьютерных системах цвет, как правило, представляется целым числом. Таким образом, каждому состоянию клетки автомата может быть поставлен в соответствие цвет спектра.

Связь между цветами и целыми числами обеспечивается разными способами, например, RGB-кодированием [3], при котором число, соответствующее цвету составляется из трех последовательных байт (дескриптор цвета), показывающих насыщенность синего (Blue), зеленого (Green) и красного (Red) цветов.

При этом необходимо отметить, что, несмотря на то, что хранение такого дескриптора требует 24-х битов (три байта), представляющий его тип данных занимает 32 бита (четыре байта). Это, отчасти, связано с необходимостью выравнивания адресов по степеням двух [4]. Например, в Visual C++ тип данных для цветового дескриптора имеет имя *COLORREF* [5]. «Лишний» байт обычно называют «байт прозрачности», и по этой причине он не рассматривается.

В результате изображение, в котором цвет каждого пикселя описывается 24-мя битами можно, рассматривать, как 24 битовые плоскости. Каждая из этих плоскостей является решеткой двумерного битового клеточного автомата. Такие автоматы описаны в работе [1].

При визуализации будем изображать единую «цветную» решетку. Цвет каждой ее клетки последовательно «собирается» из битов состояний клеток 24-х битовых решеток, имеющих одно и то же положение с соответствующей цветной клеткой. Таким образом, 24-битовый цветовой дескриптор формируется из 24 последовательных битов.

Из изложенного следует, что в любом случае совокупность 24 битовых клеточных автоматов может быть заменена одним автоматом, множество состояний каждой клетки которого совпадает с множеством значений цветового дескриптора. Это означает, что каждая клетка может быть в одном из 2^{24} состояний, закодированных числами от 0 до $2^{24}–1$.

Ниже будут рассмотрены клеточные автоматы с принципиально различными функциями переходов [1, 3].

В трех первых примерах множество состояний каждой клетки много меньше множества значений цветового дескриптора, поэтому ее функция переходов может быть задана с помощью набора из нескольких условий.

Четвертый и пятый примеры характеризуются тем, что каждый из 24-х битовых автоматов, составляющих «цветной» клеточный автомат, должен быть рассмотрен отдельно. Поэтому в функции переходов производится разбиение значения цветового дескриптора на биты. В шестом примере отдельно рассматриваются только первый и второй биты цветового дескриптора.

Однако, иногда удается записать функцию переходов клетки с помощью арифметических, логических или поразрядных операций над 24-битовыми целыми числами, не прибегая к разбиению и анализу отдельных битов. В седьмом примере показано, что с помощью поразрядной операцией «сумма по модулю два» над значениями переменных, кодирующих состояния самой клетки и четырех ее соседей, имеющих с ней общие стороны (так называемых, «главных» соседей) можно реализовать клонирование данных из некоторой области решетки («генерировать копии самих себя» [6]). Для наглядности в качестве начальных условий авторами выбрано изображение Мона Лизы, клонирование которой не является пока нарушением Российского законодательства ☺.

Реализация «цветных» клеточных автоматов

Программа, визуализирующая работу «цветного» клеточного автомата, разработанная в настоящей работе, имеет следующую структуру.

1. Вводится два массива дескрипторов цвета для хранения состояний клеток. Первый из них содержит текущее состояние каждой клетки автомата, а второй предназначен для хранения нового ее состояния.
2. Определяется функция переходов клетки решетки. В качестве параметров в функцию переходов передаются текущие значения состояний клеток окрестности, возможно, включая ее саму. Эта функция может быть задана в виде выражения над дескрипторами цвета, как целыми числами. С другой сторо-

ны, при вычислении этой функции может возникнуть необходимость рассмотрения каждой составляющей плоскости битов отдельно.

3. На нулевом шаге производится заполнение решетки (первого массива) начальными данными.
4. Для вычисления новых состояний вводится цикл. На каждой итерации для каждой клетки, используя в качестве аргументов функции переходов элементы первого массива, вычисляется ее новое состояние, которое помещается во второй массив.
5. После завершения итерации значения из всех элементов второго массива переносятся в первый. Этим обеспечивается псевдопараллельное изменение значений состояний всех клеток решетки.
6. Осуществляется визуализация содержимого первого массива.
7. Пользователь с помощью соответствующих элементов управления либо обеспечивает переход к следующему шагу (пункту 4), либо прекращает работу программы.

На рис. 1 показан пользовательский интерфейс разработанной программы.

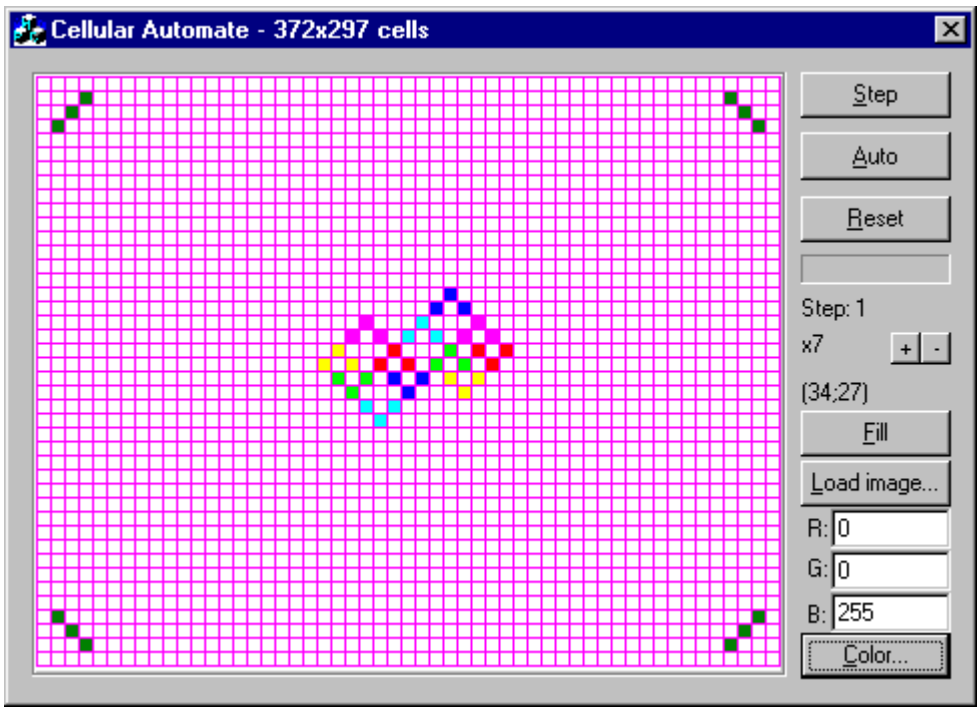

Рис. 1. Пользовательский интерфейс программы

Приведем назначение элементов управления (сверху вниз, слева направо):

- кнопка «Step» – обеспечивает выполнение одного шага клеточного автомата;
- кнопка «Auto» – переводит программу в режим автоматического выполнения шагов и выводит из него при повторном нажатии;
- кнопка «Reset» – обеспечивает восстановление начального состояния программы и моделируемого клеточного автомата (инициализация);
- индикатор выполнения процесса – показывает течение выполнения текущей задачи (шага, инициализации и т.п.);
- поле «Step» – отображает номер текущего шага клеточного автомата;

- поле для отображения текущей степени увеличения (размера представления клеток решетки);
- кнопки «+» и «–» – увеличивают и уменьшают размер представления клеток решетки. При увеличении более чем в пять раз отображается вспомогательная сетка;
- поле для отображения координат клетки решетки, над которой в данный момент находится указатель мыши;
- кнопка «Fill» – обеспечивает заполнение всех клеток решетки текущим цветом;
- кнопка «Load image...» – обеспечивает загрузку из файла и заполнение клеток решетки битовым преставлением обрабатываемого графического изображения. Маленькое изображение на решётке будет размещено по центру;
- поля ввода «R», «G» и «B» – позволяют покомпонентно задавать текущий цвет;
- кнопка «Color» – позволяет выбрать текущий цвет с помощью диалога.

Посредством мыши можно осуществлять следующие операции:
- щелчком левой кнопки при нахождении курсора на области, на которой отображается клетка решетки, осуществляется переход клетки в состояние, соответствующее текущему цвету;
- перемещение мыши при нажатой правой кнопке позволяет изменять отображаемую область решетки, если степень увеличения больше единицы.

На основе изложенного авторами разработаны программы на языке C++, иллюстрирующие каждый из приводимых ниже примеров. Эти программы можно загрузить с сайта http://is.ifmo.ru (раздел «Статьи»).

Все программы однотипны и отличаются только (кроме примера 6) двумя функциями, специфичными для каждого конкретного автомата. Ими являются функция *init()*, задающая начальные условия, и функция переходов *f(y, yU, yUR, yR, yDR, yD, yDL, yL, yUL)*, которые принадлежат классу *CCADlg*. При этом ниже для каждого примера будут приведены только эти две функции.

Примеры автоматов с функцией переходов, заданной в виде набора условий

Пример 1. В работе [7] для иллюстрации понятия клеточный автомат приведен пример «цветного» клеточного автомата, каждая клетка которого может быть в одном из четырех состояний, закодированных числами (цветами): 0 (белый), 1 (красный), 2 (зеленый) и 3 (синий). Окрестность каждой клетки составляют четыре ее главных соседа. Функция переходов в этом случае задается следующими правилами:
- если клетка находится в состоянии 0 и лишь одна из окрестных клеток находится в состоянии 1, то она переходит в состояние 2;
- если клетка находится в состоянии 0 и лишь одна из окрестных клеток находится в состоянии 2, то она переходит в состояние 3;
- если клетка находится в состоянии 0 и все четыре окрестных клетки находятся в состоянии 3, то она переходит в состояние 3;

- если клетка находится в состоянии 1 и хотя бы одна из окрестных клеток находится в состоянии 3, то она переходит в состояние 3;
- если клетка находится в состоянии 2 и хотя бы одна из окрестных клеток находится в состоянии 3, то она переходит в состояние 3.

Введем массив *s[4]*, который задает соответствие между номерами состояний и цветовыми дескрипторами, определяющими белый, красный, зеленый и синий цвета:

```
COLORREF s[4]={RGB(255,255,255),RGB(255,0,0),RGB(0,255,0),
RGB(0,0,255)};
```

В качестве начальных условий выбран красный крест из пяти клеток на белом поле:

```
void CCADlg::init()
{
	for (WORD ww=0; ww<MAXX; ww++) for (WORD wh=0; wh<MAXY; wh++) {
		Field[ww][wh]=s[0];
	}

	Field[170][150]=s[1];
	Field[171][150]=s[1];
	Field[169][150]=s[1];
	Field[170][151]=s[1];
	Field[170][149]=s[1];
}
```

Для вычисления функции переходов введем вспомогательную функцию *count(y1 ,y2 ,y3 ,y4 ,s)*, позволяющую определить, сколько клеток из *y1*, *y2*, *y3* и *y4* находится в состоянии *s*:

```
inline BYTE count(COLORREF y1,COLORREF y2,COLORREF y3,COLORREF
y4,COLORREF s)
{
	BYTE b=0;
	if (y1==s) b++;
	if (y2==s) b++;
	if (y3==s) b++;
	if (y4==s) b++;
	return b;
}
```

При этом функция переходов может быть записана следующим образом:

```
COLORREF CCADlg::f(COLORREF y, COLORREF yU, COLORREF yUR, COLORREF
yR, COLORREF yDR, COLORREF yD, COLORREF yDL, COLORREF yL, COLORREF
yUL)
{
	if (y==s[0]) {
		if (count(yU,yR,yD,yL,s[1])==1) return s[2];
		if (count(yU,yR,yD,yL,s[2])==1 ||
			count(yU,yR,yD,yL,s[3])==4) return s[3];
	}
	if (y==s[1]) {
		if (count(yU,yR,yD,yL,s[3])>=1) return s[3];
```

```
        }
        if (y==s[2]) {
              if (count(yU,yR,yD,yL,s[3])>=1) return s[3];
        }
        return y;
}
```

При использовании приведенных выше функций автомат обладает весьма «скучным» поведением. После пятого шага состояние решетки не изменяется – автомат переходит в статическое состояние. Состояния решетки этого автомата изображены на рис. 2.

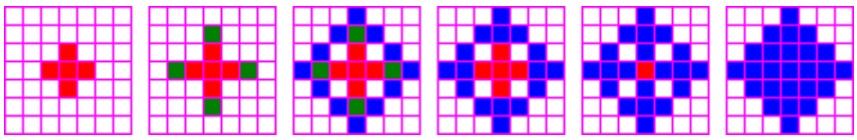

Рис. 2. Поведение автомата (пример 1 – простой крест)

Обратим внимание, что в работе [7] в пятом кадре средняя клетка ошибочно изображена синей.

Пример 2. Отметим, что добавление к правилам лишь одного условия позволяет сделать поведение автомата более интересным – периодическим:

- если клетка находится в состоянии 3 и все четыре окрестных клетки находятся в состоянии 3, то она переходит в состояние 1.

Состояния решетки, получающиеся в этом случае, при тех же начальных условиях, что и в примере 1, приведены на рис. 3. Конфигурации, приведенные для шестого и седьмого шагов, повторяются и далее, сменяя друг друга. Таким образом, с шестого шага имеет место периодическое поведение с периодом два.

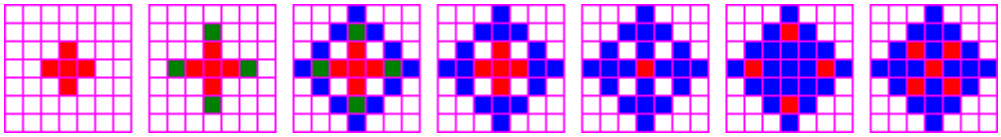

Рис. 3. Периодическое поведение автомата (пример 2 – периодический крест)

Пример 3. Если к правилам автомата из примера 1 добавить другое (относительно примера 2) условие, то автомат демонстрирует фрактальное поведение:

- если клетка находится в состоянии 3 и хотя бы одна из окрестных клеток находится в состоянии 1, то она переходит в состояние 1.

Состояния решетки, получающиеся в этом случае при тех же начальных условиях, что и в примере 1, приведены на рис. 4.

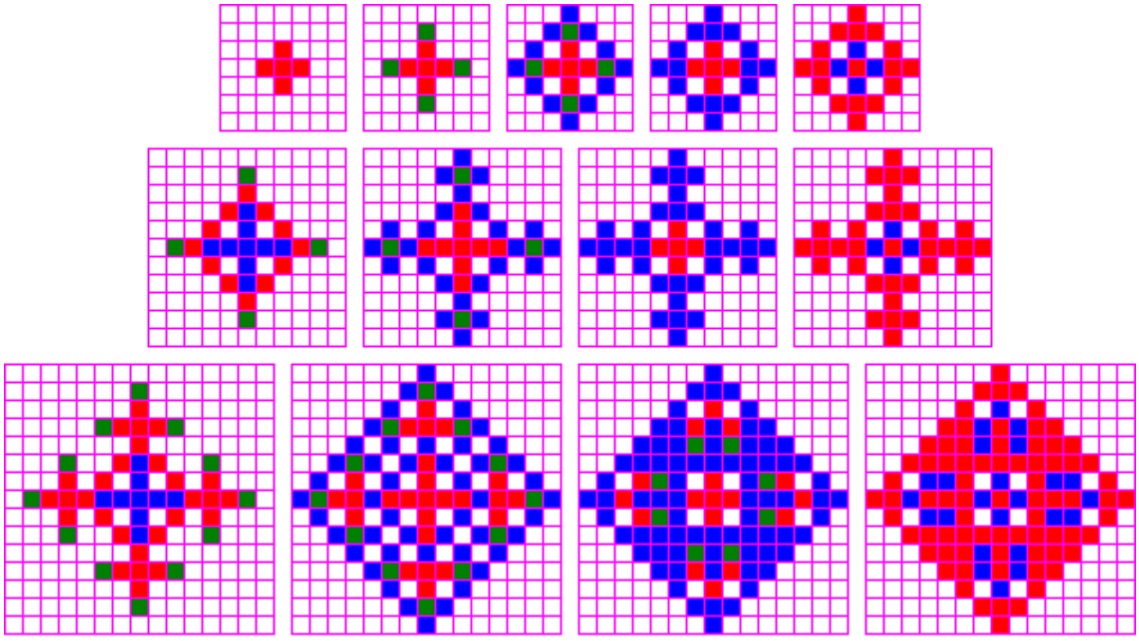

Рис. 4. Фрактальное поведение автомата (пример 3 – фрактальный крест)

Фрактальное поведение в рассматриваемом примере состоит в следующем: на красных окончаниях появляются зеленые «хвостики», которые превращаются в синие «цветы». Они «созревая», становятся красными, образуя, тем самым, новые окончания.

Цветная «Жизнь»

Рассмотрим двумерную решетку, каждая клетка которой может содержать 24-битовое число, как 24 отдельные битовые плоскости. На каждой из этих плоскостей реализуем игру «Жизнь» [1, 8]. При этом функция переходов должна быть написана так, чтобы каждая плоскость рассматривалась независимо от других:

```
COLORREF CCADlg::f(COLORREF y, COLORREF yU, COLORREF yUR, COLORREF
yR, COLORREF yDR, COLORREF yD, COLORREF yDL, COLORREF yL, COLORREF
yUL)
{
	// Переменная содержит единицу только в бите, соответствующем
	// рассматриваемой плоскости
	COLORREF b=1;

	// Переменная, в которой сохраняется число живых соседей
	// клетки на рассматриваемой плоскости
	int i;

	// Цикл для перебора всех 24 плоскостей
	for (BYTE it=0;it<24;it++) {
		// Вычисление числа соседей
		i=((yU&b)?1:0)+((yUR&b)?1:0)+((yR&b)?1:0)+((yDR&b)?1:0)+
		((yD&b)?1:0)+((yDL&b)?1:0)+((yL&b)?1:0)+((yUL&b)?1:0);


		// Мертвая клетка оживет, если у нее три живых соседа
		if ((y&b)==0 && i==3) y|=b; else
```

```
            // Живая клетка останется живой, если у нее два
            // или три живых соседа
            if ((y&b)!=0 && (i==2 || i==3)) y|=b; else

            // В остальных случаях клетка мертва
            y&=(~b);

            // Рассматриваемая плоскость заменяется на следующую
            b<<=1;
      }

      return y;
}
```

Работу цветной игры «Жизнь» проиллюстрируем двумя примерами полетов планеров [8].

Пример 4. Пусть на решетке размещены три планера, ориентированных на столкновение. Они не должны иметь общих единичных битов в числовых представлениях цвета. Поэтому выберем следующие цвета (в RGB-представлении): красный (255, 0, 0), зеленый (0, 255, 0) и синий (0, 0, 255).

Цветной планер представляет собой «эскадрилью» из восьми «классических» планеров [8], каждый из которых летит в своей плоскости.

Это, а также указанный выбор цветов, позволяют добиться того, что никакие два из 24 планеров не будут «лететь» в одной плоскости. В результате, при столкновении цветные планеры пролетят друг сквозь друга.

Полет цветных планеров изображен на рис. 5. На этом рисунке приведены кадры, соответствующие начальному, восьмому, двенадцатому и двадцать восьмому шагам автомата.

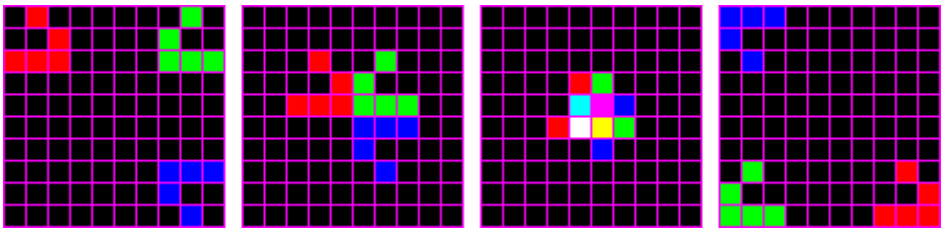

Рис. 5. Полет цветных планеров (пример 4)

Пример 5. Рассмотрим столкновение двух цветных планеров, в результате которого «выживает» лишь один из них, изменив при этом цвет.

Разместим на плоскости желтый (255, 255, 0) и красный (255, 0, 0) планеры, ориентированные на столкновение. Желтый планер представляет собой эскадрилью из 16 планеров, верхние восемь из которых сталкиваются с эскадрильей красного планера.

В результате из желтого планера получим зеленый (0, 255, 0), а также «обломки» красного цвета.

Полет цветных планеров, рассматриваемых в настоящем примере, изображен на рис. 6. На этом рисунке приведены кадры, соответствующие начальному, восьмому, двенадцатому и двадцать восьмому шагам автомата.

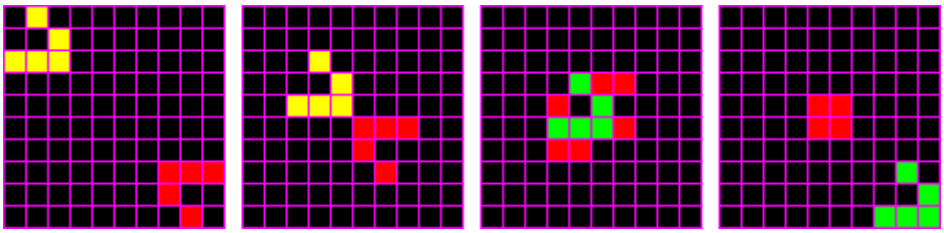

Рис. 6. Полет цветных планеров (пример 5)

«Жизнь с историей»

Как отмечалось выше, в цветных клеточных автоматах рассматриваемого типа используется 24 битовых плоскости. Реализуем в первой из них классическую игру «Жизнь» [1, 8]. При этом результат каждой предыдущей итерации будем переносить на следующую по порядку битовую плоскость. Таким образом, автомат будет «помнить» историю последних 32-х итераций, а отображать историю последних 24-х.

Ввиду того, что цветовой дескриптор хранит информацию в порядке синий, зеленый, красный, то «взведенному» биту на первой плоскости соответствует темно-синий цвет (0, 0, 128 в RGB-представлении).

Пример 6. Для того, чтобы реализовать игру «Жизнь с историей» в визуализирующей программе произведено существенное изменение, состоящее в том, что перед каждой следующей итерацией осуществляется сдвиг всех значений состояний, полученных на предыдущей итерации, на один бит вправо. Эту операцию невозможно выполнить в функции переходов, и поэтому она выполняется отдельно.

Функция переходов разместит информацию о текущей итерации на первой плоскости. При этом информацию о предыдущей итерации она получит со второй плоскости. Эта функция может быть реализована следующим образом:

```
COLORREF CCADlg::f(COLORREF y, COLORREF yU, COLORREF yUR, COLORREF
yR, COLORREF yDR, COLORREF yD, COLORREF yDL, COLORREF yL, COLORREF
yUL)
{
	// Переменная содержит единицу только в бите, соответствующем
	// первой плоскости
	COLORREF c=0x800000;

	// Переменная содержит единицу только в бите, соответствующем
	// второй плоскости
	COLORREF b=0x400000;

	// Переменная, в которой хранится число живых соседей
	// клетки на второй плоскости
```

```
        int i=((yU&b)?1:0)+((yUR&b)?1:0)+((yR&b)?1:0)+((yDR&b)?1:0)+
        ((yD&b)?1:0)+((yDL&b)?1:0)+((yL&b)?1:0)+((yUL&b)?1:0);

        // Мертвая клетка оживет, если у нее три живых соседа
        if ((y&b)==0 && i==3) y|=c; else

        // Живая клетка останется живой, если у нее два
        // или три живых соседа
        if ((y&b)!=0 && (i==2 || i==3)) y|=c; else

        // В остальных случаях клетка мертва
        y&=(~c);

        return y;
}
```

В качестве начальных условий выберем планерное ружье [8] белого цвета (255, 255, 255). Результат его эволюции после 239 итераций приведен на рис. 7.

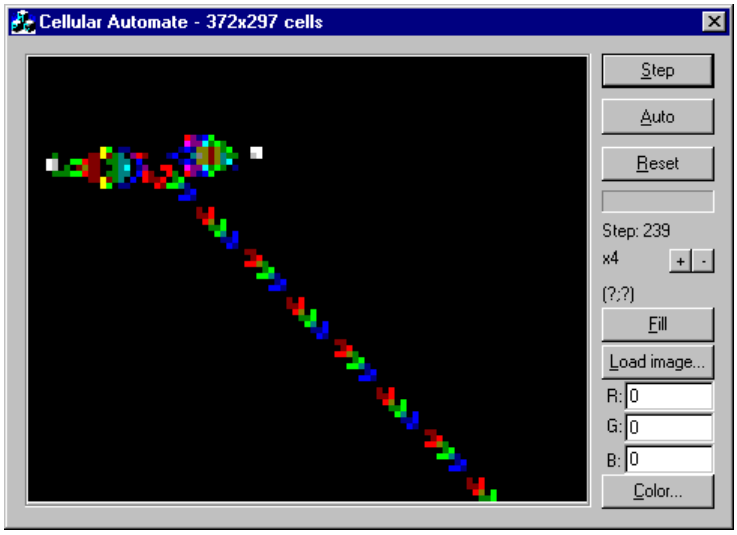

Рис. 7. Жизнь с историей на примере эволюции планерного ружья (пример 6)

Клетки, имеющие красные составляющие, были живы от 17-ти до 24-х итераций назад, имеющие зеленые составляющие – от 8-ми до 16-ти, а синие – от настоящего момента до 7-ми итераций назад.

В заключение раздела отметим, что «Жизнь с историей» дает возможность исследования поведения популяций в классической игре «Жизнь» [1, 8]. Она позволяет наглядно выявлять некоторые статические, периодические и вымирающие участки популяций. Так в обычной игре «Жизнь» в текущей момент времени можно было бы наблюдать лишь то, что происходит в цветной игре на первой битовой плоскости.

Клонирование Мона Лизы

Разработанная визуализирующая программа позволяет, кроме изложенного, воспроизводить (но не подделывать!) произведения искусства. Покажем, как

клонируются мировые шедевры на примере картины Леонардо да Винчи «Мона Лиза».

Пример 7. Выберем в качестве функции переходов операцию «сумма по модулю два» значений переменных, кодирующих состояния самой клетки и четырех ее главных соседей.

Эта функция имеет вид:

```
COLORREF CCADlg::f(COLORREF y, COLORREF yU, COLORREF yUR, COLORREF
yR, COLORREF yDR, COLORREF yD, COLORREF yDL, COLORREF yL, COLORREF
yUL)
{
      return y^yU^yR^yD^yL;
}
```

При помощи кнопки «Load Image...» загрузим bmp-файл, хранящий картину размером 41 на 64 пикселей (рис. 8).

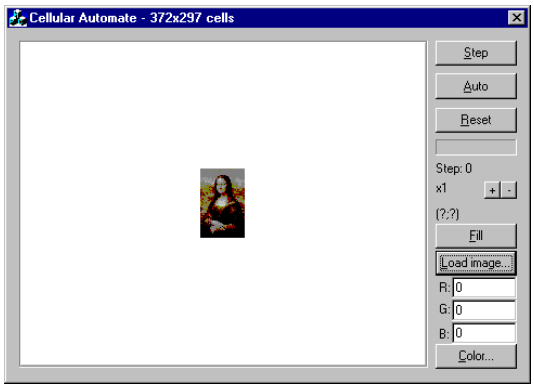

Рис. 8. Мона Лиза. Шаг 0

На двенадцатом шаге (рис.9) Мона Лизу уже не узнать (бедная Лиза ☺!).

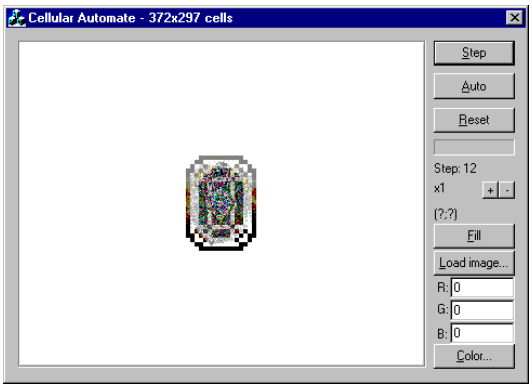

Рис. 9. Мона Лиза. Шаг 12

На 32-ом шаге (рис.10) очертания пяти Мона Лиз становятся очевидными.

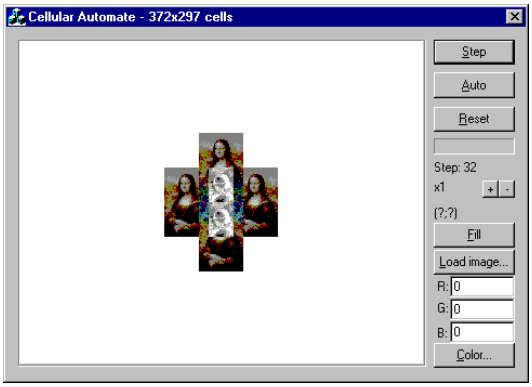

Рис. 10. Мона Лиза. Шаг 32

На 64-ом шаге (рис.11) клонирование завершено – получено пять Мона Лиз. В работе [1] приведено соотношение, показывающее, что клонирование должно завершиться именно на этом шаге.

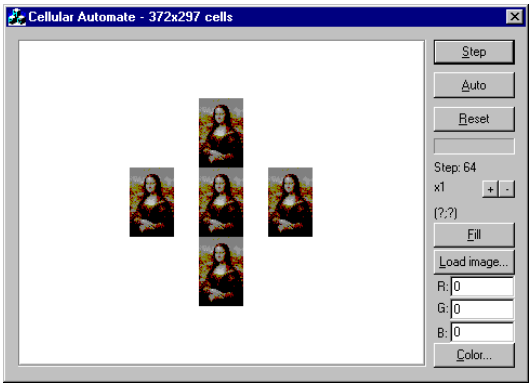

Рис. 11. Мона Лиза. Шаг 64

На 127-ом шаге имеет место состояние решетки, которое можно принять за хаотическое (рис.12).

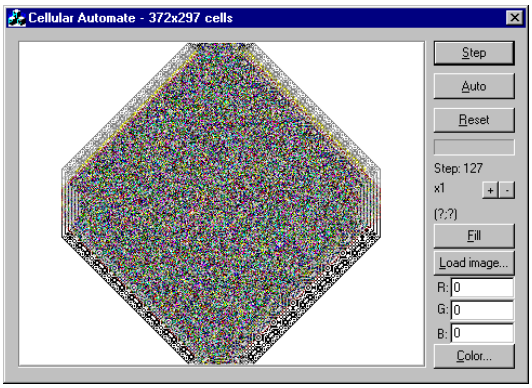

Рис. 12. Мона Лиза. Шаг 127

На следующем шаге появляется очередной результат клонирования – снова пять Мона Лиз (рис.13).

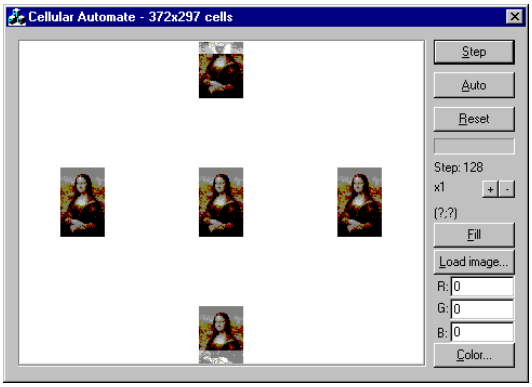

Рис. 13. Мона Лиза. Шаг 128

На этом рисунке, за счет малой размерности решетки и «заворачивания» ее в тор, верхняя и нижняя Мона Лизы помешали друг другу полноценно развиться.

Обоснование клонирования для случая битового автомата приведено в работе [1]. В «цветном» случае тот же эффект имеет место на каждой из 24 битовых плоскостей, что при визуализации дает цветное изображение.

Заключение

Настоящая работа является едва ли не единственной научно-популярной статьей, посвященной автоматам, клетки которых могут быть в большом числе состояний. При этом, если по книге М. Гарднера [8] даже школьники знают о черно-белой игре «Жизнь», то в настоящей работе «Жизнь» начинает играть новыми красками.

Множество задач, которые можно решать с помощью цветных клеточных необозримо. Например, обработка изображений, моделирование физических процессов, распознавание образов и многие, многие другие.

Разработанная программа представляет собой простейший пример средства для анализа поведения клеточных автоматов. В настоящее время авторами разрабатывается среда CAMEL (Cellular Automata Modeling Environment & Library – среда моделирования и библиотека разработчика клеточных автоматов), которая предоставит развитый инструментарий для решения задач с применением клеточных автоматов различных типов.

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту №02-07-90114 «Разработка технологий автоматного программирования».

Литература

1. *Наумов Л.А., Шалыто А.А.* Клеточные автоматы. Реализация и эксперименты // Мир ПК. 2003. №4.
2. *Тоффоли Т., Марголус Н.* Машины клеточных автоматов. М.: Мир, 1991.
3. *Троицкий Е.* Цветовые модели // Программист. 2003. №1.

4. *Гордеев А.В., Молчанов А.Ю.* Введение в системное программирование. СПб.: Питер, 2001.
5. *Мешков А.В., Тихомиров Ю.В.* Visual C++ и MFC. СПб.: БХВ-Санкт-Петербург, 2000.
6. *Гурский Д., Стрельченко Ю.* Оптимистичный футуризм // Мир ПК. 2003. №1.
7. Информатика. Энциклопедический словарь для начинающих. – М.: Педагогика-пресс, 1994.
8. *Гарднер М.* Крестики нолики. М.: Мир, 1992.

Об авторах

Наумов Лев Александрович – студент кафедры «Компьютерные технологии» Санкт-Петербургского государственного университета информационных технологий, механики и оптики – СПбГУ ИТМО. E-mail: levnaumov@mail.ru.

Шалыто Анатолий Абрамович – профессор кафедры «Компьютерные технологии» СПбГУ ИТМО. E-mail: shalyto@mail.ifmo.ru. Сайт: http://is.ifmo.ru.