Санкт-Петербургский государственный институт точной механики и оптики
(технический университет)
Кафедра «Компьютерные технологии»

Л.А. Наумов

Научный руководитель: Шалыто А.А., д.т.н., профессор

Бакалаврская работа

Разработка среды и библиотеки CAME&L для решения задач с использованием клеточных автоматов

Санкт-Петербург
2003

Содержание

ВВЕДЕНИЕ

На протяжении всей своей истории человечество постоянно генерировало какие-то идеи, обеспечивающие его развитие. Делались открытия и появлялись изобретения в самых разных отраслях жизни. Каменный топор, колесо, бумага, пенициллин, автомобиль – предметы, вещества и соответствующие им понятия, без которых развитие цивилизации представить невозможно. Истина, бог, время, функция – абстрактные понятия, возникшие в процессе развития человечества. Все они возникали тогда, когда знания и потребности человека достигали определённого уровня. По всей видимости, в конце 60-х годов XX века понятие «клеточный автомат» было просто необходимо для дальнейшего прогресса физики, биологии, химии, математики, компьютерных наук и других областей знаний, так они изобретались многократно под разными названиями, в разных странах, людьми, работающими в разных областях знаний. Тем не менее, концептуально возникшие понятия были практически эквивалентны. «Итеративные массивы», «вычисляющие пространства», «однородные структуры» и «клеточные автоматы» являются синонимами.

Когда Станислав Улам (Stanislaw Ulam) начинал свои исследования [1] он, наверное, не предполагал, что, основываясь на его идеях, выдающийся американский учёный Джон фон Нейман (John von Neumann) придумает концепцию клеточных автоматов. Хотя фон Нейман был математиком и физиком, эта идея пришла к нему при решении задач из области биологии. Он использовал клеточные автоматы для создания более правдоподобных моделей пространственно протяжённых систем. Результаты его исследований приведены в фундаментальной работе [2]. В оригинале эта работа вышла в 1966 году, в то время как фон Нейман умер в 1957. Дописывал её Артур Бёркс (Arthur Burks), ставший известным специалистом по клеточным автоматам.

Однако вернёмся назад во времени. В конце Второй Мировой Войны, в то время как фон Нейман создавал один из первых электронных компьютеров, немецкий инженер Конрад Цузе (Konrad Zuse) прятался от нацистов в австрийских Альпах. Там, в уединении, у него возникло множество идей из области параллельных вычислений. Среди прочего он придумал и «вычисляющие пространства» [3], то есть клеточные автоматы. Особый интерес Цузе вызывало их применения к задачам численного моделирования в механике. К сожалению политическая обстановка помешала работам учёному стать известными в то время. За работами же фон Неймана следил весь научный мир.

Профессиональные математики пришли к клеточным автоматам, рассматривая итерационные преобразования пространственно распределённых структур с дискретным набором состояний [4]. Сразу стали возникать решения важных теоретических задач в этой области, например, вопросов обратимости и вычислимости. В группе компьютерной логики университета штата Мичиган Джон Холланд (John Holland) применял клеточные автоматы к решению задач адаптации и оптимизации [5].

Однако настоящий эффект разорвавшейся бомбы произвела статья ведущего рубрики математических игр и головоломок журнала «Scientific American» Мартина Гарднера (Martin Gardner) [6-8]. Он опубликовал описание клеточного автомата Джона Хортона Конвея (John Horton Conway) «Жизнь». Игра «Жизнь», как стали называть этот автомат, фактически, стала культовой и сделала понятия «клеточный автомат» чрезвычайно популярным особенно среди людей с техническим образованием.

Область применения клеточных автоматов чрезвычайно широка [9, 10]. Однако необходимо отметить, что в подавляющем большинстве случаев, они применяются либо в качестве параллельных вычислительных сред, либо в качестве сред моделирования пространственно распределённых систем (например, физических, биологических, химических, социологических и других).

Многофункциональная среда моделирования клеточных автоматов позволила бы использовать вычислительную машину в качестве:

- подчас весьма дорогостоящей экспериментальной установки для физических, биологических, химических и прочих опытов;
- средства реализации и визуализации алгоритмов параллельных вычислений.

Настоящая работа посвящена разработке среды CAME&L (Cellular Automata Modeling Environment & Library – среда моделирования и библиотеки разработчика клеточных автоматов), которая представляет собой мощное средство для решения задач с использованием клеточных автоматов. В работе рассматривается вопрос о том, почему возникла необходимость написания этой среды, производится её сравнение с существующими аналогами. Приводятся основные требования, которые можно предъявить к программному обеспечению этого класса, принципиальные вопросы, которые возникают при его разработке и то, как они решены в проекте CAME&L. Подробно описывается часть среды, на-

званная CADLib (Cellular Automata Developers LIBrary – библиотека разработчика клеточных автоматов), служащая непосредственно для создания клеточных автоматов.

В первой главе приводится законченный, удобный, богатый формализм клеточных автоматов. Необходимо отметить, что строгого определения этому понятию, насколько известно автору, ещё не было дано. Математический аппарат, который здесь строится, во многом определяет основные понятия и характеристики среды CAME&L.

Во второй главе перечисляются задачи, которые ставятся перед программным обеспечением, служащим для создания моделей с использованием клеточных автоматов. Приводится перечень и описание существующих продуктов, разработанных для этих целей, а также описываются основные возможности программного обеспечения CAME&L.

В третьей главе описывается одна из важнейших частей программного обеспечения CAME&L, библиотека классов CADLib. Она предназначена для разработки клеточных автоматов при решении конкретных задач. Помимо классов в библиотеке содержатся функции, константы и макроопределения, упрощающие работу пользователя-исследователя.

ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ КЛЕТОЧНЫХ АВТОМАТОВ

Прежде, чем дать формальное определение клеточным автоматам, в настоящей главе обсуждается это понятие, не переходя на язык математики. В первую очередь даётся представление об их существенных составляющих и взаимосвязях между ними. На основании этого становятся ясными выразительные возможности клеточных автоматов.

1.1. Что такое клеточный автомат?

Клеточный автомат является дискретной динамической системой, поведение которой полностью определяется в терминах локальных зависимостей [9].

Назовём *дискретным пространством* пространство над дискретным множеством [11] элементов. Экземпляр пространства этого класса будем называть *решёткой* клеточного автомата, а каждый его элемент – *клеткой*. Каждая клетка характеризуется определённым значением из некого множества. О клетке говорят, что она *содержит* или *имеет* соответствующее значение, либо находится или пребывает в состоянии, кодируемом данным значением. Оно можем быть булевым, целым, числом с плавающей точкой, множеством или другим объектом, в зависимости от потребностей задачи.

Совокупность состояний всех клеток решётки называется *состоянием решётки*. Состояние решётки меняется в соответствии с некоторым законом, который называется *правилами* клеточного автомата. Каждое изменение состояния решётки называется *итерацией*. Время в рассматриваемой модели дискретно и каждая итерация соответствует некому моменту времени.

Правила определяют, какое значение должно содержаться в клетке в следующий момент времени, в зависимости от значений в некоторых других клетках в текущий момент, а также, возможно, от значения, содержащегося в ней самой в текущий момент. Если новое состояние клетки зависит от текущего её состояния, то о соответствующем клеточном автомате говорят, что он является *автоматом с клетками с памятью*, иначе – *автоматом с клетками без памяти*.

Множество клеток, влияющих на значение данной, за исключением её самой, называется *окрестностью* клетки. Окрестность клетки удобнее задавать, если на решётке вве-

сти метрику, поэтому далее, для удобства, будем говорить о решётке, как о дискретном метрическом пространстве.

Приведём пример клеточного автомата, который можно реализовать без использования вычислительной машины. Для этого необходимо взять пачку листов в клетку таких, чтобы при наложении одного листа на другой нижний слегка просвечивал. Кроме того, сетка, делящая лист на клетки, должна быть на каждом листе напечатана на одном и том же месте.

Допустим, что каждая клетка может содержать один бит информации. Это означает, что клетка может, например, быть либо помеченной, либо не помеченной. Пометим какие-либо клетки на первом листе, сформировав тем самым начальное состояние решётки. Для совершения итерации необходимо на первый лист положить второй и слегка обозначить (так, чтобы пометки на нижнем листе все же были видны) на верхнем листе те клетки, которые на предыдущем листе также были помеченными, если среди восьми их ближайших соседей найдутся две или три помеченные клетки. Также легонько обозначьте те клетки, которые на предыдущем листе были пустыми, если среди их соседей найдётся ровно три помеченные клетки.

Когда описанная процедура будет проделана для всех клеток верхнего листа, те из них, которые слегка выделены, можно пометить окончательно. Дальше эту процедуру можно повторять от листа к листу до тех пор, пока хватит терпения.

Таким образом, можно реализовать клеточный автомат «Жизнь», который обладает множеством интересных свойств и любопытнейшей историей [6-8].

Предложенная модель обладает всеми упомянутые выше необходимыми характеристиками клеточных автоматов. Во-первых, у неё есть решётка, составленная из квадратов. Во-вторых, на решётке определена окрестность. Для каждой клетки она представляет собой множество из восьми непосредственно примыкающих к ней соседей. В-третьих, определено множество состояний клетки. В данном случае это – множество эквивалентное множеству из двух элементов {«жива», «мертва»}. В-четвёртых, описаны правила, обладающие свойством локальности (на каждую клетку влияют только клетки окрестности). В-пятых, автомат работает итерационно. Результат каждой итерации находится на отдельном листе бумаги.

Отметим основные свойства классической модели клеточных автоматов.

1. *Локальность правил.* На новое состояние клетки могут влиять только элементы её окрестности и, возможно, она сама;
2. *Однородность системы.* Ни одна область решётки не может быть отличена от другой по каким-либо особенностям ландшафта, правил и т.п. Однако на практике решётка оказывается конечным множеством клеток (ведь не возможно выделить неограниченный объём данных). В результате могут иметь место краевые эффекты, клетки стоящие на границе решётки будут отличны от остальных по числу соседей. Во избежание этого можно ввести краевые условия, завернуть решётку в тор или, например, лист Мёбиуса.
3. *Множество возможных состояний клетки – конечно.* Это условие необходимо, чтобы для получения нового состояния клетки требовалось конечное число операций. Отметим, что оно не мешает использовать клетки для хранения чисел с плавающей точкой при решении прикладных задач. Например, переменная, имеющая тип данных `double`, может принимать, с точностью до знака, значения от $1.7 \cdot 10^{-308}$ до $1.7 \cdot 10^{+308}$ [12]. Однако множество её значений конечно, а не континуально. Так как её размер – 8 байт, то множество её значений имеет мощность 2^{64};
4. *Значения во всех клетках меняются единовременно, в конце итерации, а не по мере вычисления.* В противном случае порядок перебора клеток решётки, при совершении итерации, существенно влиял бы на результат.

Необходимо отметить, что на практике, при решении определённых задач, возникает потребность в том, чтобы отказаться от последних трёх свойств. Поэтому выше было оговорено, что это – свойства «классических» клеточных автоматов. Этот вопрос будет подробнее обсуждаться в разделе 1.4.1.

1.2. Задачи для клеточных автоматов

Каждый клеточный автомат, это – некий мир, живущий по определённым законам. Жизнь его, как замкнутой системы, бесцельна. Время идет, он меняется, но сам по себе он не в силах понять, зачем это нужно и как долго это ещё будет продолжаться. Существование мира обретает смысл, когда кто-то смотрит на него из другого мира и делает определённые выводы из процесса его эволюции.

Клеточные автоматы можно рассматривать не как замкнутые в себе миры, а как миры-генераторы определённых выходных сигналов, в ответ на входные, предоставляя, тем самым, внешнему миру дополнительные рычаги управления и дополнительную информацию о поведении автомата. Такие клеточные автоматы получили название *автоматов-трансдюсеров*, то есть – преобразователей входных сигналов в выходные. Подробнее такие автоматы рассматриваются в разделе 1.4.3.

Как уже отмечалось выше, одной из основных областей применения клеточных автоматов является создание физических, биологических, химических и прочих моделей. Клеточный автомат является аналогом понятия поля и оказывается идеальной средой для решения дифференциальных уравнений и уравнений в частных производных, например, уравнения Навье-Стокса, уравнения теплопроводности и волнового уравнения. Дифференциальные уравнения описывают непрерывные динамические системы, а клеточные автоматы, как было сказано выше, также являются динамическими системами, но дискретными. Они представляют собой простые, удобные и точные модели макро и микромира, эволюционных процессов, динамики жидкостей, турбулентности, фрактальности, хаотичности, упорядоченья и т.д.

Часто задавался вопрос о том, могут ли клеточные автоматы моделировать непосредственно законы физики, а не некие аспекты их проявления в природе. Камнем преткновения стал вопрос обратимости пути развития, основного свойства микроскопической физики.

При решении такого рода задач решётка становится пространством для моделирования некой среды. Пусть на рис. 1.a изображено некое вещество. В задаче существенен какой-то его параметр. Зоны, обладающие одинаковым значением этого параметра, выделены одним и тем же цветом. Чтобы решать задачу численно необходимо как-то описать для машины исследуемый материал, сообщить распределение значений параметра. Для этого следует произвести дискретизацию, заменить непрерывный массив данных конечным числом информативных значений.

Наложим на вещество решётки, составленные из квадратов и шестиугольников (рис. 1.b и 1.d, соответственно). В каждую клетку поместим значение, соответствующее той зоне материала, которую она охватывает. Если это – некий числовой параметр, то в ячейку можно записать его среднее значение по всем точкам, охватываемым клеткой или,

как в настоящем примере, значение, которым обладает большинство точек, охватываемых ею (рис. 1.c и 1.e, соответственно). Возможны и другие способы определения значения, которое следует поместить в клетку, соответствующую некой области среды.

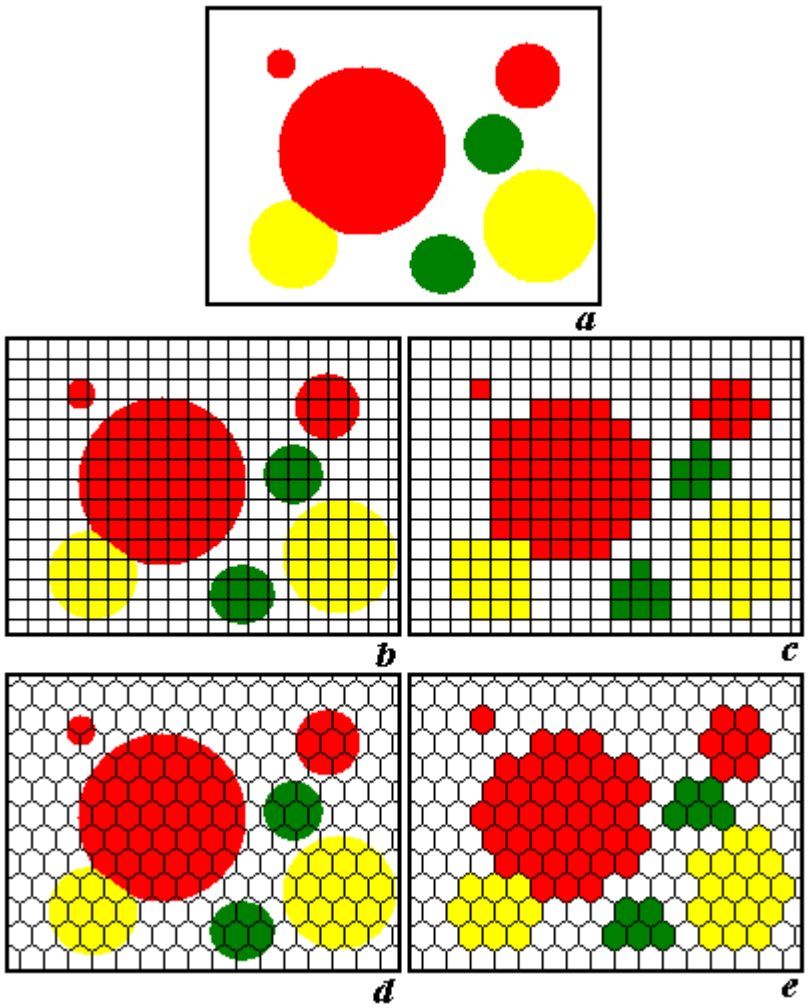

Рис 1. Дискретизация на примере квадратной и гексагональной решёток

Основной интерес со стороны специалистов области компьютерных наук к клеточным автоматам обусловлен тем, что они образуют парадигму параллельных вычислений подобно тому, как машина Тьюринга образуют парадигму последовательных. Таким образом, они могут использоваться, как модели, обладающие естественным параллелизмом.

Одно из главных отличий клеточной системы от всех прочих вычислительных систем состоит в том, что во всех других системах присутствуют две принципиально различные части: архитектурная, которая фиксирована и активна (то есть выполняет некоторые

операции) и данные, которые переменны и пассивны (то есть сами по себе они ничего сделать не могут). У клеточных автоматов и та, и другая части состоят из принципиально изоморфных, неотличимых друг от друга элементов. Таким образом, вычислительная система может оперировать своей материальной частью, модифицировать, расширять себя и строить себе подобных [2]. Хотя системы этого класса и были придуманы Джоном фон Нейманом, такая параллельная архитектура получила название «не-фон-неймановской», так как последовательную архитектуру он создал раньше.

Данное утверждение может показаться спорным. Казалось бы, к архитектурной части логичнее отнести, например, решётку и правила. Однако это, скорее – аппаратная часть. Например, рассмотрим автомат, реализующий игру «Жизнь». При данных решётке и правилах, меняя лишь состояние решётки, можно реализовать универсальные вычислительные системы, позволяющие производить любые вычисления, которые, по своим выразительным способностям эквивалентны произвольным машинам Тьюринга и клеточным автоматам [8, 13]. Теми же возможностями обладает, в частности, автомат, называемый компьютером Бэнкса [9]. Однако использовать их крайне неэффективно, но с теоретической точки зрения это – важный результат.

1.3. Аппаратные и программные реализации клеточных автоматов

Для решения задач с помощью клеточных автоматов требуется большой объём памяти для хранения значений из клеток решётки. Однако взаимодействие переменных, соответствующих клеткам локально. В то время как в большинстве программ, как правило, вводится не столь большое количество переменных, которые влияют друг на друга произвольным образом.

При проведении эксперимента на клеточном автомате, необходимо производит огромное количество итераций. В работе [9] приводятся следующие данные: для получения удовлетворительных результатов решения прикладной задачи зачастую требуется выполнить порядка 10^{15} обновлений клеток. По чрезвычайно оптимистичной оценке, обновление клетки, при моделировании работы клеточного автомата на персональном компьютере с архитектурой IBM PC i386, может потребовать несколько микросекунд. Тогда эксперимент займёт тысячелетия!

Один из возможных выходов состоит в том, чтобы использовать вычислительные машины со специальной архитектурой. Эти вычислительные машины должны обладать высокой степенью параллелизма, и позволять вычислять единообразные функции, используя значения из окрестных ячеек данных в качестве аргументов. Такая система позволит достигнуть производительности выше на большое число порядков.

Самая известная подобная «машина клеточных автоматов» была разработана в Массачусетском Технологическом Институте (Massachusetts Institute of Technology). Этот проект носит название CAM (Cellular Automation Machine) [9, 14].

Последняя, на данный момент, версия этого продукта CAM-8 [14] представляет собой устройство, подключаемое к компьютеру с архитектурой IBM PC i386 и работающее под его управлением. В частности машине необходима видеосистема персонального компьютера для визуализации происходящего при моделировании, а также электропитание, дисковая память и т.д. В обмен машина предоставляет свои вычислительные возможности. Симбиоз получился чрезвычайно выгодным. Устройства CAM нашли широкое применение во многих научно-исследовательских институтах всего мира в качества экспериментальной лаборатории благодаря чудесной производительности и весьма низкой цене.

Существуют и многочисленные программные имитаторы универсальных клеточных автоматов общего назначения. Весьма удачный проект, который впоследствии и перерос в CAM, был реализован Томазо Тоффоли (Tommaso Toffoli) и его соавторами [15]. Конечно, они очень существенно уступают аппаратным реализациям, однако вычислительные возможности персональных компьютеров постоянно растут и они несравнимо распространённее и доступнее, чем специализированные устройства.

1.4. Формализм клеточных автоматов

В настоящем параграфе будут даны формальные определения клеточного автомата, автомата-трансдюсера, построен математический аппарат, позволяющий удобно описывать эти автоматы, а также будет доказан ряд важных теорем.

1.4.1. Определение клеточного автомата

Итак, *клеточный автомат* A представляет собой четвёрку объектов $\{G, Z, N, f\}$, где

- G –дискретное метрическое множество, *решётка* автомата. Наличие метрики позволяет гарантировать, что все клетки окрестности будут на конечном расстоянии

от данной, так как ни для какой метрики никакие две точки дискретного метрического пространства не могут быть бесконечно удалены друг от друга. Кроме того, понятие метрики и координаты клетки оказывается чрезвычайно полезным при определении и нахождении окрестных клеток, а также анализе происходящего на решётке. Как правило, пространство G является одно-, двух- или трёхмерным, однако оно может быть и большей размерности;

- Z – конечное множество возможных состояний клеток. Можно написать, что *состояние решётки* является элементом множества $Z^{|G|}$;
- N – конечное множество, определяющее *окрестность* клетки, то есть, те $|N|$ клеток, которые влияют на новое состояние данной. Его можно задать, например, как множество относительных смещений (в текущей метрике) от данной клетки, позволяющее найти элементы окрестности;
- f – *правила* автомата. Они представляют собой вычислимую программу, которая, используя $|N|+1$ аргумент для автомата с клетками с памятью и $|N|$ аргументов для автомата с клетками без памяти, позволяет вычислять новое состояние данной клетки. Иногда правила могут быть записаны, как логическая или математическая *функция переходов*, действующая $Z \times Z^{|N|} \to Z$ для автомата с клетками с памятью и $Z^{|N|} \to Z$ для автомата с клетками без памяти.

Автомат, определённый таким образом, обладает всеми свойствами классических клеточных автоматов, перечисленными в разделе 1.1:

1. Локальность правил взаимодействия обеспечивается конечностью $|N|$ и дискретностью G;
2. Однородность обеспечивается единственностью определения окрестности N, множества значений Z и правил f;
3. Число возможных состояний каждой клетки конечно по определению множества Z;
4. Единовременность изменения состояний всех клеток можно обеспечить, если при вычислении их новых значений согласно правилам f использовать дополнительное хранилище для нового состояния решётки, в которое помещать вычисляемые значения. После того, как все значения получены, можно переместить значения из хранилища нового состояния решетки на место состояния с предыдущей итерации.

Как отмечалось в разделе 1.1, иногда, при решении задач, возникает необходимость отказаться от последних трёх свойств классической модели.

Рассмотрим задачу о моделировании некого вещества. Пусть в этом веществе есть отдельные области, характеризуемые различными физическими свойствами и описываемые различными уравнениями. Принадлежность клетки к той или иной области определяется её состоянием, описывающим физические характеристики участка материала, охватываемого клеткой. В результате клетка может переходить из одной области в другие. Однако различия между законами, соответствующими той или иной области могут быть столь существенными, что возникнет необходимость нарушить второе свойство об однородности структуры решётки. Тем не менее, это нарушение не выведет рассматриваемую систему за рамки понятия клеточный автомат. Перенумеруем области индексами от 1 до n. Пусть $Z = \bigcup_{i=1}^{n} Z_i$ (где Z_i – дизъюнктны) и по принадлежности состояния клетки тому или иному Z_i они и будут классифицироваться по областям. Пусть внутри каждой из них новые состояния клеток будут вычисляться по правилам f_i, учитывая клетки окрестности, определённые с помощью N_i. Тогда, можно записать $N = \bigcup_{i=1}^{n} N_i$, что не нарушит конечности множества N, а $f=f_i$, если состояние клетки, для которой вычисляется новое значение, принадлежит Z_i. Однако теперь при вычислении f могут быть использованы не все клетки окрестности N. Таким образом, нарушение второго свойства на практике, не ведёт к его нарушению в теории.

Необходимость выйти за ограничения, накладываемые третьим свойством, чаще всего возникает при использовании в качестве значений, размещаемых в клетках, чисел с плавающей точкой. Если хранить их не в переменных стандартных типов данных (например, `float` или `double` [12]), а в виде последовательность цифр с произвольной точностью. Это не нарушит работу модели, если правила f при этом останутся вычислимыми. Как правило, время получения нового состояния клетки останется конечным, либо если правила f только преобразуют значения из клеток, не классифицируя их по принадлежности некоторым множествам, либо если они учитывают принадлежность значений из клеток только конечному множеству множеств (то есть, если можно разбить множество возмож-

ных состояний клетки на конечный набор дизъюнктных множеств $Z = \bigcup_{i=1}^{n} Z_i$ и в правилах проверять значение состояния обрабатываемой клетки на принадлежность Z_i, а не на равенство каждому элементу бесконечного множества Z).

Возможность отказаться от четвёртого свойства позволяет существенно повысить производительность системы и вдвое уменьшить затраты памяти, так как при этом не нужно размещать новые вычисленные значения во временном хранилище состояния решётки и не нужно потом его копировать на место старого. Однако возможность сэкономить таким образом возникает не часто. Тем не менее, существуют процессы, которые лучшим образом описываются именно, когда порядок или направление перебора клеток решётки определено, и замена старых значений в клетках новыми значениями происходит незамедлительно. Автоматы, в которых значения состояний клеток меняются не одновременно, называются *асинхронными*.

Первое свойство автоматов нарушить невозможно. Как уже отмечалось выше, в дискретном метрическом пространстве никакие две точки не могут быть удалены друг от друга на бесконечное расстояние. Главное, чтобы количество клеток, от которых зависит данная, было конечным, чего не избежать по определению множества N.

В следующем параграфе вопросы о расстоянии между клетками и метрике будут рассмотрены подробнее.

1.4.2. Понятия, связанные с взаиморасположением клеток. Метрика для решёток клеточных автоматов

Рассмотрим клеточные автоматы с двумерными решётками из правильных многоугольников, которые встречаются на практике чаще всего. Возможны всего три таких решётки: треугольная (рис. 2), квадратная (рис. 3) и гексагональная (рис. 4). Ниже последнее утверждение будет доказано.

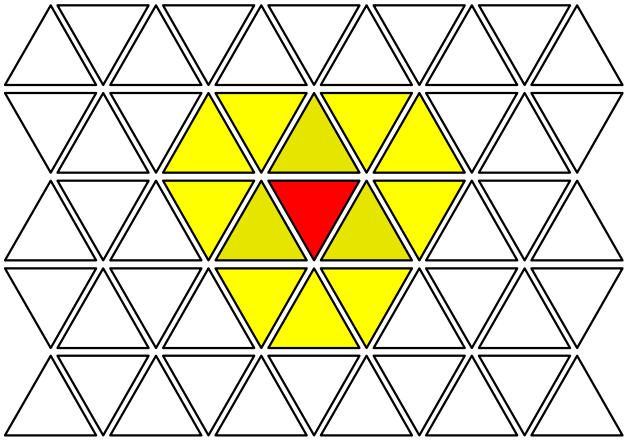

Рис 2. Треугольная решётка клеточного автомата

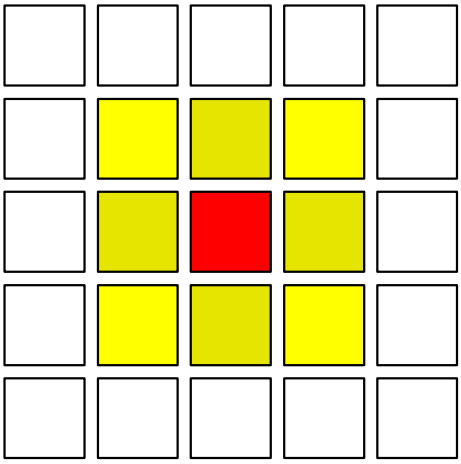

Рис 3. Квадратная решётка клеточного автомата

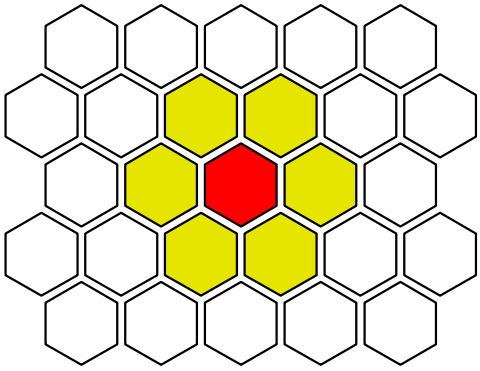

Рис 4. Гексагональная решётка клеточного автомата

Теорема 1-1:

Не существует другой решётки из правильных многоугольников, кроме треугольной, квадратной и гексагональной.

Доказательство:

Сумма углов правильного n-угольника $A(n) = p \cdot (n-2)$. Тогда $\frac{A(n)}{n}$ – величина каждого угла этого n-угольника. Пусть из правильных n-угольников удалось составить решётку. Тогда в ней угол в $2p$ радиан составляют углы целого числа (обозначим его k) фигур. То есть, k многоугольников можно составить так, чтобы они имели общую вершину. Найдём это k, как функцию от n. Это можно сделать из следующего уравнения:

$$2p = k \cdot \frac{A(n)}{n} \Leftrightarrow 2p = k \cdot \frac{p \cdot (n-2)}{n} \Rightarrow k(n) = \frac{2n}{n-2}$$

Будем рассматривать эту функцию только при $n \geq 3$, так как треугольник – многоугольник с наименьшим количеством вершин. Возьмем производную от $k(n)$ по n:

$$k'(n) = -\frac{n}{(n-2)^2}$$

Очевидно, что при $n \geq 3$ функция $k(n)$ убывает, так как $k'(n) < 0$. Таким образом, все возможные значения k меньше $k(3)$, то есть шести. К тому же, $k(n) > 2$, так как

$$k(n) > 2 \Leftrightarrow \frac{2n}{n-2} > 2 \Leftrightarrow 2n > 2n - 4 \Leftrightarrow 0 > -4$$ - истинно

Решётку можно построить, только если целому n будет соответствовать целое k. Из изложенного выше следует, что возможны лишь четыре значения k: 6, 5, 4 и 3. Построим функцию $n(k)$, обратную к $k(n)$, и проверим, каким из возможных значений k соответствует целое n:

$$k = \frac{2n}{n-2} \Leftrightarrow k \cdot (n-2) = 2n \Leftrightarrow n \cdot (k-2) = 2k \Rightarrow n(k) = \frac{2k}{k-2}$$

Имеем:

- $k = 6 \Rightarrow n(k) = 3$ – треугольная решётка;
- $k = 5 \Rightarrow n(k) = \frac{10}{3}$ – не целое n, то есть решётку не построить;

- $k = 4 \Rightarrow n(k) = 4$ – квадратная решётка;
- $k = 3 \Rightarrow n(k) = 6$ – гексагональная решётка;

Таким образом, действительно возможны только три перечисленные выше решётки из правильных многоугольников.

Q.e.d.

Обозначим через a некоторый элемент множества G, клетку решётки. Множество клеток, составляющее окрестность клетки a, будем обозначать, как $N(a)$. Как уже отмечалось выше окрестность в общем виде можно задавать, как набор из $|N|$ смещений относительно данной клетки. Отдельно следует отметить, что для треугольной решётки может возникнуть необходимость в том, чтобы задать N, как набор из двух множеств, первое – для клеток ориентированных «вверх» (▲), а второе для ориентированных «вниз» (▼).

Наиболее часто, в качестве окрестностей клеток используют их непосредственных соседей, так называемую, *окрестность Мура*. На рис. 2–4 тёмным цветом выделена сама клетка, а светлым – её соседи. Те из них, который имеют с клеткой общую сторону, называются *главными соседями*. На рисунках они выделены штриховкой. Окрестность, составленная из главных соседей клетки, называется *окрестностью фон Неймана*.

Каждая клетка треугольной решётки окружена двенадцатью соседями, из которых три – главные. На квадратной решётке любая клетка имеет по восемь соседей, четыре из которых – главные. У клетки гексагональной решётки все шесть соседей – главные. Множество непосредственных соседей клетки a будем обозначать, как $S(a)$, А множество главных соседей – как $S_0(a)$. Ясно, что $S_0(a)$ содержится в $S(a)$.

Введём понятие *кольца* для клетки решётки. Сама клетка является собственной соседкой нулевого кольца, а её непосредственные соседи – соседями первого кольца. Далее – рекурсивно. Для данной клетки, соседями i-го кольца называются все клетки являющиеся непосредственными соседями какой-либо клетки $(i–1)$-го кольца, исключая те клетки, которые сами принадлежат $(i–1)$-му или $(i–2)$-му кольцам.

На рис 5-7 изображены кольца с нулевого по четвёртое для тёмной клетки на трёх обсуждавшихся решёток из правильных многоугольников. Кольца выделены различными цветами или штриховками.

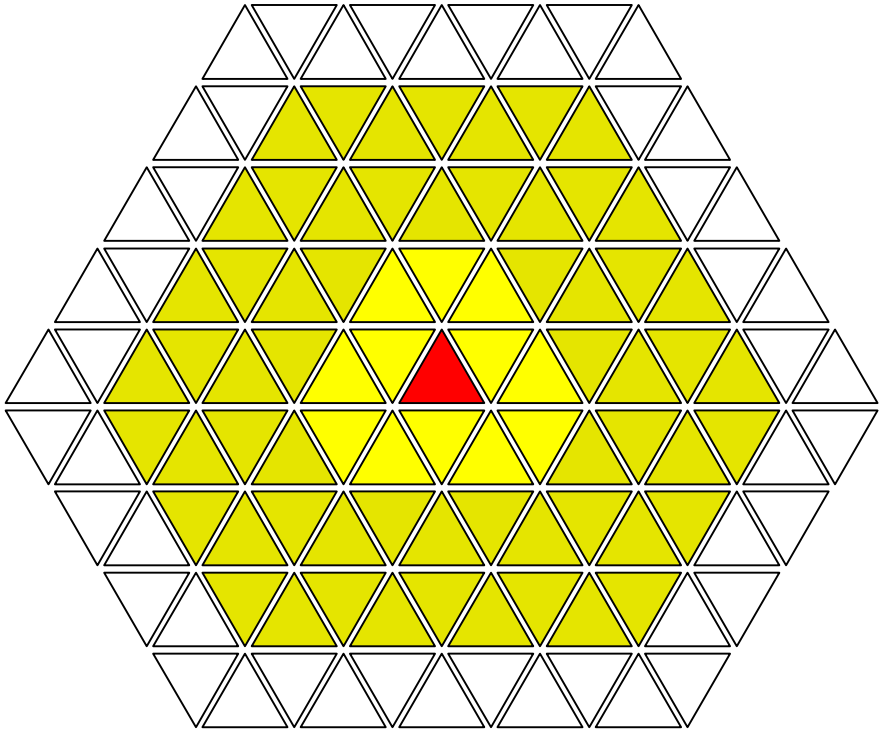

Рис 5. Кольца для треугольной решётки

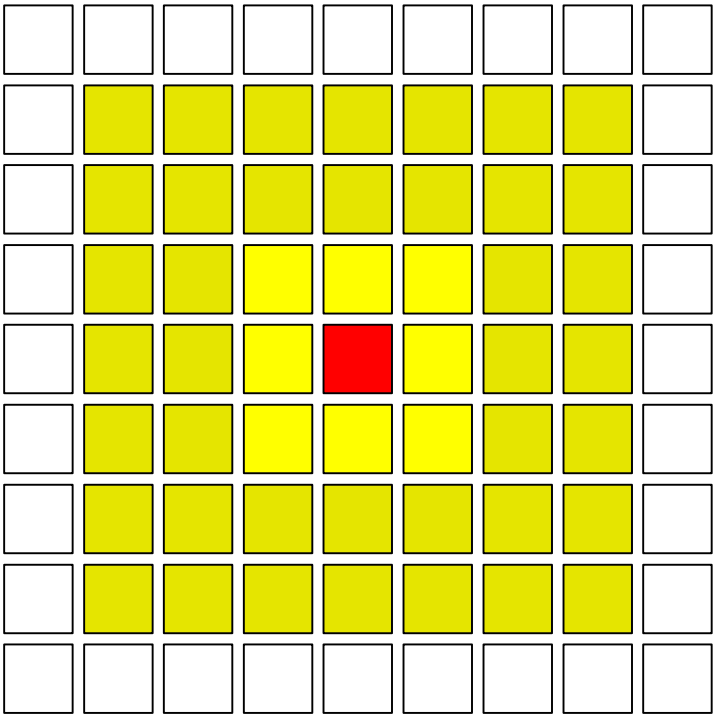

Рис 6. Кольца для квадратной решётки

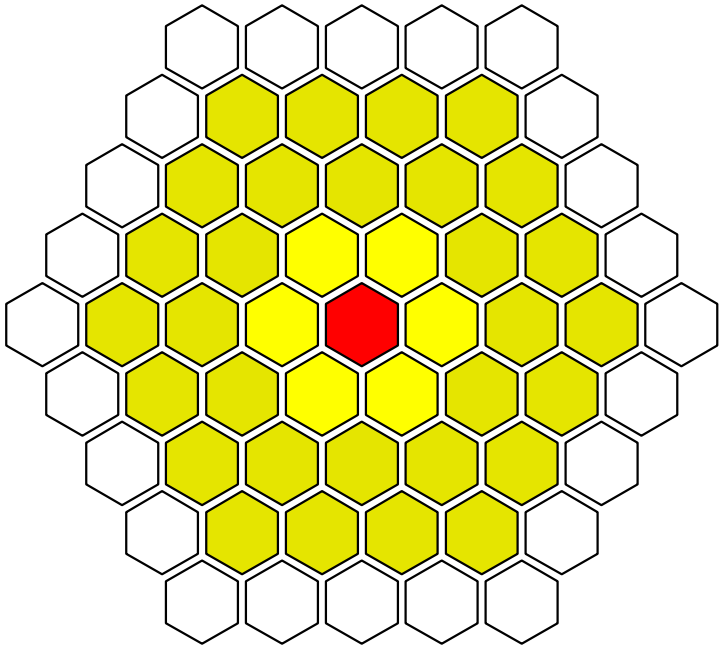

Рис 7. Кольца для гексагональной решётки

Фразы «соседи *i*-го кольца некой клетки», «клетки *i*-го кольца некой клетки» и «*i*-е кольцо некой клетки» будем считать синонимичными.

Обозначим через $R(a, i)$ множество клеток *i*-го кольца клетки a ($i \geq 0$). Тогда формально определение понятия кольца можно записать так:

$$R(a,0) = \{a\}, R(a,1) = N(a)$$
$$R(a,i) = \{b \mid \exists c : b \in R(c,1), c \in R(a,i-1), b \notin R(a,i-1), b \notin R(a,i-2)\} \quad (1)$$

После этого определения можно доказать теоремы и лемму, декларирующие важные свойства понятия кольца.

Теорема 1-2. Основное свойство понятия кольца

Для любых двух клеток a и b, существует и единственно неотрицательное число i такое, что $b \hat{I} R(a, i)$.

Доказательство:

Существование. Пусть не существует такого *i*, чтобы $b \in R(a, i)$. Тогда не существует таких чисел i_a и i_b, что для них найдётся две клетки c_a и c_b, такие, что $a \in R(c_a, i_a)$,

$b \in R(c_b, i_b)$, а $c_a \in S(c_b)$ или $c_b \in S(c_a)$. Таким образом, решётка разделится, как минимум, на два подмножества клеток так, что ни одна из клеток одного подмножества не будет соседкой никакой из клеток другого подмножества. А это противоречит определению решётки G.

Единственность. Пусть i – не единственно, а существует ещё и k, для которого выполнено $b \in R(a, k)$. Это означает, что:

$\exists c_i$: $b \in R(c_i, 1)$, $c_i \in R(a, i)$, $b \notin R(a, i-1)$, $b \notin R(a, i-2)$ (a)

$\exists c_k$: $b \in R(c_k, 1)$, $c_k \in R(a, k)$, $b \notin R(a, k-1)$, $b \notin R(a, k-2)$ (b)

Так как b одновременно является соседом и c_i, и c_k, то $|k - i| \le 2$, так как $c_k \in S(S(c_i))$.

Рассмотрим три случая: $k=i+2$, $k=i+1$ и $k=i$. Пусть $k=i+2$, тогда выражение (b) примет вид:

$\exists c_k$: $b \in R(c_k, 1)$, $c_k \in R(a, i+2)$, $b \notin R(a, i+1)$, $b \notin R(a, i)$

Однако то, что $b \notin R(a, i)$ противоречит предположению теоремы. Пусть теперь $k=i+1$, тогда выражение (b) примет вид:

$\exists c_k$: $b \in R(c_k, 1)$, $c_k \in R(a, i+1)$, $b \notin R(a, i)$, $b \notin R(a, i-1)$

Однако то, что $b \notin R(a, i)$ противоречит предположению теоремы, а то, что $b \notin R(a, i-1)$ противоречит выражению (a). Таким образом, $k=i$, а, следовательно, значение i единственно.

Q.e.d.

Лемма 1-1. О симметрии отношения соседства

Для любых двух клеток a и b, утверждения $b \hat{I} S(a)$ и $a \hat{I} S(b)$ эквивалентны.

Доказательство:

Пусть $b \in S(a)$ или, что то же, $b \in R(a, 1)$. Отсюда по теореме 1-2 следует, что существует значение i такое, что $a \in R(b, i)$. Причём i не может быть равно нулю, так как $b \in S(a)$, а, следовательно, $a \neq b$. К тому же i не может быть больше единицы, так как у a и b по условию леммы есть общие вершины или стороны (так как $b \in S(a)$), а это означает, что $i=1$, а по теореме 1-2 значение i единственно. Таким образом, $a \in S(b)$. Аналогично проводятся рассуждения в другую сторону.

Q.e.d.

Теорема 1-3. О симметрии отношения принадлежности и кольцу

Для любых двух клеток a *и* b*, утверждения* $b \in R(a, i)$ и $a \in R(b, i)$ *эквивалентны.*

Доказательство:

Докажем утверждение методом математической индукции по i:

База. Пусть $i=0$. Тогда, если $b \in R(a, 0)$, то a и b совпадают и следовательно $a \in R(b, 0)$. Аналогично проводятся рассуждения в другую сторону. Случай $i=1$ доказан леммой 1-1.

Переход. Пусть для $i=j-1$ и $i=j$ утверждение верно. И, в частности, $a \in R(b_{j-1}, j-1) \Leftrightarrow b_{j-1} \in R(a, j-1)$ и $a \in R(b_j, j) \Leftrightarrow b_j \in R(a, j)$. Докажем для $i=j+1$, что $a \in R(b_{j+1}, j+1) \Leftrightarrow b_{j+1} \in R(a, j+1)$.

$\Rightarrow$ Докажем, что из утверждения $a \in R(b_{j+1}, j+1)$ следует, что $b_{j+1} \in R(a, j+1)$. Действительно, по формуле (1), чтобы $b_{j+1} \in R(a, j+1)$ необходимо существование клетки c такой, чтобы:

1. $b_{j+1} \in R(c, 1)$;
2. $c \in R(a, j)$;
3. $b_{j+1} \notin R(a, j)$;
4. $b_{j+1} \notin R(a, j-1)$.

В качестве клетки c выберем клетку b_j, для которой выполнено первое из условий. Это можно сделать по определению: у клетки из $(j+1)$-го кольца должна быть соседка с j-го и, следовательно, она была рассмотрена на предыдущем шаге индукции. Второе условие верно по индукционному предположению. А третье и четвёртое условия верны, так как, если бы клетка b_{j+1} принадлежала кольцу с меньшим индексом, то она бы уже была рассмотрена на предыдущих шагах индукции, и на этом возникнуть уже не могла в силу единственности индекса кольца по теореме 1-2. Таким образом, действительно, если $a \in R(b_{j+1}, j+1) \Rightarrow b_{j+1} \in R(a, j+1)$.

$\Leftarrow$ Докажем, что из утверждения $b_{j+1} \in R(a, j+1)$ следует, что $a \in R(b_{j+1}, j+1)$. Действительно, по формуле (1), чтобы $a \in R(b_{j+1}, j+1)$ необходимо существование клетки c такой, чтобы:

1. $a \in R(c, 1)$;

2. $c \in R(b_{j+1}, j) \Leftrightarrow b_{j+1} \in R(c, j)$ – по индукционному предположению ;
3. $a \notin R(b_{j+1}, j)$;
4. $a \notin R(b_{j+1}, j-1)$.

В качестве клетки c возьмём соседку клетки a (что гарантирует выполнение первого условия), для которой выполнено второе условие. Это можно сделать так как, если $c \in R(b_{j+1}, j)$, то $c \notin R(b_{j+1}, j+1)$ (по теореме 1-2), а по определению у клетки из $(j+1)$-го кольца должна быть соседка с j-го. Эту соседку и следует выбрать. Третье и четвёртое условия верны, так как, если бы клетка a принадлежала кольцу с меньшим индексом, то она бы уже была рассмотрена на предыдущих шагах индукции, и на этом возникнуть уже не могла в силу единственности индекса кольца по теореме 1-2. Таким образом, действительно $b_{j+1} \in R(a, j+1) \Rightarrow a \in R(b_{j+1}, j+1)$.

Q.e.d.

Зададим метрику для пространства G. Это можно сделать многими способами, например, так: расстоянием $D(a, b)$ от клетки a до клетки b будем называть номер кольца клетки a (или b), которому принадлежит клетка b (или a). Это можно записать так:

$$\begin{aligned} D(a,b) &= i : a \in R(b,i) \\ D(a,b) &= i : b \in R(a,i) \end{aligned} \qquad (2)$$

Теорема 1-4:

Определённая формулой (2) функция $D(a, b)$ – метрика.

Доказательство:

Проверим три основные свойства метрики [11].

1. $D(a, b)$ неотрицательно определена, так как по определению индекс кольца неотрицателен. $D(a, b)=0 \Leftrightarrow a=b$ – верно, так как $R(a, 0)=\{a\}$.
2. $D(a, b)=D(b, a)$ – верно, так как $b \in R(a, i) \Leftrightarrow a \in R(b, i)$ по теореме 1-3.
3. Докажем, что $D(a, b) \le D(a, c)+D(c, b)$ методом математической индукции по D(c, b):

База. Если D(c, b)=0, то клетки b и c совпадают и $D(a, b)=D(a, c)+0$.

Переход. Пусть для всех клеток b и c таких, что $D(c, b)=i$, и любых a, верно утверждение $D(a, b)\le D(a, c)+ D(c, b)$. Докажем, что для всех клеток b и c таких, что $D(d, b)=i+1$, и любых a, верно утверждение $D(a, b)\le D(a, d)+D(d, b)$, где $d\in S(c)$.

Так как $d\in S(c)$, то $D(a, d)$ может принимать лишь три значения: $D(a, c)-1$, $D(a, c)$ и $D(a, c)+1$. Подставим поочерёдно все три значения в неравенство $D(a, b)\le D(a, d)+D(d, b)$, учитывая, что $D(d, b)=D(c, b)+1$. В результате, получим $D(a, b)\le D(a, c)+D(c, b)+e$, где $0\le e\le 2$, что верно по индукционному предположению.

Q.e.d.

Понятие кольца и расстояния могут быть обобщены и распространены на случай решёток любой размерности.

Необходимо отметить, что вопрос о метрике для решётки клеточного автомата не поднимался другими авторами [16]. Тем не менее, она всегда неявно присутствовала. Если при написании правил без рассмотрения координат клеток можно обойтись, используя относительные смещения из клетки, состояние которой необходимо вычислить, то при рассмотрении реализации, для того, чтобы правильно ставить в соответствие клетке переменную в памяти вычислительной системы, понятие координаты клетки практически необходимо. А координаты и метрика тесно взаимосвязаны, так как они обуславливают друг друга. Для исследователя же координаты предоставляют удобный механизм идентификации клеток, оценки их взаиморасположения и, как следствие, расстояния между ними.

Для одной и той же метрики, в зависимости от задачи координаты можно ввести различными способами. Простейший из них – декартовы координаты, при использовании которых каждая клетка характеризуется двумя целыми числами x и y. Пусть клетка a имеет координаты $(x_a; y_a)$, а клетка b – $(x_b; y_b)$. Тогда описанную выше метрику $D(a, b)$ можно ввести по следующей формуле:

$$D(a,b) = \max(\mathrm{abs}(x_b - x_a); \mathrm{abs}(y_b - y_a)) \qquad (3)$$

Другой способ введения координат для описанной выше метрики – обобщённые координаты [17, 18]. Каждая клетки характеризуется при этом неким уникальным целым числом. Организовать это можно, например, так: обозначим некоторую клетку номером

ноль, далее пронумеруем по часовой стрелке все клетки её первого кольца, потом продолжим во втором кольце, затем – в третьем и так далее.

Для таких координат построен механизм нахождения множества всех непосредственных соседей произвольной клетки [17, 18]. В случае более сложной окрестности, её клетки могут быть получены, как соседи соседей или соседи соседей соседей и так далее.

Обобщённые координаты имеют большое значение, так как позволяют хранить информацию из многомерных решёток в одномерной линейной структуре.

1.4.3. Определение клеточного автомата-трансдюсера. Теорема об эквивалентности набору конечных автоматов

Выше было дано определение автономного клеточного автомата. Однако, как отмечалось ранее, автоматы этого класса могут выступать и в роли трансдюсеров, то есть получать некие входные воздействия и преобразовывать их в выходные. Определим клеточный автомат-трансдюсер, но сначала введём определения конечных автоматов-трансдюсеров Мили и Мура [19].

Конечный автомат Мили есть пятёрка объектов $A=\{Z, X, Y, f, g\}$, где

- Z – конечное множество состояний автомата;
- X – конечное множество входных воздействий на автомат;
- Y – конечное множество выходных воздействий автомата;
- f – функция переходов автомата, $Z\times X \to Z$;
- g – сюръективная функция выходов автомата, $Z\times X \to Y$.

Конечный автомат Мура есть пятёрка объектов $A=\{Z, X, Y, f, g\}$, отличающаяся от автомата Мили лишь тем, что функция выходов g – есть сюръективное отображение $X \to Y$. Понятие «функция» в двух последних определениях может быть заменено понятием «вычислимая программа».

Клеточный автомат-трансдюсер A представляет собой семёрку объектов $\{G, Z, X, Y, N, f, g\}$, где

- X – конечное множество входных воздействий на клетку;
- Y – конечное множество выходных воздействий клетки;
- f – *правила* автомата. Они представляют собой вычислимую программу, которая, используя $|N|+1$ пару аргументов для автомата с клетками с памятью и $|N|$ пар аргу-

ментов для автомата с клетками без памяти, позволяет вычислять новое состояние данной клетки. Каждая пара аргументов представляет собой состояние клетки и входное воздействие на неё. Иногда правила могут быть записаны, как логическая или математическая *функция переходов*, действующая $(Z\times X)\times(Z\times X)^{|N|}\to Z$ для автомата с клетками с памятью и $(Z\times X)^{|N|}\to Z$ для автомата с клетками без памяти. Можно в правилах не рассматривать значения входных воздействий на клетки, как в конечных автоматах Мура;

- g – *правила выходов* автомата. Они представляют собой вычислимую программу, которая, используя $|N|+1$ пару аргументов для автомата с клетками с памятью и $|N|$ пар аргументов для автомата с клетками без памяти, позволяет вычислять новое состояние данной клетки. Каждая пара аргументов представляет собой состояние клетки и входное воздействие на неё. Иногда правила могут быть записаны, как логическая или математическая *функция выходов*, действующая $(Z\times X)\times(Z\times X)^{|N|}\to Y$ для автомата с клетками с памятью и $(Z\times X)^{|N|}\to Y$ для автомата с клетками без памяти. Можно в правилах выходов не рассматривать значения входных воздействий на клетки, как в конечных автоматах Мура.

Остальные объекты сохраняют свои значения из определения обычного клеточного автомата.

Потребность в использовании автоматов-трансдюсеров возникает при решении задач управления или преобразования потоков информации:

Докажем, что клеточный автомат эквивалентен не более чем счётному набору конечных автоматов. Не умаляя общности можно рассматривать только клеточные автоматы-трансдюсеры (автономные автоматы являются их частным случаем, в котором множества входных и выходных воздействий пусты). Необходимо отметить, что даже при построении системы, эквивалентной автономному клеточному автомату, потребуются конечные автоматы-трансдюсеры, так как они должны будут взаимодействовать при помощи входных и выходных воздействий.

Теорема 1-5. Теорема об эквивалентности набору конечных автоматов

Клеточный автомат $A=\{L, Z, X, Y, N, f, g\}$ по функциональности эквивалентен набору идентичных конечных автоматов Мили $\{A_i\}$, где $A_i=\{Z_i, X_i, Y_i, f_i, g_i\}$, если $X_i=X\times Z^{|N|}$, а $i=1\div|L|$.

Доказательство:

Построим формальные конструктивные методы перехода от клеточного автомата-трансдюсера к множеству конечных автоматов-трансдюсеров и обратно. Тем самым, докажем их эквивалентность.

$\Rightarrow$ Рассмотрим преобразование клеточного автомата $A=\{G, Z, X, Y, N, f, g\}$ в набор конечных автоматов Мили $\{A_i\}$, где $A_i=\{Z_i, X_i, Y_i, f_i, g_i\}$, $1\le i\le|G|$. Каждый конечный автомат будет реализовывать функции одной клетки клеточного автомата. Для каждого i опишем пять составляющих компонентов автомата Мили:

- $Z_i=Z$;
- $X_i=(X\times Z)^{|N|}\cup\{\perp\}$ для клеточного автомата с клетками без памяти или $X_i=(X\times Z)^{|N|+1}\cup\{\perp\}$ для клеточного автомата с клетками с памятью. Каждый элемент множества $X\times Z$ служит для передачи автомату информации о состоянии и входном воздействии на соседа. Кроме того, к множеству входных воздействий необходимо добавить элемент «$\perp$», сигнал обновления;
- $Y_i=Y$;
- f_i должна реализовывать следующую функциональность. Если входное воздействие на автомат не равно сигналу обновления, то f_i должна вызвать для соответствующей клетки функцию $f(t)$, где $t\in X_i$, и сохранить полученное значение во временном хранилище №1. Вернуть, при этом, функция f_i должна текущее состояние автомата. Если входное воздействие на автомат равно сигналу обновления, то функция f_i должна вернуть значение из временного хранилища №1;
- g_i должна реализовывать следующую функциональность. Если входное воздействие на автомат не равно сигналу обновления, то g_i должна вызвать для соответствующей клетки функцию $g(t)$, где $t\in X_i$, и сохранить полученное значение во временном хранилище №2. Вернуть, при этом, функция g_i должна то же значение, которое она возвращала в предыдущий раз (для организации этого понадобится временное хранилище №3). Если входное воздействие на автомат равно сигналу обновления, то функция g_i должна вернуть значение из

сигналу обновления, то функция g_i должна вернуть значение из временного хранилища №2.

Итерация модели должна будет состоять из двух фаз. Первая заключается в передаче всем автоматам входных воздействий. По завершении этого, на второй фазе, автоматы должны получить сигналы обновления.

⇐ Рассмотрим преобразование набора конечных автоматов Мили $\{A_i\}$, где $A_i=\{Z_i, X_i, Y_i, f_i, g_i\}$, $1 \le i \le |G|$ в клеточный автомат $A=\{G, Z, X, Y, N, f, g\}$. Каждый конечный автомат будет реализовываться одной клеткой автомата A. Опишем семь составляющих компонентов клеточного автомата:

- решётка G выбирается произвольно с тем лишь условием, чтобы мощность этого множества была не меньше количества элементов в наборе $\{A_i\}$. Не умоляя общности, будем считать, что в наборе $\{A_i\}$ ровно $|G|$ элементов;
- $Z = \bigcup_{i=1}^{|G|} Z_i$;
- $X = \bigcup_{i=1}^{|G|} X_i \cup \{\perp\}$;
- $Y = \bigcup_{i=1}^{|G|} Y_i \cup \{\perp\}$;
- f должна реализовывать следующую функциональность. Если входное воздействие на клетку не равно сигналу обновления, то f должна вызвать соответствующую данной клетке функцию $f_i(t)$, где $t \in X$, и сохранить полученное значение во временном хранилище №1. Вернуть, при этом, функция f должна текущее состояние клетки. Если входное воздействие на клетку равно сигналу обновления, то функция f должна вернуть значение из временного хранилища №1;
- g должна реализовывать следующую функциональность. Если входное воздействие на клетку не равно сигналу обновления, то g должна вызвать для соответствующей клетки функцию $g_i(t)$, где $t \in X$, и сохранить полученное значение во временном хранилище №2. Вернуть, при этом, функция g должна то же значение, которое она возвращала в предыдущий раз (для организации этого понадобится временное хранилище №3). Если входное воздействие на клетку равно

сигналу обновления, то функция g должна вернуть значение из временного хранилища №2.

Упомянутые выше три временных хранилища должны существовать для каждой клетки автомата.

Итерация модели должна будет состоять из двух фаз. Первая заключается в передаче всем клеткам входных воздействий. По завершении этого, на второй фазе, клетки должны получить сигналы обновления.

Таким образом, клеточный автомат эквивалентен не более чем счётному набору конечных автоматов Мили.

Q.e.d.

Необходимо отметить, что для асинхронных автоматов потребность в сигналах обновления отпадает и тогда последняя теорема будет верна для набора автоматов не только Мили, но и Мура.

Метод конструирования клеточного автомата на основе множества конечных автоматов и обратный ему, изложенные в последней теореме, в силу своей общности, позволяют создать весьма неэффективные автоматы. Однако эта теорема доказывает важный факт: так как конечный автомат является моделью последовательного вычисления, то становится ясно, почему клеточные автоматы образуют парадигму параллельных.

Тем не менее, рассматривать клеточный автомат, как множество одинаковых конечных автоматов – не верно. Так как это нарушает представление об «источнике действия» в модели. Действия совершаются правилами, причём централизовано, для всех клеток по очереди, а изменения состояний клеток происходят единовременно.

Конечный автомат – замкнутая система. Использование этого понятия для представления клетки решётки может создать иллюзию того, что клетки самостоятельны и конкурируют друг с другом. Синхронизацию в эту систему приходится вводить искусственно с помощью, например, сигналов обновления, как это было сделано для доказательства теоремы 1-5.

ГЛАВА 2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ РЕШЕНИЯ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ КЛЕТОЧНЫХ АВТОМАТОВ. ПРОЕКТ CAME&L

В настоящей главе перечисляются задачи, которые ставятся перед программным обеспечением, служащим для решения задач с использованием клеточных автоматов. Помимо этого приводится перечень и описание существующих продуктов, разработанных для этих целей, а также описываются причины появления и основные возможности среды CAME&L.

2.1. Требования и задачи

Перечислим требования и задачи, которые возникают перед программным обеспечением служащим обсуждаемым целям. Приводимые ниже характеристики, конечно, не являются обязательными, однако, такая функциональность позволит проекту претендовать на то, чтобы быть универсальной средой для клеточных вычислений. Итак, программное обеспечение, по возможности, должно:

1. Предоставлять пользователям дружественный интерфейс взаимодействия с программой;
2. Позволять удобно и наглядно наблюдать эволюцию состояний решётки;
3. Позволять создавать и настраивать клеточные автоматы для решения конкретной задачи. Как минимум, программа должна иметь возможность использовать ту или иную окрестность, задавать правила и начальное состояние решётки. Дополнительное удобство для пользователя предоставила бы возможность автоматизировать задание начального состояния решётки или позволить его задание с посредством анализа определённых файлов;
4. Выполнять итерации максимально быстро, эффективно распараллеливая итерации автомата между доступными вычислительными ресурсами, а также позволяя создавать вычислительные кластеры;
5. Позволять удобно производить продолжительные вычислительные эксперименты, анализировать текущее состояние решётки и результаты вычислений;

6. Сохранять и восстанавливать из файла состояние решётки. Такая возможность может привести к возникновению де-факто стандарта сохранения клеточной информации.

Первое, второе, третье и шестое требования кажутся интуитивно ясными. Так как отражают проекцию общих представлений об удобстве программы на обсуждаемую задачу. Четвёртое и пятое требования требуют пояснения.

Как отмечалось выше, клеточный автомат – параллельная архитектура (см. раздел 1.3), требующая специфической аппаратной платформы (такой как, например, CAM [9, 14]). Те программные продукты, которые работают на неприспособленной архитектуре, как, например, IBM PC i386, требуют использовать какие-либо методы для оптимизации производительности. Вычисления должны эффективно разделяться между доступными процессорами или сетевыми вычислительными ресурсами. Поэтому организация вычислительного кластера может быть вменена в обязанность программе, моделирующей клеточный автомат. В этом и состоит содержание четвёртого требования.

Пятая задача состоит в том, что программа должна позволять пользователю проводить и наблюдать продолжительные эксперименты. При том, что почти все вычислительные ресурсы системы будут заняты производством итераций, программа должна продолжать быть интерактивной и отвечать на запросы пользователя. Кроме того, полезной была бы возможность анализа состояния решётки и результатов вычислений. Имеется в виду построение разнообразных графиков, демонстрация изменения различных величин и параметров, меняющихся от итерации к итерации.

В следующем параграфе будут рассмотрены существующие на настоящий момент программный продукты для решения обсуждаемых задач и перечислены причины возникновения проекта CAME&L.

2.2. Существующее программное обеспечение. Причины возникновения проекта CAME&L

Перечислим и кратко охарактеризуем наиболее известные и удачные продукты, служащие для решения задач с помощью клеточных автоматов.

- *CAM Simulator*. Авторы: З. Белзо (Z. Belso) и М. Варджиас (M. Vargyas);
  Адрес: ftp://ftp.lifesci.ucla.edu/pub/alife/public/cam.zip;

Распространяется свободно;

Требования к среде выполнения: рабочая станция с операционной системой MS-DOS;

Программа представляет собой симулятор устройства CAM-6 [9, 14]. От CAM-6 программа унаследовала ряд ограничений (использование в качестве элементов окрестности клеток, расположенных не далее, чем на единичном расстоянии от данной и т.п.).

- *CAMEL*. Авторы: Д. Талия (D. Talia) и Дж. Спеззано (G. Spezzano);

  Адрес: http://isi-cnr.deis.unical.it:1080/~talia/;

  Свободно не распространяется;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Unix и вычислительный кластер;

  По иронии судьбы, этот проект является почти тёзкой проекта описываемого в настоящей работе. Однако в данном случае CAMEL означает Cellular Automata environMent for systEms modeLing. Для описания автомата, определения решётки, задания правил, окрестности и прочего используется специально разработанный авторами язык CARPET (CellulAR Programming EnvironmenT). Программа с этого языка транслируется на язык C и выполняется средой CAMEL. При этом задачи разделяются между доступными вычислительными ресурсами. Программа поддерживает работу в кластере. Ограничений на задаваемые автоматы практически нет. Только требование, чтобы решётки были квадратные и не более, чем трехмерные.

- *CANL*. Авторы: P. Наполитано (R. Napolitano), К. ди Наполи (C. Di Napoli);

  Свободно не распространяется;

  Требования к среде выполнения: Cray YMP или Sun workstation;

  Программа поддерживает автоматы с двумерными квадратными решётками и декартовыми метриками. В окрестности могут присутствовать только непосредственные соседи.

- *CAPow*. Автор: P. Ракер (R. Rucker);

  Адрес: http://www.mathcs.sjsu.edu/capow/capow4a.zip;

  Распространяется свободно;

Требования к среде выполнения: рабочая станция с операционной системой семейства Windows или Windows NT;

Весьма удачный проект с богатым инструментарием для визуализации результатов. Программа поддерживает автоматы с одно- и двумерными решётками и декартовыми метриками. Правила для автомата описываются на языке C++ с использованием предназначенной для этого библиотеки. Собранные в динамическую библиотеку правила выполняются средой.

- *CASim.* Автор: С. Кёлер (S. Kohler);

  Адрес: http://www.uni-jena.de/~msk/bin/casim.tar.gz;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Unix с X11/Motif;

  Программа поддерживает автоматы с двумерными решётками и декартовыми метриками.

- *CAT/CARP.* Авторы: И. Ясеньяк (I. Jaceniak), С. Фоке (S. Focke), У. Линк (U. Link) и Г. Юнгер (G. Junger);

  Адрес: ftp://ftp.gmd.de:/GMD/cat/;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Windows, Windows NT или OS/2;

  Программа поддерживает автоматы с двумерными квадратными решётками и декартовыми метриками.

- *CDL.* Автор: К. Хошбергер(C. Hochberger);

  Адрес: ftp://ftp.informatik.th-darmstadt.de/pub/MP/CDL/;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Unix и вычислительный кластер машин с операционной системой Unix или FPGA семейства CEPRA;

  Программа поддерживает автоматы с двумерными квадратными решётками и декартовыми метриками. Правила пишутся на языке C и компилируются.

- *CDM/SLANG.* Автор: Х. Зибург (H. Sieburg);

Адрес: ftp://ftp.essex.ac.uk/pub/robots/ArtificialLife/public/cdm/;

Распространяется свободно;

Требования к среде выполнения: рабочая станция Apple Macintosh;

Программа поддерживает автоматы с двумерными решётками и декартовыми метриками.

- *CelLab.* Авторы: Р. Ракер (R. Rucker) и Дж. Уолкер (J. Walker);

  Адрес: http://www.fourmilab.ch/cellab/cellab.zip;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Windows;

  Один из авторов этого проекта разработал весьма удачную среду CAPow. Правила пишутся на языке С и компилируются.

- *CELLAS/FUNDEF*. Автор: Т. Легенди (T. Legendi);

  Свободно не распространяется;

  Требования к среде выполнения: рабочая станция с операционной системой семейства MS-DOS;

  Программа поддерживает автоматы с двумерными решётками и декартовыми метриками.

- *Cellsim.* Авторы: К. Ленгтон (C. Langton) и Д. Хибелер (D. Hiebeler);

  Адрес: ftp://alife.santafe.edu/pub/SOFTWARE/Cellsim/;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Unix или станция Connection Machine CM-2;

  Программа поддерживает автоматы с одно- и двумерными квадратными решётками и декартовыми метриками. Клетки окрестности могут находиться на расстоянии не более трёх от данной. Программа разрабатывалась в университете города Сантафе, одном из лидеров в области исследования клеточных автоматов.

- *Cellular/Cellang.* Автор: Дж. Дана Эккарт (J. Dana Eckart);

  Адрес: ftp://rucs2.sunlab.cs.runet.edu/pub/ca/cellular.tar.gz;

  Распространяется свободно;

Требования к среде выполнения: рабочая станция семейства Sun workstation или SGI с операционной системой семейства Unix или станция с операционной системой семейства MS-DOS;

Программа поддерживает автоматы с произвольными квадратными, кубическими, гипер-кубическими решётками и декартовыми метриками.

- *CEPROL.* Автор: F. Зойтер (F. Seutter);

  Требования к среде выполнения: рабочая станция с операционной системой семейства Unix;

  Программа поддерживает автоматы с двумерными решётками и декартовыми метриками. Правила пишутся на языке Pascal и компилируются.

- *DDLab.* Автор: К. Ленгтон (A. Wuensche);

  Адрес: ftp://ftp.santafe.edu/pub/wuensch/;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция семейство Sun workstation, Apple Macintosh или станция с операционной системой семейства MS-DOS;

  Весьма богатый по своим возможностям проект, который разрабатывался в университете города Сантафе.

- *HICAL.* Авторы: А. Бекерс (A. Beckers), Т. Уорш (T. Worsch) и др.;

  Адрес: http://liinwww.ira.uka.de/~worsch/;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Unix с X11;

  Программное обеспечение с очень широкими возможностями. Поддерживает автоматы с одно- и двумерными квадратными решётками и декартовыми метриками. Позволяет использовать различные локальные правила, использовать множества взаимодействующих решёток, использовать структурированные значения состояний клеток. Удобный инструментарий для наблюдения за экспериментами.

- *LCAU.* Автор: Х. В. Макинтош (H. V. Macintosh);

  Адрес: http://delta.cs.cinvestav.mx/~mcintosh/newweb/software.html;

  Распространяется свободно;

Требования к среде выполнения: рабочая станция с операционной системой семейства MS-DOS или станция NeXTSTEP;

Программа предоставляет качественно новые и важные средства исследования клеточных автоматов. В частности, с её помощью можно находить циклы в изменении состояния решётки, строить диаграммы де Брюна, и определять, с некоторой вероятностью, предыдущее состояние решётки.

- *SCARLET.* Авторы: М. Кутриб (M. Kutrib) и др.;

  Адрес: http://www.informatik.uni-giessen.de/staff/kutrib.html;

  Требования к среде выполнения: рабочая станция с операционной системой Sun Solaris 2.x;

  Программа поддерживает автоматы с произвольными решётками и декартовыми метриками. В качестве состояний клеток могут быть использованы целые числа и строки.

- *WinCA.* Авторы: Б. Фиш (B. Fisch) и Д. Гриффит (D. Griffeath);

  Адрес: ftp://cam8.math.wisc.edu/pub/winca_10.exe;

  Распространяется свободно;

  Требования к среде выполнения: рабочая станция с операционной системой семейства Windows или Windows NT;

  Программа поддерживает автоматы с двумерными решётками и декартовыми метриками.

Не смотря на такое изобилие существующего программного обеспечения, возникла потребность в разработке нового проекта для решения задач с помощью клеточных автоматов. Причины, послужившие этому, состоят в следующем:

1. На множество автоматов, с которыми может работать та или иная программа из перечисленных выше, накладываются жёсткие ограничения, как то: параметры и размерность решётки, множество возможных состояний клетки, форма окрестности, возможность реализовывать асинхронные и вероятностные автоматы и многие другие. Далеко не все из них упоминались при перечислении. Многие из этих параметров жёстко фиксируются и их невозможно изменить. Поэтому CAME&L было решено разрабатывать совершенно универсальным. Планировалась возможность изменять и настраивать любую составляющую автомата по желанию пользователя;

2. Большинство упомянутых выше программ разрабатывались более пяти лет назад. А некоторые и более десяти. Они не удовлетворяют современным требованиям, предъявляемым к пользовательскому интерфейсу, а также не оптимизированы под современные производительные вычислительные системы. Поэтому CAME&L было решено разрабатывать с учётом возможностей современных процессоров;
3. Многие из них имеют сложные языки описания автоматов, нетривиальный интерфейс, неудобные средства управления экспериментом. Поэтому CAME&L было решено разрабатывать максимально простым и интуитивно ясным для пользователя.

В разделе 2.3 будет подробнее рассмотрен вопрос о представлении и работе клеточных автоматов в среде CAME&L, а в разделе 2.4 будет описан пользовательский интерфейс среды.

2.3. Представление клеточных автоматов в среде CAME&L

Любой клеточный автомат в среде CAME&L представляется набором из четырёх так называемых компонентов. В этот набор входят:

- grid – решётка, которая отвечает за визуализацию состояний клеток автомата, а также обеспечивает взаимодействие с пользователем, изменение значений, хранящихся в клетках и т.п.;
- metrics – метрика, служащая для присвоения координат клеткам, измерения расстояния между ними, а также для определения окрестности клетки;
- datum – хранилище данных, которое обеспечивает размещение данных в памяти, управляет отображением данных решёткой, выполняет сохранение и восстановление данных из файла и обеспечивает, если это нужно, единовременность изменения состояний клеток решётки;
- rules – правила автомата, управляющие итерациями.

Каждый компонент реализуется с помощью библиотеки разработчика клеточных автоматов CADLib в виде класса, помещённого в динамическую библиотеку. CADLib предоставляет удобный инструментарий для разработки этих классов.

Для каждого типа компонентов в библиотеке есть соответствующий базовый класс, который должен быть родителем (или прародителем и т.д.) разрабатываемого компонента. Эти классы называются `CAGrid`, `CAMetrics`, `CADatum` и `CARules`, соответственно.

Помимо функциональности, специфичной для каждого конкретного компонента, все компоненты должны уметь предоставить информацию о себе, о своей совместимости с другими компонентами и прочее. Эти обязанности возложены на базовый класс `CAComponent`. Подробнее библиотека CADLib рассматривается в главе 3.

2.4. Интерфейс среды CAME&L

Среда CAME&L представляет собой многодокументное приложение для операционной системы Windows. Требования к среде выполнения: рабочая станция с операционной системой семейства Windows (Windows 98 или более новая (Windows 95 только с Internet Explorer 3.0 или более новым)) или Windows NT (Windows 2000 или более новая (Windows NT 4.0 только с Internet Explorer 3.0 или более новым)).

Каждый документ представляет собой среду для проведения эксперимента с использованием клеточных автоматов. На рис. 8 изображёно окно программы с открытым документом.

Окно документа разделено на две части. В правой части находится дерево доступных компонентов. На первом уровне дерева – имя класса компонентов, которому принадлежит данный. На втором – имена компонентов, а на третьем – их параметры, изменение значений которых позволяет настраивать компонент. Выбранные в данный момент компоненты выделяются кружком в начале названия. Жирным начертанием выделяются компоненты, совместимые с выбранными в данный момент.

В левой части отображается текущее состояние решётки клеточного автомата. Необходимо отметить, что она не будет отображаться до тех пор, пока пользователь не выберет совместимые компоненты решётки, метрики и хранилища данных.

Опишем справа налево поля строки состояния:

- индикатор работы вычислительного кластера (пока поддерживается только режим отдельной рабочей станции);
- индикатор выполнения текущей операции (зарезервировано для последующей реализации);
- индикатор прокрутки решётки (определяет управляет ли пользователь курсором или прокруткой решётки);
- поле вывода номера итерации;

Ø поле вывода информации и подсказок.

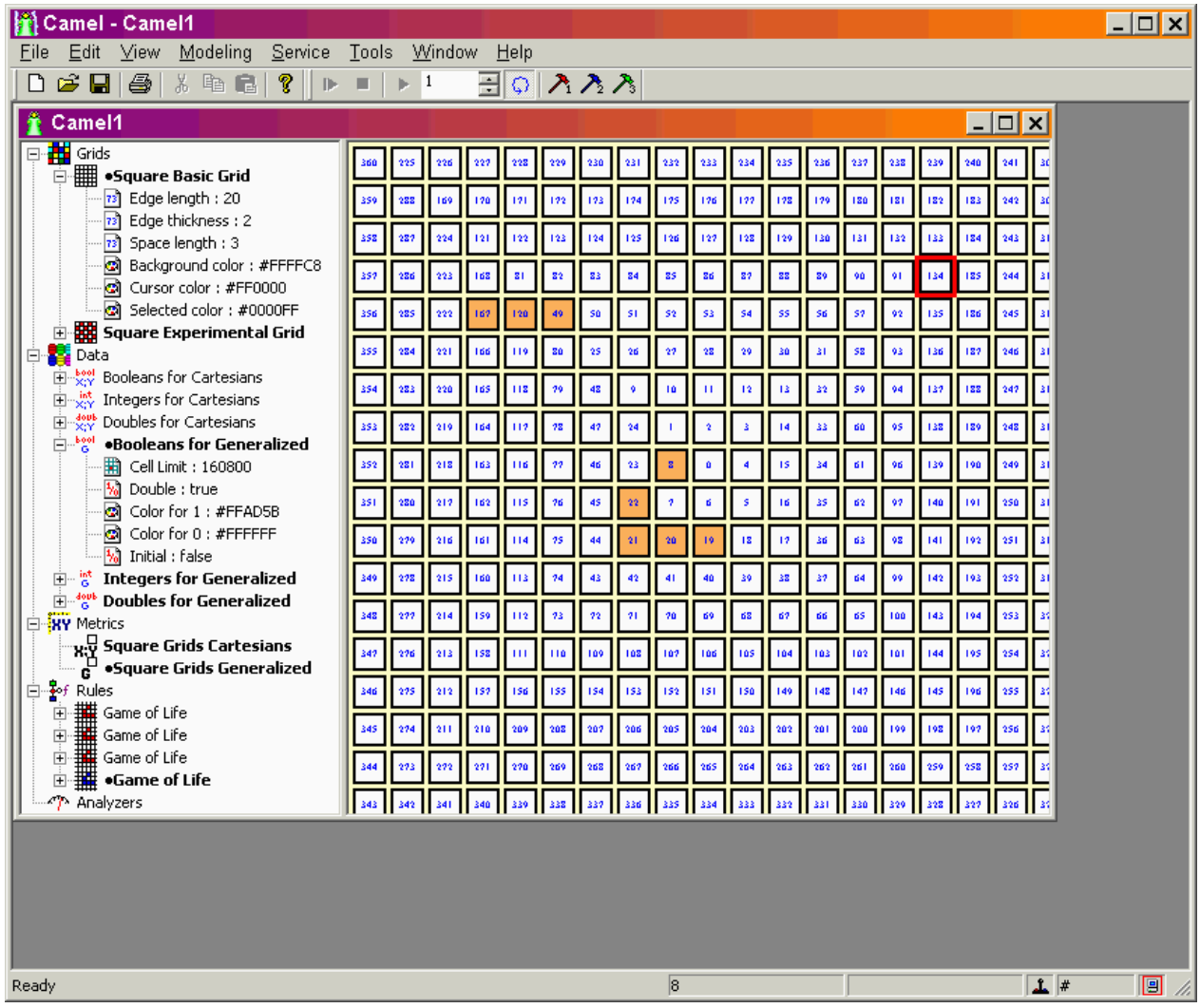

Рис 8. Окно среды CAME&L с открытым документом

Рассмотрим главное меню среды CAME&L. Приведём список опций меню File.

Ø New – создать новый документ;

Ø Open… – открыть файл документа. Среда CAME&L сохраняет документы в файлы с расширением cml, представляющие собой xml-документы [20]. Синтаксис этих файлов можно описать так:

```
<?xml version="1.0"?>
<!-- CAMEL Document File. Used CAMEL Framework Version 1.0 -->
```

```
<Document>
  <Components>
    <Metrics [Name="..."] Library="..." [Version="?,?,?,?"]>
      ...
      <Parameter [Name="..."] Index="?" Value="..."/>
      ...
    </Metrics>
    <Datum [Name="..."] Library="..." [Version="?,?,?,?"]
      (Data="Default")>
      ...
      <Parameter [Name="..."] Index="?" Value="..."/>
      ...
    </Datum>
    <Grid [Name="..."] Library="..." [Version="?,?,?,?"]>
      ...
      <Parameter [Name="..."] Index="?" Value="..."/>
      ...
    </Grid>
    <Rules [Name="..."] Library="..." [Version="?,?,?,?"]>
      ...
      <Parameter [Name="..."] Index="?" Value="..."/>
      ...
    </Rules>
  </Components>

  <Data (Name="...") [Portion="?"]>
    ...0
  </Data>
</Document>
```

Тело документа заключается между тегами `<Document> … </Document>`. Внутри документа может находиться любое количество тегов `<Components> … </Components>` и `<Data> … </Data>`. Между первыми должны размещаться те-

ги, описывающие выбранные компоненты, а именно <Metrics> … </Metrics>, <Datum> … </Datum>, <Grid> … </Grid> и <Rules> … </Rules>. У каждого из них возможны следующие атрибуты:

- Library – обязательный атрибут, указывающий путь (возможно, относительный) к библиотеке компонента;
- Name – необязательный атрибут, содержащий имя компонента. Нужен скорее для помощи пользователю, изучающему текст файла;
- Version – необязательный атрибут, содержащий версию библиотеки компонента в формате «?,?,?,?». Если атрибут указан и версия имеющегося в системе компонента – более ранняя, чем указанная в файле, то при открытии будет выведено сообщение о несоответствии версий;
- Storage – атрибут может присутствовать только для тега <Datum>…</Datum>. Он указывает имя тега <Data> … </Data>, данные из которого нужно использовать.

Между этими тегами размещаются теги <Parameter/>, содержащие значения параметров компонентов. Теги <Parameter/> имеют обязательные атрибуты Index (индекс данного параметра в списке параметров компонента) и Value (значение параметра), а также необязательный атрибут Name (имя параметра).

Тег <Data> … </Data> служит для сохранения состояния решётки. У него есть атрибут Name, содержащий имя хранилища, по которому на него может ссылаться атрибут Storage тега <Datum> … </Datum>. Если в качестве значения атрибуты Name указано «Autoload», и тег <Datum> … </Datum> не ссылается ни на какое хранилище, то будет загружено именно это состояние решётки. Кроме того, у тега <Data> … </Data> есть необязательный параметр Portion, указывающий по сколько символов в строке содержится в записи состояния решётки. Эти данные записываются с помощью символов с ASCII-кодами от 64 до 192. Любые другие символы игнорируются. Последним символом должен быть символ с нулевым кодом.

В документе может быть несколько хранилищ данных с разными именами.

CML-файл интерпретируется и выполняется средой по мере прочтения.

- Close – закрыть документ;
- Save и Save As… – сохранить текущий документ в CML-файле;

- Print… – распечатать документ;
- Print Preview – предварительно просмотреть документ перед печатью;
- Print Setup… – отобразить окно настройки печати;
- Exit – выйти из программы;

Список опций меню View.

- Components Tree – отображать ли дерево доступных компонентов;
- Cells Labels – показывать ли координаты клеток в текущей метрике;
- Visualize Grid – изображать ли решётку автомата. При проведении эксперимента, после настройки модели рекомендуется отключить визуализацию состояния решётки автомата для повышения производительности системы;
- Cursor Position… – отобразить окно выбора позиции курсора;
- Position on Grid… – отобразить окно выбора позиции на решётке (запрашиваются координаты клетки, находящийся в верхнем левом углу отображаемой области);
- Toolbars – меню для выбора того, какие панели инструментов следует отображать.
    - Standard – отображать ли стандартную панель инструментов;
    - Modeling – отображать ли панель инструментов, служащую для проведения экспериментов;
- Status bar – отображать ли строку состояния;

Список опций меню Modeling.

- Start Experiment – начать эксперимент;
- Finish Experiment – окончить эксперимент;
- Go – выполнить итерацию автомата или начать последовательности шагов, если включён режим автоматического выполнения итераций;
- Auto – включить/выключить режим автоматического выполнения итераций;
- First Tool Function – вызвать первую вспомогательную функцию, предоставляемую компонентом правил автомата;
- Second Tool Function – вызвать вторую вспомогательную функцию, предоставляемую компонентом правил автомата;
- Third Tool Function – вызвать третью вспомогательную функцию, предоставляемую компонентом правил автомата;

Список опций меню Service.

Ø Store data – сохранять ли состояние решётки в CML-файл или же сохранять лишь сведения о выбранных компонентах.

Список опций меню Tools.

Ø Components Manager – отобразить окно менеджера компонентов (рис. 9).

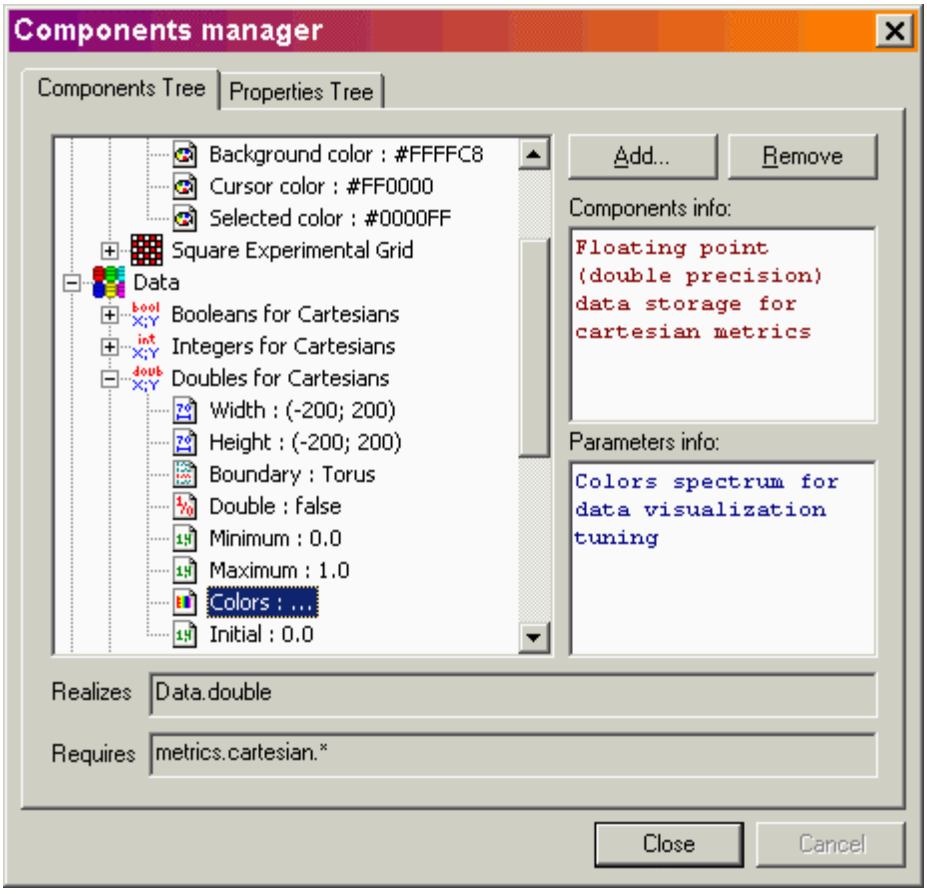

Рис 9. Окно менеджера компонентов

Страница Components Tree содержит дерево компонентов такое же, как в любом окне документа среды. Это средство позволяет:

- просмотреть дерево;
- добавить в него компонент с помощью кнопки Add…;
- удалить существующий компонент с помощью кнопки Remove;
- получить описание компонента, его параметров, а также информацию о совместимости компонентов с другими, для чего служат четыре текстовых поля;

Страница Properties Tree позволяет просмотреть дерево свойств, используемых компонентами при обмене информацией о совместимости.

- Options… – отобразить окно настройки приложения;

Функциональность меню Window и Help стандартная.

Некоторые пункты описанных выше меню вынесены на панель инструментов.

ГЛАВА 3. БИБЛИОТЕКА РАЗРАБОТЧИКА КЛЕТОЧНЫХ АВТОМАТОВ CADLib

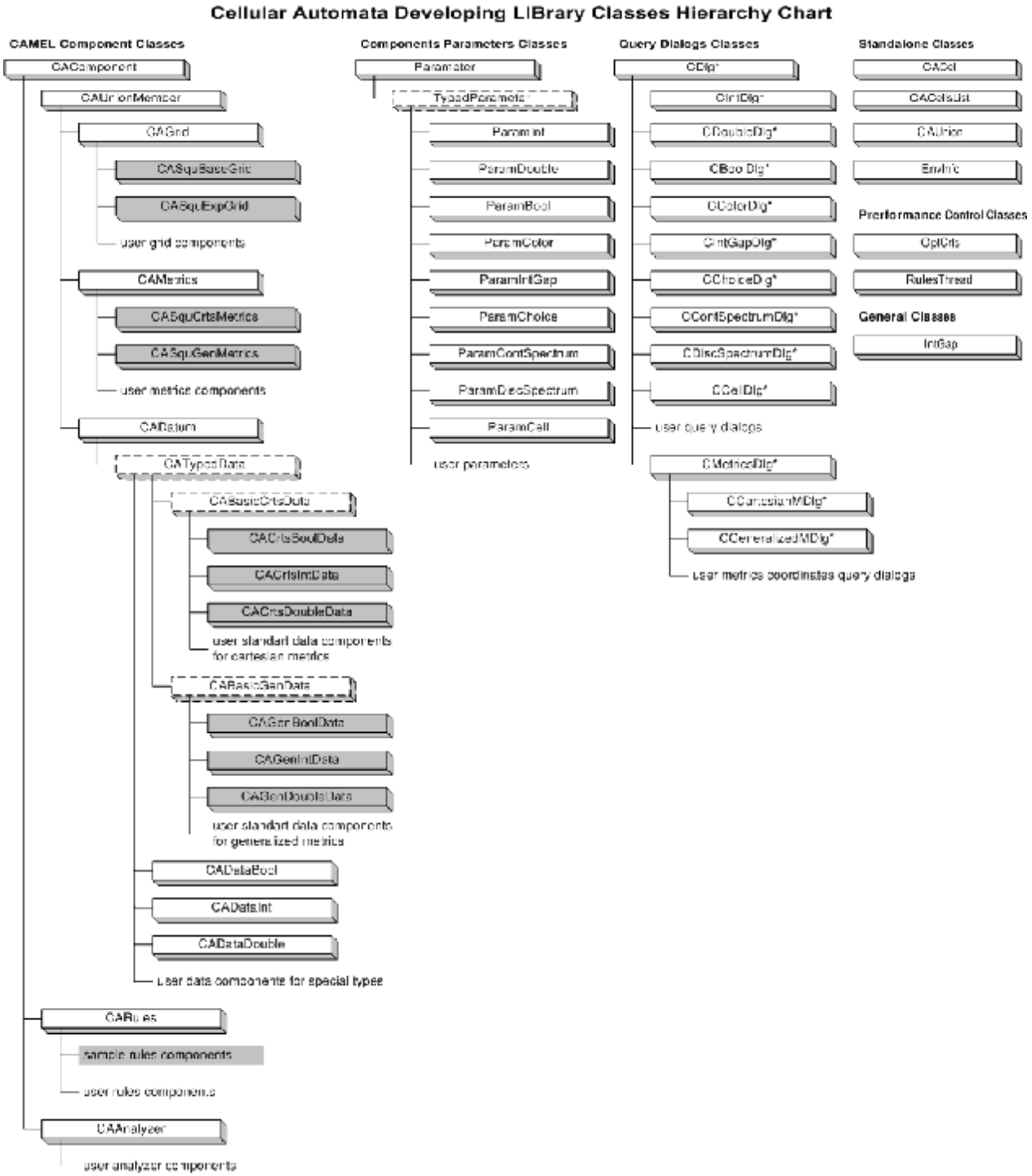

Рис 10. Диаграмма классов библиотеки CADLib

В настоящей главе описывается одна из важнейших частей программного обеспечения CAME&L, библиотека CADLib (Cellular Automata Developers LIBrary – библиотека разработчика клеточных автоматов). Она служит непосредственно для разработки клеточных автоматов, решающих конкретные задачи. Библиотека содержит базовые классы, для разработки новых компонентов клеточных автоматов (см. раздел 2.3).

Диаграмма классов библиотеки изображена на рис. 10. Параллелограммы с пунктирными границами изображают шаблоны классов [12]. Серые фигуры изображают реализованные компоненты, поставляемые вместе со средой CAME&L, которые могут быть использованы для проведения экспериментов. Функциональности этих компонентов достаточно для решения большинства задач. Соответствующие им классы распространяются вместе с исходным кодом, однако в библиотеку CADLib они не входят и поэтому в настоящей главе рассматриваться не будут.

Большинство классов библиотеки разрабатывались так, чтобы они на уровне исходного кода не зависели от платформы. То есть в них, по возможности, используются только стандартные библиотеки и библиотека STL (Standard Template Library – стандартная библиотека шаблонов) [12]. Однако этого удалось достичь. Те классы, которые используют библиотеку MFC (Microsoft Foundation Classes – основные классы Microsoft) [21], отмечены на диаграмме (рис. 10.) знаком «*».

Кроме того, на диаграмме специально отмечены места, которые могут занимать классы, создаваемые пользователем и сторонними разработчиками.

Помимо классов, в библиотеке CADLib содержатся функции, константы и макроопределения, упрощающие разработку наследников.

3.1. Базовые классы компонентов

В настоящем разделе будут описаны классы, функциональность которых необходимо хорошо себе представлять для разработки новых компонентов клеточных автоматов.

3.1.1. Класс `CAComponent`

Иерархия базовых классов компонентов берёт своё начало с класса `CAComponent`. Он реализует функциональность, общую для всех компонентов. Опишем этот класс. Основные поля и функции-члены:

- `public CAComponent()` – конструктор;
- `public ~CAComponent()` – деструктор;
- `protected int m_iType` – поле для хранения типа компонента. Оно может принимать одно из следующих значений:
  - `CT_COMPONENT` –абстрактный компонент. Среда проигнорирует его;
  - `CT_GRID` – решётка;
  - `CT_DATUM` – хранилище данных;
  - `CT_METRICS` – метрика;
  - `CT_RULES` – правила.
- `public inline UINT GetType()` – возвращает тип компонента.

Функции-члены, которые предоставляют информацию о компоненте. Их можно или нужно переопределять в классах-наследниках:

- `public virtual inline STRING GetName()` – возвращает имя компонента. Функция должна быть переопределена;
- `public virtual inline STRING GetInfo()` – возвращает информацию о компоненте. Функция должна быть переопределена;
- `public virtual inline LPTSTR GetIconID()` – возвращает идентификатор ресурса, содержащего иконку для данного компонента. Если вернёт `NULL`, то среда будет использовать иконку по умолчанию. Функция по умолчанию возвращает `NULL`;
- `public virtual inline STRING GetRealize()` – возвращает набор свойств, которые реализует настоящий компонент, разделённых точкой с запятой. Каждое свойство представляет собой последовательность слов, разделённых точкой. Каждое следующее слово уточняет предыдущее. Например, для компонента, представляющего собой хранилище булевых данных, функция может вернуть «Data.bool». Никаких ограничений на то, какие слова могут выступать в качестве свойств, нет. По умолчанию функция возвращает пустую строку, что обеспечит компоненту совместимость со всеми остальными компонентами;
- `public virtual inline STRING GetRequire()`– возвращает выражение, описывающее набор свойств, которыми должны обладать другие компоненты для со-

вместимости с данным. Выражение принадлежит языку, задаваемому следующей грамматикой:

```
<EXPRESSION> ::= (<XOR_EXPRESSION>)
<EXPRESSION> ::= !<XOR_EXPRESSION>
<XOR_EXPRESSION> ::= <OR_EXPRESSION>^<OR_EXPRESSION>
<XOR_EXPRESSION> ::= <OR_EXPRESSION>
<OR_EXPRESSION> ::= <AND_EXPRESSION>|<AND_EXPRESSION>
<OR_EXPRESSION> ::= <AND_EXPRESSION>
<AND_EXPRESSION> ::= <EXPRESSION>&<EXPRESSION>
<AND_EXPRESSION> ::= <EXPRESSION>
```

Каждое выражение (лексема `<EXPRESSION>`) может представлять с тобой свойство или же начало свойства, заканчивающееся на «.*» или «.?». Первый суффикс понимается, как любое, в том числе и нулевое, количество слов. Второй – как любое ненулевое количество слов. Например, если компонент требует хранилище булевых данных и метрику с обобщёнными координатами, то функция должна вернуть строку «Data.bool.*&Metrics.generalized.*». По умолчанию функция возвращает строку «*», что обеспечит всем компонентам совместимость с данным;

Функции-члены для поддержки работы с параметрами. Параметры это – наследники класса `Parameter`, которые предоставляют пользователю возможность настраивать компонент. В среде предусмотрена возможность изменения их значений. Самому компоненту также предоставляется возможность узнавать об их изменении и следить за их значениями.

- `public virtual inline WORD GetParametersCount()` – возвращает число параметров данного компонента. По умолчанию функция возвращает ноль.
- `public virtual inline Parameter* GetParameter(WORD n)` – возвращает указатель на n-ый параметр компонента. Число n лежит на отрезке от нуля до результата, возвращаемого `GetParametersCount()`–1. По умолчанию функция возвращает `NULL`.

  Для удобства реализации двух последних функции введены макроопределения `PARAMETERS_COUNT` (определяет число параметров), `BEGIN_PARAMETERS` (начинает список параметров), `PARAMETER` (добавляет параметр с таким-то номером в список параметров) и `END_PARAMETERS` (заканчивает список параметров), позволяю-

щие легко составить, так называемую карту параметров. Для этого достаточно в public-секции класса компонента написать:

```
PARAMETERS_COUNT(N)
BEGIN_PARAMETERS
    PARAMETER(0,&p_Par1)
    PARAMETER(1,&p_Par2)
    ...
    PARAMETER(N-1,&p_ParN)
END_PARAMETERS
```

- `public virtual void ParameterChanged(Parameter* pPar)` – функция вызывается ядром, когда в среде произошло изменение параметра, на который указывает `pPar`. Ещё функция вызывается с параметром `NULL` при инициализации компонента, когда он выбран впервые в данном документе.

3.1.2. Класс `CAUnionMember`

Три компонента клеточных автоматов, решётка, метрика и хранилище данных, должны функционировать слаженно, взаимодействуя во время выполнения определённых операций. Они образуют, так называемый, союз компонентов. Функционирование этого союза поддерживается на уровне их общего предка, класса `CAUnionMember`.

В библиотеке CADLib существует класс `CAUnion`, обеспечивающий существование и взаимосвязь упомянутого выше союза компонентов. Опишем этот класс:

- `public CAGrid* Grid` – поле, хранящее указатель на компонент решётку;
- `public CADatum* Datum` – поле, хранящее указатель на компонент хранилище данных;
- `public CAMetrics* Metrics` – поле, хранящее указатель на компонент метрику;
- `public CAUnion()` – конструктор, инициализирующий указатели на все компоненты значением `NULL`;
- `public inline bool IsTrioOK()` – функция, возвращающая `true`, если указатели на все компоненты не равны `NULL`, то есть существует союз выбранных пользователем компонентов. Возвращает `false` в противном случае.

Опишем теперь класс `CAUnionMember`. К средствам поддержки союза компонентов относятся следующие:

- `protected void* m_pUnion` – поле, указывающее на соответствующий объект класса `CAUnion`;
- `public void SetUnion(void* pUnion)` – функция, устанавливающая значение предыдущего поля `m_pUnion`. Её вызывает среда, устанавливая значение соответствующее данному документу;

Кроме того, класс обеспечивает следующие функции:

- `public CAUnionMember()` – конструктор;
- `public ~CAUnionMember()` – деструктор;
- `private bool m_bWasInit` – поле, показывающее, проводился ли цикл инициализации данного компонента. Одной из основных операций при инициализации является вызов функции-члена `ParameterChanged` с параметром `NULL`. Потребность в данном поле объясняется тем, что инициализация должна производиться единожды;
- `public inline bool GetWasInit()` – возвращает значение поля `m_bWasInit`. Эта функция используется средой и не должна вызываться пользовательскими библиотеками;
- `public inline void ResetWasInit()` – присваивает полю `m_bWasInit` значение `true`. Обратное присвоение уже невозможно. Эта функция используется средой и не должна вызываться пользовательскими библиотеками.

Рассмотрим теперь трёх основных наследников класса `CAUnionMember`.

3.1.3. Класс `CAGrid`

Класс `CAGrid` является базовым для всех компонентов решёток. Напомним, что данный компонент отвечает за визуализацию состояния решётки. Опишем этот класс:

- `public CAGrid()` – конструктор;
- `protected CDC* m_pDC` – контекст устройства для отрисовки решётки;
- `protected CSize* m_pszSize` – размер изображения решётки в пикселях;
- `protected CSize* m_pszVArea` – размер видимой части изображения решётки в пикселях. Он не может превосходить значение `m_pszSize`;

- `public void SetOutput(CDC* pDC,CSize* pszSize,CSize* pszVArea)` – вызывается средой для того, чтобы установить значения последних трёх полей `m_pDC`, `m_pszSize` и `m_pszVArea`. Функция не должна вызываться пользовательскими библиотеками;
- `public CACell m_RefCell` – ссылочная клетка, то есть левая верхняя клетка в видимой области решётки. Класс `CACell` служит для описания клетки в произвольной метрике;
- `public CACell m_CurCell` – клетка, на которой располагается курсор;
- `public CACellsList m_SelCells` – список клеток, выделенных в настоящий момент. Класс `CACellsList` представляет собой список объектов класса `CACell`;
- `public bool m_bVizualize` – отображать ли состояние решётки (если значение `true`) или нет, для ускорения итераций. В последнем случае можно отображать произвольное изображение с надписью, что визуализация выключена;
- `public bool m_bShowLabels` – показывать ли координаты клеток;

Функции-члены класса, которые должны быть переопределены пользователями, для того, чтобы среда могла вызывать их:

- `public virtual void Paint()` – самая часто вызываемая функция компонента. Она обеспечивает перерисовку решётки. Параметры перерисовки определяются значениями полей `m_pDC`, `m_pszSize`, `m_pszVArea`, `m_RefCell`, `m_CurCell`, `m_SelCells`, `m_bVizualize` и `m_bShowLabels`. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual CACell HitTest(int x,int y,int& type)` – позволяет определить, что находится на решётке в точке с координатами (`x`; `y`). Через ссылку `type` функция может вернуть одно из следующих значений:
  - `HT_NONE` – в точке не находится ничего. Она соответствует промежутку между клетками. Значение, которое вернёт сама функция, никак не интерпретируется;
  - `HT_FRAME` – в точке находится рамка клетки. Значение, которое вернёт сама функция, никак не интерпретируется;

- HT_CELL – в точке находится сама клетка. Сама функция вернёт координату соответствующей клетки.

Функция должна быть переопределена в пользовательской библиотеке;

- Ø `public virtual void MoveGrid(int dir, bool repaint=true)` – переместить текущее положение на решётке в направление, указанное параметром `dir`. Этот параметр может принимать одно из следующих значений:
  - DIR_UP – сдвиг вверх;
  - DIR_DOWN – сдвиг вниз;
  - DIR_LEFT – сдвиг вправо;
  - DIR_RIGHT – сдвиг влево;
  - DIR_PAGEUP – сдвиг вверх на страницу;
  - DIR_PAGEDOWN – сдвиг вниз на страницу;
  - DIR_PAGELEFT – сдвиг влево на страницу;
  - DIR_PAGERIGHT – сдвиг вправо на страницу.

  Перемещение текущего положения обеспечивается изменением значения ссылочной клетки, координата которой хранится в поле `m_RefCell`. Если значение `repaint` истинно, то после сдвига будет вызвана функция `Paint()`. Функция должна быть переопределена в пользовательской библиотеке;
- Ø `public virtual void MoveCursor(int dir,bool shift)` – сдвинуть курсор в направлении указанном параметром `dir`. Его возможные значения приведены в описании предыдущей функции. Нужно стараться реализовать эту функцию так, чтобы курсор всегда был в видимой части решётки. Если параметр `shift` равен истине, то за курсором должен (или может) оставаться след из выделенных клеток. Функция должна быть переопределена в пользовательской библиотеке.

Далее следуют функции поддержки выделения клеток. При желании их можно не переопределять.

- Ø `public virtual void Select(CACell cell,bool select)` – выделить клетку `cell`, если параметр `select` равен истине и снять выделения с клетки, если `select` равен лжи. Базовую реализацию функции содержит макроопределение IMPLEMENT_SELECTABLE. По умолчанию функция не делает ничего;

- `public virtual void SelectRegion(CACell from,CACell to)` – выделить диапазон клеток от клетки `from` до клетки `to`. Функция вызывается, например, если пользователь щёлкнул мышью по какой-либо клетке, и при этом была нажата клавиша «Shift». Тогда в качестве параметров функции будут переданы координаты курсора и координаты клетки, на которую пришёлся щелчок. По умолчанию функция не делает ничего;
- `virtual void UnSelect()` – снять выделение со всех клеток. Базовую реализацию функции содержит макроопределение `IMPLEMENT_SELECTABLE`. По умолчанию функция не делает ничего.

3.1.4. Класс `CAMetrics`

Класс `CAMetrics` является базовым для всех компонентов метрик. Напомним, что данный компонент отвечает за нахождение соседей клетки, присвоение клеткам координат, вычисление расстояния между клетками. Опишем этот класс:

- `public CAMetrics()` – конструктор;
- `public virtual DWORD GetDistance(CACell c1,CACell c2)` – вычислить расстояние между клетками `c1` и `c2`. По умолчанию функция возвращает ноль. Функцию можно не реализовывать, но она может оказаться удобной при написании правил автомата;
- `public virtual inline LONG GetDistanceX(CACell c1,CACell c2)` – возвращает расстояния между проекциями клеток на ось абсцисс. Значение, возвращаемое функцией – знаковое. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual inline LONG GetDistanceY(CACell c1,CACell c2)` – возвращает расстояния между проекциями клеток на ось ординат. Значение, возвращаемое функцией – знаковое. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual inline CACell GetOrigin()` – возвращает координаты клетки, помещаемой в центр решётки при инициализации. По умолчанию функция возвращает ноль;

- `public virtual inline WORD NeighboursCount()` – возвращает число соседей каждой клетки в данной метрике. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual CACell GetNeighbour(CACell c,WORD neig)` – возвращает клетке, соседнюю с клеткой `c`. То, какого именно соседа необходимо вернуть определяется параметром `neig`, идентификатором соседа. Это – целое число от 0 до результата выполнения функции `NeighboursCount()`–1. Удобные идентификаторы соседних клеток можно найти в файле «Neighbours.h» в библиотеке CADLib. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual void GetNeighbours(CACell c,CACell* pNeig)` – возвращает через указатель `pNeig` массив из `NeighboursCount()` элементов, представляющих собой координаты соседей данной клетки. Как привило, эта функция будет работать быстрее, чем вариант, при котором все соседи будут запрашиваться по одному с помощью функции `GetNeighbour(c,neig)`. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual bool Query(CWnd* pParent, CACell& c)` – отображает окно для запроса координаты клетки в данной метрики. Параметр `c` определяет ссылку на клетку, значение координаты которой нужно изменить. Параметр `pParent` определяет окно, родительское для отображаемого окна. Возвращает истину в случае удачного выполнения и ложь в противном случае. Для создания этого диалога можно использовать наследование от класса `CMetricsDlg`. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual void ToString(CACell c,STRING s)` – возвращает строковое представление координаты клетки `c` через параметр `s`. Функция должна быть переопределена в пользовательской библиотеке;

3.1.5. Классы `CADatum` и `CATypedData`

Класс `CADatum` является базовым для всех компонентов хранилищ данных. Напомним, что данный компонент отвечает за хранение состояний всех клеток решётки в памяти, обеспечивает двойную буферизацию состояния решётки, его сохранение в файл и восстановление из файла. Помимо всего вышеперечисленного, этот компонент взял на себя неко-

торые аспекты визуализации данных, а именно – преобразование состояния решётки в восемь байт, которые, в большинстве случаев, могут быть проинтерпретированы, как два дескриптора цвета [21]. Опишем этот класс:

- `public CADatum()` – конструктор;
- `public bool m_bComputation` – равно `true` во время итерации и `false` в остальное время. Это значение используется шаблоном классов `CATypedData`, который обсуждается далее;
- `public virtual inline COLORREF GetCellColor(CACell c)` – возвращает дескриптор цвета, который будет использоваться для отображения тела клетки с координатой `c`. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual inline COLORREF GetPlaceColor(CACell c)` – возвращает дескриптор цвета, который будет использоваться для отображения рамки клетки с координатой `c`. Таким образом реализуется, так называемая, «память места». По умолчанию функция возвращает дескриптор, соответствующий чёрному цвету. То есть, по умолчанию, механизм «памяти места» не используется;
- `public virtual bool Query(CWnd* pParent, long x, long y, CACell c, bool selected)` – отображает окно для запроса координаты значения для данной клетки или клеток. Параметр `pParent` определяет окно, родительское для отображаемого окна. Параметры `x` и `y` определяют положение окна относительно родительского. Если параметр `selected` равен лжи, то параметр `c` определяет на клетку, значение в которой нужно изменить. Если значение `selected` равно истине, то результат запроса будет присвоен всем, выделенным в данный момент, клеткам. Возвращает истину в случае удачного выполнения и ложь в противном случае. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual void Swap()` – функция должна обеспечивать переключение между двумя буферами данных при моделировании синхронного автомата. Эта функция вызывается ядром в следующих случаях:
  - до и после инициализационного цикла правил автомата;
  - перед каждой итерацией;
  - до и после финализационного цикла правил автомата.

По умолчанию функция не делает ничего. Если хранилище не должно поддерживать двойную буферизацию данных, то не стоит менять реализацию по умолчанию;

- `public virtual inline bool CellExists(CACell c)` – возвращает истину, если клетка с координатой `c` существует и ложь в противном случае. Эта функция позволяет использовать решётки с витиеватым взаиморасположением клеток. По умолчанию функция возвращает истину;
- `public virtual void Save(IOStream* st)` – сохраняет состояние решётки в поток, на который указывает `st`. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual bool Load(IOStream* st)` – восстанавливает состояние решётки из потока, на который указывает `st`. Возвращает истину в случае удачного выполнения и ложь в противном случае. Функция должна быть переопределена в пользовательской библиотеке;

Однако, наследников непосредственно класса `CADatum` создавать не стоит. Так как понятие хранилище данных непосредственно связано с типом данных, которые она хранит, возникла необходимость создания шаблона `CATypedData<class type>`, добавляющего к описанному выше классу функции, необходимые для работы с типом данных `type`. Опишем этот класс:

- `public virtual inline type Get(CACell c)` – возвращает значение, хранящееся в клетке `c`. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual inline void Set(CACell c,type value)` – помещает в клетку `c` значение `value`. Возвращает истину в случае удачного выполнения и ложь в противном случае. Функция должна быть переопределена в пользовательской библиотеке;

Создавая новые хранилища данных можно использовать класс `CATypedData` в качестве родительского класса и создавать хранилища для новых типов данных или же воспользоваться классами `CADataBool`, `CADataInt` и `CADataDouble` для булевых, целочисленных и вещественных данных соответственно.

Написание компонентов хранилищ данных – непростая задача, при решении в которой, однако, можно выделить ряд стандартных приёмов. Именно эти приёмы использо-

вались при написании шаблонов классов `CABasicCrtsData<class type, class dlg, int idd>` и `CABasicGenData <class type, class dlg, int idd>`. Эти шаблоны позволяют создавать хранилища для структур данных с декартовой и обобщённой метрикой, соответственно. Параметр `type` определяет тип хранимых данных. Параметр `dlg` – класс диалога (наследник класса `CDlg`) для запроса значений типа `type`. Параметр `idd` – идентификатор ресурса библиотеки, содержащего этот диалог.

Эти два шаблона берут на себя большую часть работы и реализуют функции члены `Swap()`, `CellExists(c)`, `Get(c)`, `Set(c, value)`, `Query(pParent, x, y, c, selected)`, `Save(st)` и `Load(st)`. Помимо этого они вводят ряд важных для компонента хранилища параметров. Если не требуется некой специфической, экономичной структуры хранения данных, то эти шаблоны – большое подспорье при разработке компонентов. На откуп разработчику отдаётся лишь функция `GetCellColor(c)`. Однако, справедливости ради, надо отметить, что порой приходится переопределять и другие функции члены, обращаясь при этом к их реализации в шаблонах.

3.1.6. Класс `CARules`

Класс `CARules` является базовым для всех компонентов правил. Напомним, что данный компонент отвечает за выполнение итерации автомата и другие преобразования состояния решётки автомата. Опишем этот класс:

- `public CARules()` – конструктор;
- `protected CAUnion* m_pUnion` – указатель на союз компонентов, с которыми должен работать данный компонент;
- `public void SetUnion(CAUnion* pUnion)` – устанавливает значение поля `m_pUnion`;
- `public EnvInfo* m_pEnvInfo` – объект класса `EnvInfo`, служащего для описания платформы и окружения, в котором выполняются вычисления. Эта информация может быть проанализирована и учтена функциями-членами класса `CARules` по их усмотрению;
- `public void SetEnvInfo(EnvInfo* pEnvInfo)` – устанавливает значение поля `m_pEnvInfo`;

- `public bool m_bFinalizeIfChanged` – если значение этого поля равно `true`, то после любого изменения состояния решётки пользователем, потребуется повторный вызов функции инициализации `Initialize()`, описанной ниже. Эта возможность необходима при использовании классов оптимизации производительности (таких, как класс `OptCtrs`);

Далее описываются функции для управления итерациями и состоянием решётки, которым могут или должны быть переопределены пользователем. Основные три из них описываются ниже:

- `public virtual void Initialize()` – реализует инициализационный цикл эксперимента. По умолчанию функция не делает ничего;
- `public virtual bool Compute()` – реализует правила автомата. Возвращает `true`, пока эксперимент должен продолжаться. Когда функция вернёт `false` эксперимент перейдёт в стадию финализации. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual void Finalize()` – реализует финализационный цикл эксперимента. По умолчанию функция не делает ничего;

Кроме этого для управления состоянием решётки в классе предусмотрены следующие функции:

- `public virtual bool SubCompute(CACell a1, CACell b1, CACell a2, CACell b2,CACell a3, CACell b3)` – функция может вызываться функцией `Compute()` для распараллеливания вычислений. Она вызывается классом `RulesThread`, предоставляющим функциональность потока для параллельного выполнения итерации. Шесть параметров этой функции должны описать ту часть задачи, которую нужно решить в данном потоке. Они могут быть проинтерпретированы, как три отрезка [`ai`; `bi`]. Возвращаемое функцией значение может быть проинтерпретировано как угодно. По умолчанию функция возвращает `true`;
- `public virtual void Tool1()` – реализует функциональность первого инструмента, который может вызвать пользователь из среды. Это инструмент предоставляет возможность некого преобразования или анализа состояния решётки. По умолчанию функция отображает окно с сообщением о том, что она не была переопределена;

- `public virtual void Tool2()` – реализует функциональность второго инструмента, который может вызвать пользователь из среды. Это инструмент предоставляет возможность некого преобразования или анализа состояния решётки. По умолчанию функция отображает окно с сообщением о том, что она не была переопределена;
- `public virtual void Tool3()` – реализует функциональность третьего инструмента, который может вызвать пользователь из среды. Это инструмент предоставляет возможность некого преобразования или анализа состояния решётки. По умолчанию функция отображает окно с сообщением о том, что она не была переопределена.

В случае если пользователю нужно более трёх инструментальных функций можно ввести фиктивные параметры компонента и воспользоваться функцией `ParameterChanged()`.

3.2. Классы параметров компонентов и диалогов для запросов

Базовым классом для всех классов параметров компонентов является класс `Parameter`. Опишем его:

- `protected STRING m_sName` – поле, хранящее имя параметра;
- `protected STRING m_sInfo` – поле, хранящее описание параметра;
- `protected void* m_pComp` – поле, хранящее указатель на компонент, владеющий этим параметром;
- `public Parameter(STRING name,STRING info,void* pComp=NULL)` – конструктор, заполняющий поля m_sName, m_sInfo и m_pComp;
- `public ~Parameter()` – деструктор;
- `public virtual inline STRING GetName()` – возвращает имя компонента;
- `public virtual inline STRING GetInfo()` – возвращает описание компонента;
- `public virtual HICON GetIcon()` – возвращает дескриптор иконки, соответствующей компоненту. Если вернёт `NULL`, то будет использована иконка по умолчанию. Для разработки единообразных иконок, в качестве основы для новых изобра-

жений пользователю рекомендуется использовать файл <путь к CADLib>\Pics\Param.ico;

Далее перечислим функции, которые подлежат обязательному переопределению в классах-наследниках.

- `public virtual void GetString(STRING str)` – через параметр `str` возвращает строковое представление значения параметра для отображения в дереве компонентов. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual void Store(STRING s)` – через параметр `s` возвращает строковое представление значения параметра для сохранения в файле. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual bool Restore(STRING s)` – восстанавливает значение параметра по строке, переданной через параметр `s`, полученной с помощью функции `Store(s)`. Возвращает истину в случае успеха и ложь в противном случае. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual bool Query(CWnd* pParent, long x, long y)` – отображает окно для запроса значения параметра над родительским окном `pParent` с верхним левым углом в точке с координатами (`x`; `y`). Возвращает истину в случае успеха и ложь в противном случае. Функция должна быть переопределена в пользовательской библиотеке;
- `virtual inline void ReleaseIcon()` – вызывается средой, когда нужно освободить ресурс, хранящий иконку, соответствующую компоненту.

Однако не следует наследовать новые классы параметров непосредственно от класса `Parameter`. Для этих целей служит его наследник, шаблон классов `TypedParameter<class type, int icon, int dlg, class dlgclass>`, позволяющий быстро разрабатывать параметры для новых типов данных. Аргумент шаблона `type` определяет тип значения параметра, `icon` – идентификатор ресурса, хранящего соответствующую иконку в библиотеке ресурсов CADLib, `dlg` – идентификатор ресурса, хранящего диалог для запроса значений, а `dlgclass` – класс этого диалога (наследник `CDlg`).

В описываемом шаблоне представлены стандартные реализации конструктора, а также функций `ReleaseIcon()`, `GetIcon()`, `Query(pParent, x, y)`. Кроме того, в этом классе дополнительно вводятся следующие функции:

- `public virtual inline type Get()` – возвращает значение параметра. Функция должна быть переопределена в пользовательской библиотеке;
- `public virtual inline bool Set(type value)` – устанавливает значение параметра равное `value`. Возвращает истину в случае успеха и ложь в противном случае. Функция должна быть переопределена в пользовательской библиотеке.

Для направления запросов пользователю следует использовать диалоги, функциональность которых реализуют наследники класса CDlg. Опишем этот класс:

- `public bool m_bResult` – поле, получающее значение «истина» в случае успешного запроса и «ложь» в противном случае;
- `public bool m_bParam` – если поле хранит истину, то диалог был создан для запроса значения параметра. Иначе – для запроса значения состояния некой клетки. Конструкторы класса `CDlg` присваивают полу истинное значение. Если класс-наследник реализует функциональность диалога для запроса состояния клетки, то в нём нужно предусмотреть соответствующий конструктор;
- `public CWnd* m_pParent` – указатель на родительское окно;
- `protected long m_x` – абсцисса верхнего левого угла при создании диалога;
- `protected long m_y` – ордината верхнего левого угла при создании диалога;
- `protected CDlg(CWnd* pParent)` – конструктор, создающий диалог с родительским окном `pParent`;
- `public CDlg(CWnd* pParent, long x, long y)` – конструктор, создающий диалог с родительским окном `pParent` и верхним левым углом в точке (`x`; `y`);
- `private virtual BOOL OnInitDialog()` – обработчик инициализации диалога. Эта функция не должна вызываться пользователем;
- `private afx_msg void OnDestroy()` – обработчик уничтожения диалога. Эта функция не должна вызываться пользователем.

Далее следуют функции, переопределяемые пользователем:

- `public virtual void PrePrompt()` – обработчик, вызываемый перед отображением диалога. В нём следует инициализировать элементы управления и т.п. Функция должна быть переопределена в пользовательской библиотеке;

- `private virtual void OnOK()` – обработчик успешного завершения запроса (пользователь нажал кнопку «OK»). Эта функция должна анализировать введённое значение и присваивать его параметру или состоянию клетки;
- `public virtual void PostPrompt()` – обработчик, вызываемый после закрытия диалога;

Стандартные классы параметров и соответствующие им классы диалогов перечислены в таблице 1.

Таблица 1. Стандартные классы параметров и соответствующие им классы диалогов

Класс параметра	Класс диалога	Тип параметра
`ParamInt`	`CIntDlg`	Целое число
`ParamDouble`	`CDoubleDlg`	Число с плавающей точкой (двойной точности)
`ParamBool`	`CBoolDlg`	Булева переменная
`ParamColor`	`CColorDlg`	Цветовой дескриптор
`ParamIntGap`	`CIntGapDlg`	Отрезок с целочисленными границами
`ParamChoice`	`CChoiceDlg`	Конечное множество альтернатив
`ParamContSpectrum`	`CContSpectrumDlg`	Непрерывный цветовой спектр
`ParamDiscSpectrum`	`CDiscSpectrumDlg`	Дискретный цветовой спектр
`ParamCell`	`CCellDlg`	Идентификатор клетки решётки

Помимо перечисленных в таблице наследников, от класса `CDlg` происходит класс `CMetricsDlg`, служащий для запроса координат клеток решётки в виде, сообразном текущей метрике. В этом классе добавляется конструктор и поля для хранения идентификатора выбранной клетки и союза компонентов. Существуют дочерние классы этого класса для декартовых (класс `CCartesianMDlg`) и обобщённых (класс `CGeneralizedMDlg`) координат.

3.3. Пример: игра Жизнь в среде CAME&L

К среде CAME&L прилагаются четыре различных реализации игры Жизнь [6-8]. Перечислим их:

1. Реализация «в лоб». На каждой итерации перебираются все клетки решётки. Данный компонент работает с декартовыми координатами.
2. Реализация, оптимизированная для работы на многопроцессорной системе. На каждой итерации создаётся множество потоков, каждый из которых работает с определённой частью решётки. Компонент поддерживает декартовы координаты.
3. Реализация, исключающая из рассмотрения клетки, изменение состояния в которых невозможно. Оптимизация осуществлена с использованием класса `OptCtrs`. Данный компонент работает с декартовыми координатами.
4. Реализация «в лоб» для обобщённых координат [17, 18]. На каждой итерации перебираются все клетки решётки.

На рис. 11–16 приведены окна среды во время эволюции планерного ружья.

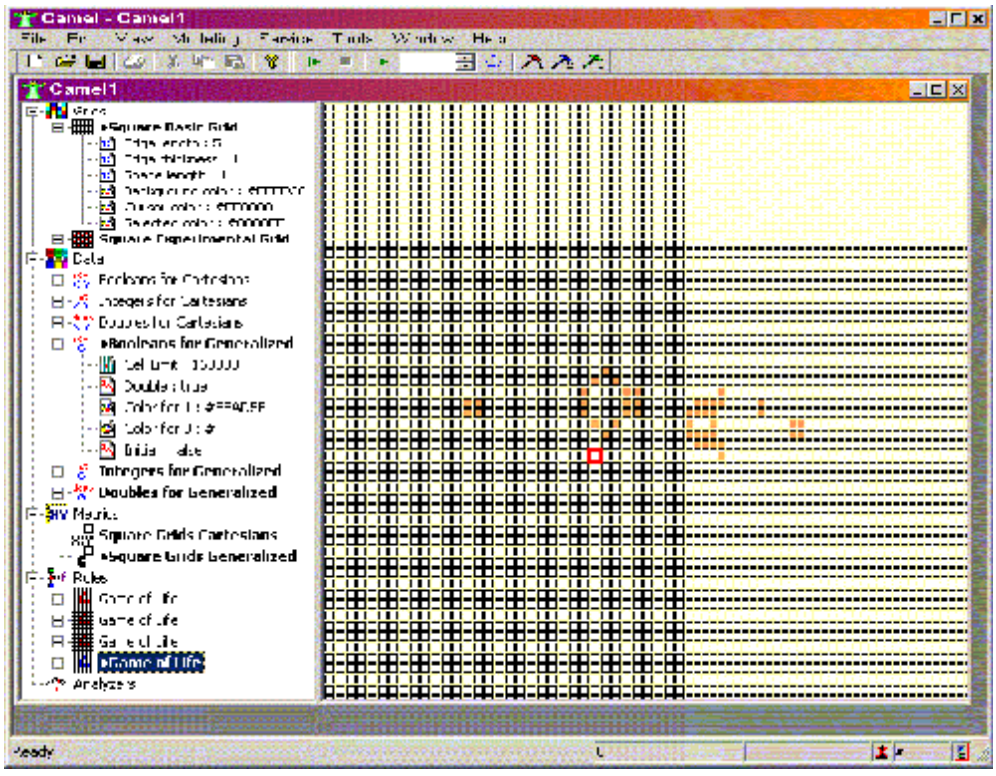

Рис 11. Планерное ружьё на нулевом шаге

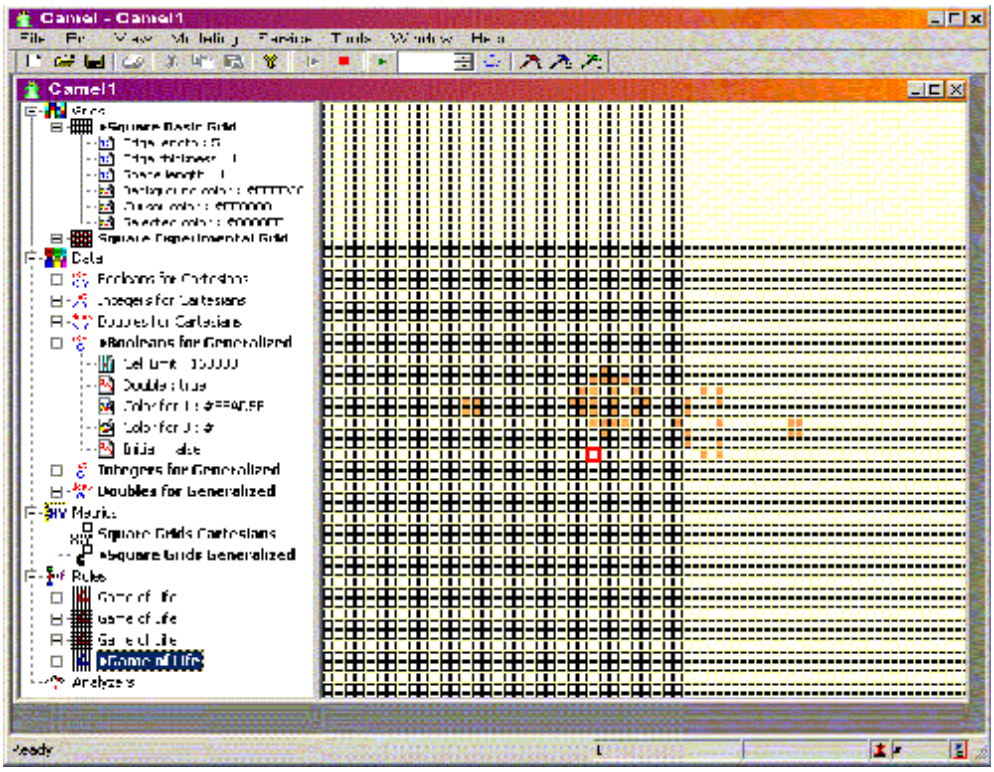

Рис 12. Планерное ружьё на первом шаге

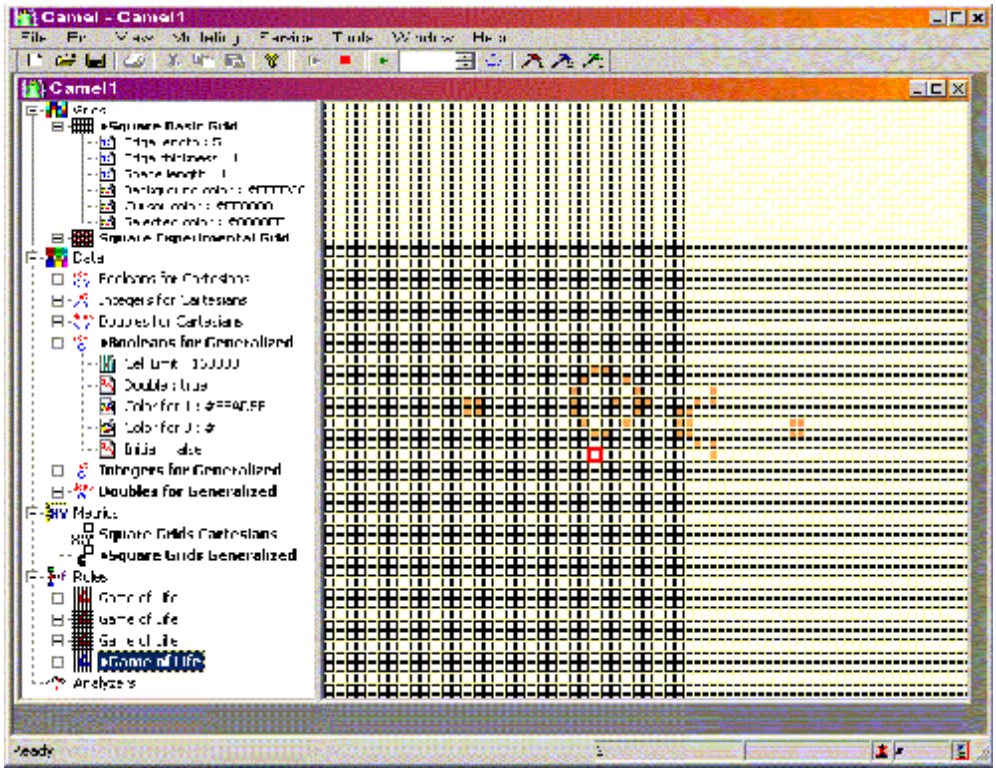

Рис 13. Планерное ружьё на втором шаге

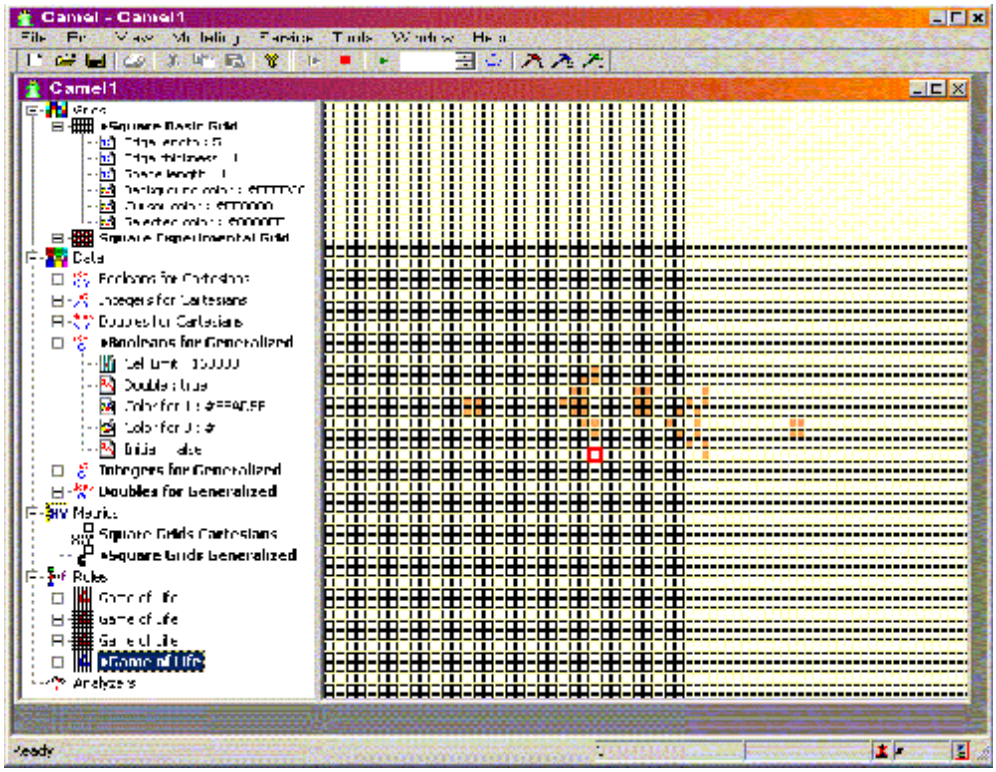

Рис 14. Планерное ружьё на третьем шаге

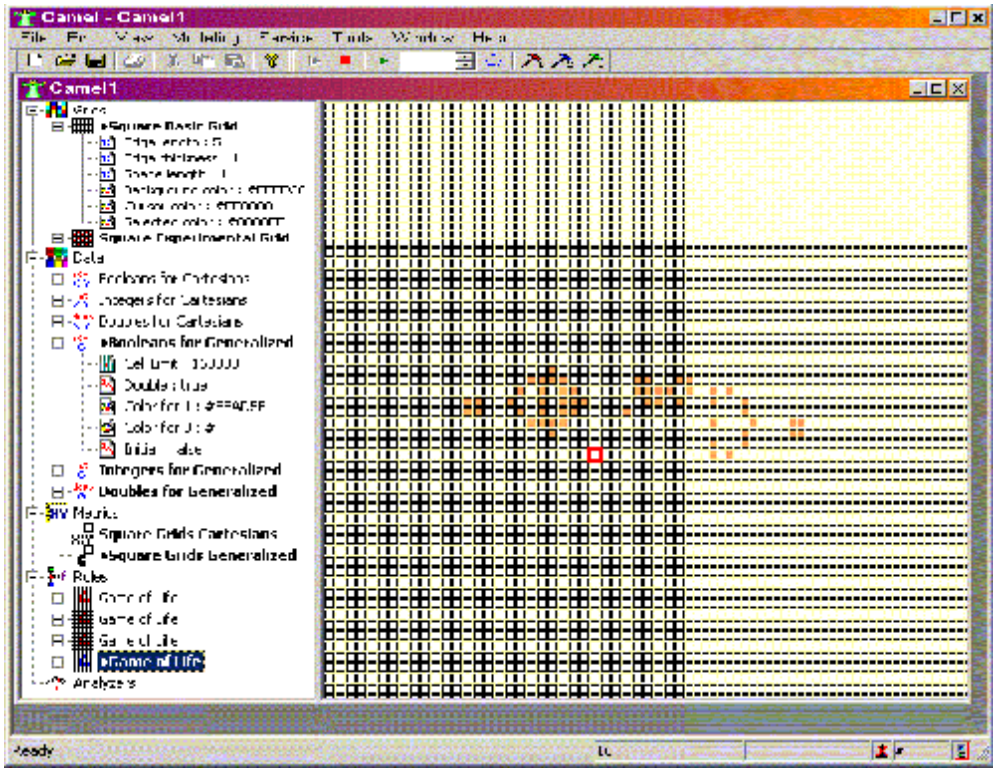

Рис 15. Планерное ружьё на шестнадцатом шаге

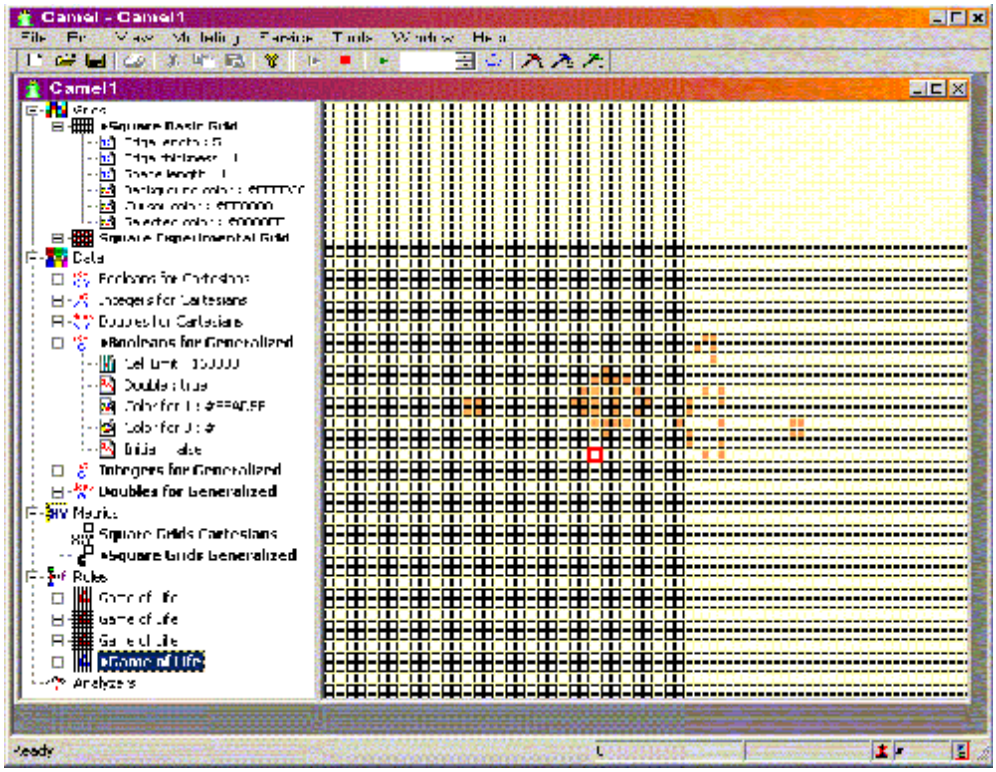

Рис 16. Планерное ружьё на тридцать втором шаге

Было произведено сравнительное тестирование производительности каждого из описанных выше компонентов на различных вычислительных машинах. Приняв за единицу результаты показанные первым вариантом реализации игры Жизнь, статистику производительности компонентов можно описать так:

Таблица 2. Относительная производительность компонентов, реализующих игру Жизнь

Вариант компонента	Pentium 233/128 Mb	Celeron 333/256 Mb	Athlon 1300/256 Mb
1	1.0	1.0	1.0
2	1.1	1.05	1.05
3	0.15	0.1	0.1
4	1.4	1.3	1.25

Так как все тестирования производились на однопроцессорных вычислительных системах, то второй вариант, оптимизированный под многопроцессорные системы, проигрывает первому варианту.

Напротив, оптимизация, исключающая бесполезные вычисления (третья версия компонентов), не смотря на то, что она имеет некие накладные расходы, несомненно выигрывает.

Введение обобщённых координат негативно сказывается на производительности. Однако они имеют ряд преимуществ [17, 18], которые могут оказаться чрезвычайно полезными. Необходимо отметить, что проигрыш в производительности (например, на Athlon'e) не так уж и велик.

ЗАКЛЮЧЕНИЕ

Задачи, для решения которых чрезвычайно полезными оказываются клеточные автоматы, возникают в самых различных отраслях науки. Таким образом, можно с уверенностью утверждать, что существует потребность в программном обеспечении, которое позволило бы удобно и эффективно решать такие задачи.

Каждый из существующих на данный момент пакетов программного обеспечения для решения задач с использованием клеточных автоматов накладывает те или иные ограничения на тип и возможности используемых автоматов. Среда CAME&L – первая попытка создать универсальный инструмент, расширяемый в любом направлении, пригодный для произвольных задач. Библиотека CADLib позволяет описать систему моделирования вплоть до малейших деталей и тонкостей выполнения итераций. Может сложиться впечатление, что для создания компонентов от пользователя требуются глубокие знания особенностей среды, прилагаемой библиотеки и вообще, серьёзных программистских навыков. На самом деле, это не совсем так. Конечно, базовые знания языка C++ [12] необходимы, но лишь базовые! В библиотеке CADLib содержится множество вспомогательных функций и макроопределений, делающих разработке чрезвычайно быстро и простой.

Большинство существующих пакетов имеют неудобный сложный интерфейс, ориентированный на специалистов. Как правило, среда требует знания терминологии, связанной с конкретным классом задач, решаемых с помощью клеточных автоматов. Среда CAME&L предоставляет пользователю интуитивно ясный интерфейс с богатыми возможностями для исследователей. Необходимо отметить, что понятия «пользователь» и «исследователь» становятся синонимичными и определяют человека, использующего описываемую среду для решения задач.

Описываемая среда имеет большое образовательное значение. Она является удобным средством визуализации параллельных алгоритмов. Большинство последовательных алгоритмов иллюстрируются примерами на моделях последовательных вычислений (конечных автоматах, машинах Тьюринга и т.п.). Однако для параллельных вычислений не существует столь ясной, наглядной и универсальной модели, как клеточные автоматы.

Проект CAME&L был анонсирован в работе [18] и на Международной Конференции по Вычислительным Наукам (International Conference on Computational Science), проходившей в Санкт-Петербурге в июне 2003 года.

Предметный указатель

- М -

- О -

- П -

- Р -

- С -

- У -

- Ф -

Литература

[1] *Ulam S.* Random Processes and Transformations // Procedings Int. Congr. Mathem. – 1952. – №2. – pp. 264-275;

[2] *Фон Нейман Дж.* Теория самовоспроизводящихся автоматов: Пер. с англ. – М.: Мир, 1971;

В оригинале: *von Neumann J.* Theory of Self-Reproducing Automata: Edited and completed by A. Burks. – University of Illinois Press, 1966;

[3] *Zuse K.* Calculating Space: Translated from German. – Tech. Transl. AZT-70-64-GEMIT. – MIT Project MAC, 1970;

В оригинале: *Zuse K.* Rechnender Raum. – Vieweg, Braunschweig, 1969;

[4] *Hedlung G. A.* Endomorphism and Automorphism of the Shift Dynamic System // Math. Syst. Theory. – 1969. – №3. – pp. 51-59;

[5] *Holland J.* Universal Spaces: A Basis for Studies in Adaptation // Automata Theory. – Academic Press. – 1966. – pp. 218-230;

[6] *Gardner M.* The Fantastic Combinations of John Conway's New Solitaire Game "Life" // Scientific American. – 1970. – №223. – pp. 120-123;

[7] *Гарднер М.* Математические досуги: Пер. с англ. Ю. А. Данилова / Под ред. Я. А. Смородинского. – М.: Мир, 1972;

[8] *Гарднер М.* Крестики-нолики: Пер. с англ. – М.: Мир, 1988;

[9] *Тоффоли Т., Марголус Н.* Машины клеточных автоматов. Пер. с англ. – М.: Мир, 1991;

[10] *Wolfram S.* A New Kind of Science. – Wolfram Media Inc. – 2002;

[11] *Корн Г., Корн Т.* Справочник по математике (для научных работников и инженеров). – 4-е изд. – М.: Наука, 1978;

[12] *Страуструп Б.* Язык программирования C++. – 3-е изд. – СПб: «Невский диалект», 1999;

[13] *Berlekamp E., Conway J. Guy R.* What Is Life? In: Winning Ways, v. 2. – Academic Press. – 1982;

[14] *Margolus N.* CAM-8: a computer architecture based on cellular automata. In: Pattern Formation and Lattice-Gas Automata. – Addison-Wesley. – 1996;

[15] *Toffoli T.* Cellular Automata Mechanics // Tech. Rep. 208. – Comp. Comm. Sci. Dept., The Univ. of Michigan. – 1977;

[16] *Sarkar P.* A Brief History of Cellular Automata // ACM Computing Surveys. – 2000. – Vol. 32, №1. – pp. 80-107;

[17] *Наумов Л.* Введение и использование обобщённых координат в клеточных автоматах с квадратной, гексагональной и треугольной решётками. – готовится к публикации;

[18] *Naumov L.* Computational Science – ICCS 2003. Part 2 // Generalized Coordinates for Cellular Automata Grids. – Springer-Verlag. – 2003;

[19] *Брауэр В.* Введение в теорию конечных автоматов. – М.: Радио и связь, 1987;

[20] *Спэйнауэр С., Экштейн Р.* Справочник вебмастера. – СПб.: Символ-Плюс, 2001;

[21] *Мешков А., Тихомиров Ю.* Visual C++ и MFC. – 2-е изд. – СПб.: БХВ–Санкт-Петербург, 2000;