Оглавление

1. Введение

1.1. Описание клеточных автоматов

Клеточный автомат — это дискретная модель, которая изучается в теории вычислимости, математике, физике, микромеханике, теоретической биологии и некоторых других науках. Клеточный автомат состоит из регулярной решётки ячеек (или клеток), причем каждая из этих ячеек находится в одном из конечного множества состояний (как правило, в тексте состояния обозначаются разными числами или названиями, в графическом представлении отображаются клетками разных цветов). Решетка может быть любой размерности, может быть плоской, тороидальной и т.п. Для каждой клетки определено некоторое множество ячеек – окрестность клетки. Например, существуют окрестности Мура (совокупность восьми клеток, которые имеют общую вершину с данной клеткой) и Фон Неймана (совокупность четырёх клеток на плоскости, имеющих общую сторону с данной клеткой), но также могут существовать и иные виды окрестностей. Для моделирования клеточного автомата необходимо задание начального состояния всех ячеек, а также правил перехода ячеек из одного состояния в другое. На каждом шаге (время в клеточных автоматах дискретное) определяется новое состояние для каждой клетки исходя из правил перехода и состояния соседних ячеек (т.е. окрестности). В большинстве автоматов правила перехода одинаковы для всех клеток и одновременно применяются ко всей решётке.

1.2. История клеточных автоматов

Еще в 1930-х годах Станислав Улам, работавший в то время в Лос-Аламосской лаборатории, используя простую решёточную модель, изучал рост кристаллов. В то же время один из коллег Улама Джон фон Нейман работал над проблемой самовоспроизводящихся систем. Изначально фон Нейман описывал концепцию робота, который собирает другого робота. Это так называемая кинематическая модель. Однако, после разработки подобной модели фон Нейман понял, что возникают множественные сложности создания такого самовоспроизводящегося

робота (в частности, необходимо наличие «запаса частей», из которых и должен строиться сам робот). В то же время Станислав Улам предложил фон Нейману использовать более абстрактную математическую модель, подобную той, которую сипользовал сам Улам при изучении процесса роста кристаллов. Так возникла первая клеточно-автоматная система. Аналогично решётке Улама, клеточный автомат фон Неймана является двухмерным, а самовоспроизводящийся робот описан с помощью алгоритма. В результате был получен универсальный конструктор –клеточный автомат, имеющий окрестность из непосредсвтенно прилегающих ячеек и 29 состояний. Джоном фон Нейманом было доказано, что для этой модели существует образец, который может бесконечно копировать самого себя.

Позднее, в 1940-е годы, Норбертом Винером и Артуро Розенблютом была разработана модель возбудимой среды в виде клеточного автомата. Целью создания этой модели было математическое описание процесса распространения импульса в сердечных нервных узлах.

В 1960-е годы клеточные автоматы изучались как частный тип динамических систем, также впервые была установлена связь КА с областью символьной динамики.

В 1970 году Джоном Конвеем были открыт клеточный автомат, который по праву считается в настоящее время самым известным, – игра «Жизнь». Этот клеточный автомат был в свое время популяризирован Мартином Гарднером. Его правила довольно просты: существует два вида клеток (живые и мертвые), если у клетки ровно два живых соседа, то ее состояние не изменится. Если клетка имеет трёх живых соседей, она "оживает". В остальных случаях клетка умирает (от перенаселения или от одиночества). Несмотря на подобную простоту правил, "Жизнь" проявляет колоссальное разнообразие поведения: от очевидного хаоса до порядка повторяющихся и стабильных структур. Одним из феноменов игры «Жизнь» являются глайдеры (хотя они присутствуют и во многих других клеточных автоматах) — такие сочетания клеток, которые движутся по сетке как единое целое (глайдеры принадлежат к виду образцов, называемому "космическими кораблями").

Существует возможность построения автомата, в котором глайдеры смогут выполнять вычисления, и впоследствии в игре "Жизнь" была реализована машина Тьюринга с использованием глайдеров.

В 1969 году немецкий инженер Конрад Цузе опубликовал знаковую книгу «Вычислимый космос» (другой вариант перевода - "Вычислительное пространство"), в которой выдвинул предположение, что физические законы по своей природе являются дискретными, а вся Вселенная –это по сути клеточный автомат или компьютер. Это была первая книга, которая положила начало новому направлению физике – цифровой физике.

В 1983 Стивен Вольфрам (являющийся автором системы компьютерной алгебры Mathematica) опубликовал статью, в которой исследовал простой, но до тех пор неизвестный класс клеточных автоматов, называемых элементарными (их основное отличие от остальных автоматов состоит в том, что решетка в них не плоская двумерная, а одномерная). Неожиданная сложность поведения автоматов с очень простыми правилами заставила Вольфрама сделать предположение, что сложность естественных систем может быть обусловлена сходным механизмом.

В 1987 году Брайан Сильверман предложил ставший впоследствии популярным клеточный автомат Wireworld. Позже в этом клеточном автомате была создана модель компьютера, полного по Тьюрингу.

В 2002 году Стивен Вольфрам опубликовал книгу «Новый тип науки» (A New Kind of Science), в которой он утверждает, что мир является дискретным, а методы, применимые при изучении клеточных автоматов, имеют большое значение для всех областей науки.

В нашей стране клеточные автоматы также не остались без внимания. В частности, широко изучались их теоретические положения и свойства, поведение клеточных автоматов (процессы, происходящие в них), проблемы вычислений в клеточных автоматах (вычисление булевых и алгебраических операторов). Публиковалась различная литература, посвященная клеточным автоматам, например в 1990 вышла книга В.Б. Кудрявцева, А.С. Подколзина и А.А. Болотова "Основы

теории однородных структур" (в некоторых публикациях клеточные автоматы именовались однородными структурами).

1.3.Применение клеточных автоматов

Однако, оказалось, что клеточные автоматы можно широко применять в различных практических областях деятельности.

В частности, опять же Стивен Вольфрам еще в 1985 предложил использовать правило 30 (один из самых известных элементарных клеточных автоматов) в качестве генератора псевдослучайных чисел и шифратора последовательностей в криптографии.

Также клеточные автоматы могут моделировать поведение экосистемы, состоящей из травоядных и хищников.

Несмотря на кажущуюся простоту автоматы описывают и эволюцию самих клеток (то есть разновидность простейшего генома).

В игре "Жизнь" реализована машина Тьюринга, а в мире проводов wireworld создан универсальный компьютер, то есть существует возможность писать программы непосредственно на самих клеточных автоматах.

Довольно простые правила моделируют движение толпы, что может использоваться при проектировании общественных зданий и проведении массовых мероприятий.

В Сибирской государственной автомобильно-дорожной академии (СибАДИ) в 2012 году была создана модель автотранспортного потока города Омска.

С помощью клеточных автоматов возможно прогнозирование урожайности сельскохозяйственной культуры.

Также клеточными автоматами могут моделироваться многие химические процессы:

На заре возникновения теории КА Станислав Улам использовал простую решеточную модель (прообраз КА) при изучении процесса роста кристаллов.

Правило "HPP" моделирует хаотическое движение молекул газа. Другие похожие правила моделируют поведение сыпучих материалов и вязкой жидкости.

В клеточных автоматах реализованы такие процессы как диффузия и растворение твердых тел, а также реакция Белоусова — Жаботинского.

1.4.Хэш-алгоритм

В 1984 году Билл Госпер опубликовал статью "Exploiting regularities in large cellular spaces" (Эксплуатация закономерностей в больших клеточных пространствах), в которой предложил новый алгоритм для расчета эволюции клеточных автоматов – алгоритм hashlife. Этот алгоритм позволяет существенно уменьшить время расчета повторяющихся структур в клеточных автоматах.

1.5.Постановка задачи

Целью данной дипломной работы является разработка модуля хэш-алгоритма для системы моделирования клеточных автоматов Xlife.

Задачами работы в соответствии с целью являются:

– разработка модуля для системы моделирования клеточных автоматов Xlife, обеспечивающего поддержку алгоритма hashlife;

– портирование программного обеспечения в операционную систему Windows;

– сравнение скорости хэш-алгоритма с плиточным алгоритмом.

2. Основная часть

2.1. Портирование в операционную систему MS Windows

Задача портирования системы xlife в операционную систему Microsoft Windows может быть решена следующими путями:

- портирование графической системы X Window System в MS Windows;
- полный перевод программы xlife в одну из существующих кроссплатформенных библиотек (Qt, GTK+, wxWidgets, fltk и т.д.);
- запуск xlife в UNIX-подобной среде Cygwin в операционной системе MS Windows;
- модификация уже существующей обертки библиотеки (wrapper) X11.

Рассмотрим преимущества и недостатки каждого из вариантов.

1.1.1. Портирование графической системы X Window System

X Window System — оконная система, которая обеспечивает стандартные протоколы и инструменты для построения графического интерфейса пользователя.

X Window System предоставляет следующие базовые графические функции: отрисовку и перемещение окон и графических объектов на них на экране, взаимодействие с устройствами ввода информации (мышь, клавиатура, мультитач и т.д.). Система X Window не определяет деталей интерфейса — это задача оконных менеджеров, которых существует довольно большое количество. Поэтому внешний вид программ может существенно отличаться в зависимости от настроек конкретного оконного менеджера.

В X Window System существует возможность технология т.н. сетевой прозрачности: графические приложения выполняются на одной машине в сети, а их интерфейс при этом передается по сети и отображается на другом устройстве (если это разрешено настройками и политикой безопасности). В контексте системы X Window термины «клиент» и «сервер» имеют непривычное значение: сервер – это дисплей пользователя (дисплейный сервер), а клиент — программа, использующая

этот дисплей (может выполняться как на локальном, так и на удалённом компьютере).

Система X Window System была разработана в 1984 году в Массачусетском технологическом институте (MIT) – одном из признанных лидеров высоких технологий. Текущая версия протокола X11 появилась в сентябре 1987 года, а текущая дата выпуска – 6 июня 2012 года (версия X11R7.7). Референсная реализация системы свободно доступна на условиях лицензии MIT. В настоящее время X Window является основной оконной системой для UNIX-подобных операционных систем.

Полная реализация X Window System в среде MS Windows могла бы быть полезной также и для других проектов, однако библиотека системы X Window System насчитывает большое количество низкоуровневых (работающих с мышью, клавиатурой, а также выполняющих отрисовку объектов) функций (порядка нескольких сотен). Их реализация стала бы очень трудозатратным процессом: необходимо написать большое количество кода библиотечного уровня с использованием функций WinAPI, поэтому данный вариант был отвергнут.

1.1.2. Перевод программы xlife в одну из кроссплатформенных библиотек.

В настоящее время самыми используемыми кроссплатформенными библиотеками элементов графического интерфейса пользователя являются Qt, GTK+, wxWidgets, fltk.

Qt – кроссплатформенный инструментарий разработки программного обеспечения на C++. Также существуют библиотеки, позволяющие использовать функционал Qt в других языках программирования: Python; Ruby; Java; PHP и др.

Позволяет запускать написанное с его помощью программное обеспечение во многих современных операционных системах путём компиляции программы для каждой ОС без изменения исходного кода. Включает в себя основные классы, требуемые при разработке прикладного программного обеспечения: элементы графического интерфейса, средства для работы с мультимедиа, классы для работы с

сетью, базами данных, XML и др. Qt является объектно-ориентированным, легко расширяемым, поддерживает парадигму компонентного программирования.

Имеются версии библиотеки для Microsoft Windows, UNIX-подобных систем (с графической подсистемой X11), Android, iOS, Mac OS X, Microsoft Windows CE, QNX, встраиваемых Linux-систем и платформы S60. На сегодняшний день осуществляется портирование на Windows Phone, Windows RT, а также Haiku и Tizen.

GTK+ (сокращение от GIMP ToolKit) — кроссплатформенная библиотека элементов интерфейса, имеющая простой интерфейс программирования приложений (API). В настоящее время является одной из двух наиболее используемых библиотек для X Window System наряду с Qt.

Сначала GTK+ была частью популярного графического редактора GIMP, однако позже она развилась в отдельный. Сейчас GTK+ – это официальная библиотека для создания GUI в проекте GNU.

GTK+ написана на языке программирования C, но, несмотря на это, является объектно-ориентированной. Выбор C в качестве рабочего языка обусловлен желанием иметь возможность создавать интерфейсы для других языков программирования без значительных усилий. В итоге с помощью GTK+ можно писать программы на более чем 20 языках программирования, включая C++, языки платформы .Net, D, Fortran, Java, Python, Ruby и многих других. GTK используется в UNIX-подобных ОС, а также Microsoft Windows и Mac OS X.

FLTK (Forms Library Toolkit) — кроссплатформенная библиотека инструментов с открытым исходным кодом (распространяется по лицензии LGPL), предназначенная для построения графического интерфейса пользователя (также может применяться для создания 3D-приложений на базе OpenGL). FLTK – это библиотека виджетов, которая работает на ОС UNIX/Linux X11, Microsoft Windows и MacOS X. Малый объём библиотеки позволяет легко использовать ее во встраиваемых системах. FLTK написана на C++ и является объектно-ориентированной. Существует возможность ее использования в языках программирования Perl, Python, Lua и Ruby.

wxWidgets (ранее называлась wxWindows) — это кроссплатформенная библиотека инструментов с открытым исходным кодом, предназначенная для разработки кроссплатформенных (на уровне исходного кода) приложений, в частности для создания графического интерфейса пользователя.

wxWidgets может применяться не только для построения GUI. Она также имеет возможность работы с сетями, графическими изображениями, HTML, XML документами, файловыми системами, процессами, архивами, системами печати, мультимедиа, имеет классы для организации многопоточности, отладки, отправки дампов и др. wxWidgets выпущена под лицензией LGPL.

wxWidgets позволяет компилировать программы в системах Microsoft Windows, Apple Macintosh, UNIX-подобных ОС (для X11, Motif и GTK+), OpenVMS и OS/2. Встраиваемая версия сейчас находится в разработке.

Библиотека написана на C++, но имеет возможность использования в языках Ruby, Python, Smalltalk, Perl, Erlang, Haskell.

В целом, перевод программных кодов xlife в одну из существующих кроссплатформенных библиотек элементов графического интерфейса пользователя мог бы существенно помочь в продвижении проекта xlife, поскольку позволил бы создавать версии программы для различных операционных систем без изменения исходных кодов. Однако данный подход повлек бы за собой и многие существенные трудности, основной из которых стало бы преобразование всего исходного кода xlife, особенно в части графического отображения и обработки событий. Этот подход также был отвергнут в связи с большими трудозатратами.

1.1.3. Запуск xlife в среде Cygwin в операционной системе MS Windows.

Cygwin — это UNIX-подобная среда, а также интерфейс командной строки, предназначенные для операционной системы Microsoft Windows. Назначение Cygwin – обеспечить тесное взаимодействие Windows приложений, ресурсов и данных с приложениями, данными и ресурсами UNIX-подобной среды. Существует возможность запускать из среды Cygwin обычные Windows программы, а также возможно использовать инструменты Cygwin из Microsoft Windows.

Cygwin состоит из двух частей: динамически подключаемая библиотека (DLL) cygwin1.dll, обеспечивающая совместимость на уровне API и реализующая значительную часть стандарта POSIX, и большая коллекция приложений, обеспечивающих среду UNIX, включая Unix shell.

Cygwin является инструментом для портирования программного обеспечения, написанного для UNIX в Windows, и представляет собой библиотеку, реализующую интерфейс прикладного программирования POSIX на основе системных вызовов Win32 (WinAPI). Кроме того, Cygwin включает в себя инструменты разработки GNU для написания ПО, а также коллекцию прикладных программ, которые эквивалентны базовым программам UNIX. В 2001 году в Cygwin был также добавлен пакет графической системы X Window System.

Cygwin содержит библиотеку MinGW, позволяющую работать с Windows API; MinGW менее требовательна к объёмам как оперативной, так и дисковой памяти, распространяется под свободной лицензией и имеет возможность работать с любым программным обеспечением, однако многие функциональные возможности спецификации POSIX реализованы в ней менее полно, чем в Cygwin.

Запуск программы xlife в среде Cygwin имеет многие существенные преимущества: не нужно вносить изменения в исходный код программы, реализовывать функции графической системы X Window System, что позволит существенно снизить время разработки. Однако, у данного подхода существует также и значительный недостаток: для запуска программы xlife необходимо установить среду Cygwin в ОС MS Windows, что может вызвать неудобства для большого количества пользователей. В связи с этим данный метод решения был признан «запасным вариантом».

1.1.4. Модификация существующей обертки библиотеки (wrapper) X11.

Обёртка библиотеки (с англ. wrapper – обертка) — это промежуточная прослойка между программой пользователя и другой библиотекой (коллекцией функций или классов) или интерфейсом программирования приложений (API).

Библиотеки обычно предоставляют ряд интерфейсов или «публичных» функций, а также классов, которые могут прямо использоваться прикладной программой. Целью написания обёртки может быть предоставление возможностей библиотеки для какого-либо другого языка программирования, в котором прямой невозможен либо существенно затруднен вызов функций этой библиотеки. Также обертки могут применяться, чтобы обеспечить более удобный интерфейс (объектно-ориентированный стиль, улучшенный дизайн и т.п.), для упрощения работы с несовместимыми форматами данных либо для обеспечения кроссплатформенности (в таком случае библиотечные функции скрывают вызов системных функций на разных платформах, предоставляя единый интерфейс).

В ранних версиях программы xlife существовал порт в операционную систему MS Windows – написанная Дэвидом Киндером (David Kinder) обертка для библиотеки оконной системы X Window System. Эта обертка реализует только небольшую часть функций X Window (только те из них, которые используются в xlife). На протяжении нескольких лет данная обертка не обновлялась, несмотря на то, что в xlife произошли существенные изменения, в связи с чем библиотека Дэвида Киндера фактически потеряла актуальность и не могла больше использоваться с целью портирования программы в MS Windows. Подход модификации уже существующего кода обертки имеет ряд преимуществ: необходимо реализовать лишь очень маленькую часть функций X Window, что снижает временные и трудовые затраты в несколько раз, также написанный код уже использовался в программе xlife ранее, а, следовательно, был многократно протестирован при использовании. Основным недостатком является более низкая скорость работы графической части обертки по сравнению с библиотекой X Window System.

1.1.5. Выбор решения

После анализа всех вариантов было принято решение использовать существующую обертку Дэвида Киндера с внесением в нее всех необходимых изменений. Данное решение является наиболее перспективным исходя из сочетания трудозатрат, эффективности результата и приемлемости для пользователей. Главным

достоинством этого варианта является наличие уже существующего программного кода, реализующего большую часть необходимых функций, а также отсутствие необходимости установки дополнительного программного обеспечения (как в случае с Cygwin).

1.1.6. Реализация решения

Обертка библиотеки X11 Дэвида Киндера представляет собой написанный на языке программирования C++ набор классов и функций, реализующих основные графические функции (отрисовку, перемещение и изменение окон и простых графических объектов), функции ввода информации (нажатие клавиш, действия с мышью) и некоторые системные функции UNIX (возврат текущего времени в UNIX-формате, получение списка файлов в директории и т.п.).

Обертка предоставляет программисту-пользователю интерфейс, полностью аналогичный интерфейсу библиотеки X Window System, позволяя создавать графические оконные приложения, использующие функции X11, но предназначенные для использования в операционной системе Microsoft Windows.

Однако, в связи с тем, что на протяжении долгого времени, несмотря на продолжающиеся обновления программы xlife, фактически не велось работы по усовершенствованию библиотеки Дэвида Киндера, данная оболочка устарела. В частности в ней не были реализованы некоторые функции, использующиеся в более поздних версиях xlife (например, такие функции как XChangeGC, XSetWindowBackground, XSetWMName, XDrawRectangles), также не было поддержки колесика мыши.

2.1.1. Функция XChangeGC

Синтаксис функции:

```
XChangeGC (display, gc, valuemask, values)
    Display *display;
    GC gc;
    unsigned long valuemask;
    XGCValues *values;
```

Аргументы:

Таблица 1. Аргументы функции XChangeGC

display	Определяет соединение с X-сервером (дисплеем).
gc	Указывает на изменяемый графический контекст GC.
valuemask	Указывает, какие компоненты графического контекста изменяются, используя информацию из структуры values. Этот аргумент определяется с помощью побитового исключающего или нескольких масок.
values	Определяет новые свойства графического контекста в соответствии с маской.

Описание функции:

Функция XChangeGC() изменяет параметры графического контекста gc в соответствии с параметрами values (передаются новые значения параметров) и маской valuesmask (указано, какие именно параметры необходимо изменить).

Исходный код функции:

```
int XChangeGC(Display* display, GC gc, unsigned long valuemask, XGCValues* values)
{
	unsigned long mask = GCForeground|GCBackground|GCFunction|GCPlaneMask;
	if ((valuemask & ~(mask)) != 0)
		::OutputDebugString("X11Lib: XCreateGC() valuemask includes unsupported values\n");

	if (valuemask & GCForeground)
		((X11Wnd::Context*)gc)->setForegroundColour(values->foreground);
	if (valuemask & GCBackground)
		((X11Wnd::Context*)gc)->setBackgroundColour(values->background);
	if (valuemask & GCFunction)
	{
		switch (values->function)
		{
		case GXcopy:
			((X11Wnd::Context*)gc)->setDrawMode(R2_COPYPEN);
			break;
		case GXinvert:
			((X11Wnd::Context*)gc)->setDrawMode(R2_NOT);
			break;
		case GXxor:
			((X11Wnd::Context*)gc)->setDrawMode(R2_XORPEN);
			break;
		default:
			::OutputDebugString("X11Lib: XCreateGC() GCFunction not recognized\n");
			break;
		}
	}
	if (valuemask & GCPlaneMask)
		::OutputDebugString("X11Lib: XCreateGC() GCPlaneMask ignored\n");
```

```
    return 0;
}
```

2.1.2. Функция XSetWindowBackground

Синтаксис функции:

XSetWindowBackground (display, w, background_pixel)

Display *display;

Window w;

unsigned long background_pixel;

Аргументы функции:

Таблица 2. Аргументы функции XSetWindowBackground

display	Указатель на соединение с X-сервером.
w	Указывает изменяемое окно.
background_pixel	Определяет пиксель, используемый для заднего плана.

Описание функции:

Функция XSetWindowBackground() устанавливает цвет фона окна, используя значение background_pixel. Изменение цвета фона не приводит к перерисовке. Нельзя использовать установку цвета фона для окон InputOnly, иначе появится ошибка BadMatch..

Исходный код функции:

```
int XSetWindowBackground(Display*, Window wnd, unsigned long background)
{
    ((X11Wnd*)wnd)->setBackgroundColour(background);
    return 0;
}
```

2.1.3. Функция XSetWMName

Синтаксис функции:

void XSetWMName (display, w, text_prop)

Display *display;

Window w;

XTextProperty *text_prop;

Аргументы функции:

Таблица 3. Аргументы функции XSetWMName

display	Указатель на соединение с X-сервером (например, дисплей).
w	Определяет окно.
text_pro p	Указывает используемую структуру XTextProperty, содержащую новый заголовок окна.

Описание функции:

Функция XSetWMName() вызывает функцию XSetTextProperty(), чтобы установить свойство WM_NAME (заголовок окна).

Исходный код функции:

```
void XSetWMName(Display* display, Window w, XTextProperty* window_name)
{
        ((X11Wnd*)w)->setWindowTitle((char*)(window_name->value));
}
```

2.1.4. Функция XDrawRectangles

Синтаксис функции:

XDrawRectangles (display, d, gc, rectangles, nrectangles)

Display *display;

Drawable d;

GC gc;

XRectangle rectangles[];

int nrectangles;

Аргументы функции:

Таблица 4. Аргументы функции XDrawRectangles

display	Указывает на соединение с X-сервером (дисплеем).
d	Определяет окно или другой т.н. графический образ
gc	Указывает графический контекст GC.
rectangles	Представляет собой массив прямоугольников.
nrectangles	Определяет количество прямоугольников в массиве.

Описание функции:

Функция XDrawRectangles() рисует контуры определенных в списке rectangles прямоугольников как пятиточечную полилинию, используя координаты углов прямоугольников ([x,y], [x+width,y], [x+width,y+height], [x,y+height], [x,y]).

Функция отрисовывает каждый пиксель только однократно. XDrawRectangles() рисует прямоугольники в том порядке, в котором они перечислены в массиве rectangles. Если прямоугольники пересекаются, то точки, находящиеся в местах пересечений, отрисовываются повторно.

Исходный код функции:

```
int XDrawRectangles(Display* display, Drawable d, GC gc, XRectangle* rectangles, int nrectangles)
{
	for (int i = 0; i < nrectangles; i++)
	{
		r[i].left = rectangles[i].x;
		r[i].top = rectangles[i].y;
		r[i].right = rectangles[i].x + rectangles[i].width + 1;
		r[i].bottom = rectangles[i].y + rectangles[i].height + 1;
	}

	((X11Wnd*)d)->drawRects((X11Wnd::Context*)gc,r,nrectangles);

	return 0;
}
```

2.1.5. Поддержка колесика мыши

Основная сложность реализации поддержки колесика мыши заключалась в различии подходов к проблеме в интерфейсе программирования приложений WinAPI и в библиотеке X Window System.

В основе обработки действий пользователя и наступления определенных условий как в Microsoft Windows, так и в UNIX-подобных и многих других современных операционных системах лежит концепция перехвата и обработки событий. Событие – это сообщение, которое возникает в определенных точках программного кода при выполнении различных условий.

Для решения задачи обычно создаются специальные функции (методы) или объекты – обработчики событий: при условии возникновения события посылается сообщение, а обработчик перехватывает это сообщение и определенным образом реагирует на него (например, вызывает другие функции, обрабатывает полученные данные, передает информацию другой программе/машине, пишет лог, выводит пользователю сообщения и т.п.). Примерами событий могут быть, в частности, нажатия клавиш, движение мыши, нажатие кнопок мыши, прокрутка колесиком, изменение положения или размера окна, сигнал о завершении или начале некоторой

процедуры, сигнал о передаче данных по сети или от приложения к приложению и многое другое.

Однако в ОС MS Windows и в оконной системе X Window System различается подход к событию, вызванному прокруткой колесика мыши. В WinAPI такое действие посылает приложению сообщение WM_MOUSEWHEEL, а в старшем слове поля сообщения wparam будет находиться число, означающее направление вращения и «прокрученное» расстояние, измеренное в WHEEL_DELTA (обычно равно 120). Если число в старшем слове положительное, это означает, что колесико прокручивали вперед (от пользователя), отрицательно число показывает, что имела место прокрутка назад (к пользователю). Для определения значения этого числа используется функция GET_WHEEL_DELTA_WPARAM(wParam).

В системе X Window System прокрутка колесиком мыши является, по сути, имитацией нажатия кнопки мыши и вызывает событие типа ButtonPress. При этом обычно прокрутка вперед считается нажатием четвертой кнопки мыши, а прокрутка назад – пятой. Таким образом, в реализацию функции XCheckMaskEvent в обертке Дэвида Киндера необходимо внести некоторые изменения.

Синтаксис функции:

bool XCheckMaskEvent (display, event_mask, event_return)

Display *display;

long event_mask;

XEvent *event_return;

Аргументы функции:

Таблица 5. Аргументы функции XCheckMaskEvent

display	Определяет соединение с X-сервером.
event_mask	Указывает маску событий, которые необходимо обработать.
event_retur n	Возвращает событие, совпавшее с маской.

Описание функции:

Функция XCheckMaskEvent() просматривает очередь событий соединения с сервером display и ищет события, подходящие по маске event_mask. Если функция находит совпадения, то она копирует событие в связанную структуру event_return,

удаляет это событие из очереди и возвращает true. Остальные события в очереди не изменяются и не обрабатываются. Если события запрашиваемого типа в очереди не оказалось, то XCheckMaskEvent() вызывает функцию XFlush() для соединения display и возвращает false.

Фрагменты исходного кода функции XCheckMaskEvent() (измененные участки выделены **жирным шрифтом**):

```
bool XCheckMaskEvent (Display* display, long event_mask, XEvent* event_return)
{
	// Make sure any outstanding Windows messages are handled
	X11App::getApp().processMessages();
	X11App::MsgSet msgs; // Create message set
	if (event_mask & KeyPressMask)
		msgs.insert(WM_KEYDOWN);
	if (event_mask & (PointerMotionMask|Button1MotionMask|Button2MotionMask|Button3MotionMask))
		msgs.insert(WM_MOUSEMOVE);
	if (event_mask & StructureNotifyMask)
		msgs.insert(WM_SIZE);
	if (event_mask & ButtonPressMask)
	{
		msgs.insert(WM_LBUTTONDOWN);
		msgs.insert(WM_RBUTTONDOWN);
		msgs.insert(WM_MBUTTONDOWN);
		msgs.insert(WM_MOUSEWHEEL);
	}
	…
	X11App::MsgData msg;
	if (X11App::getApp().getFromMsgQueue(msgs,msg))
	{
		// Initialize the event header
		::ZeroMemory(event_return,sizeof(XEvent));
		event_return->xany.display = display;
		event_return->xany.window = (Window)msg.m_wnd;
		switch (msg.m_msg) // Check the event type
		{
		case WM_KEYDOWN:
			…
			break;
		…
		case WM_MOUSEWHEEL:
			event_return->xany.type = ButtonPress;
			event_return->xbutton.x = LOWORD(msg.m_param2);
			event_return->xbutton.y = HIWORD(msg.m_param2);
			if (GET_WHEEL_DELTA_WPARAM(msg.m_param1) > 0)
				event_return->xbutton.button = Button4;
			else
				event_return->xbutton.button = Button5;
			break;
		…
		case WM_NCCALCSIZE:
			event_return->xany.type = Expose;
			break;
		}
```

```
            return true;
        }
        return false;
}
```

Рассмотрим внесенные изменения более подробно:

Для того, чтобы отслеживать вращение колесика мыши в тех же случаях, что и в X Window System (при наличии в маске значения ButtonPressMask), в исходный код функции была добавлена строка «msgs.insert(WM_MOUSEWHEEL);», которая вставляет сообщение о прокрутке в список отслеживаемых событий.

Для корректной обработки события колесика «в стиле» X Window System были добавлены следующие строки:

```
case WM_MOUSEWHEEL:
        event_return->xany.type = ButtonPress;
        event_return->xbutton.x = LOWORD(msg.m_param2);
        event_return->xbutton.y = HIWORD(msg.m_param2);
        if (GET_WHEEL_DELTA_WPARAM(msg.m_param1) > 0)
                event_return->xbutton.button = Button4;
        else
                event_return->xbutton.button = Button5;
        break;
```

Строка «event_return->xany.type = ButtonPress;» присваивает возвращаемому событию новый тип – нажатие кнопки мыши. Следующие две строки сохраняют координаты указателя мыши в момент действия. Далее происходит проверка: если старшее слово msg.m_param1 – это положительное число, то колесико было прокручено вперед, и необходимо присвоить возвращаемому событию необходимый номер нажатой кнопки (4), если отрицательное – считать, что была нажата пятая кнопка мыши.

2.1.6. Итог портирования xlife в ОС Microsoft Windows

В результате проделанной работы использование событий и сообщений WinAPI для прокрутки колесика мыши было приведено в соответствие с событиями в оконной системе X Window System, также были добавлены функции XChangeGC, XSetWindowBackground, XSetWMName, XDrawRectangles и исправлены некоторые мелкие недочеты и ошибки в предыдущей версии реализации. Таким образом использование обертки библиотеки X11 Дэвида Киндера сделало возможным портирование программы xlife в операционную систему MS Windows практически без изменений исходного кода самой программы.

2.2. Разработка модуля хэш-алгоритма

1.1.1. Описание системы моделирования клеточных автоматов xlife

XLife представляет собой систему моделирования клеточных автоматов с различными правилами, количеством состояний, разными окрестностями (по Муру и по Фон Нейману) и другими параметрами с использованием нескольких алгоритмов генерации.

В XLife допустимы следующие правила клеточных автоматов:

1) Простейшие с двумя состояниями (к ним относятся Игра "Жизнь" Джона Конвея, Seeds и многие другие). Такие правила представлены в виде S0..8/B0..8. Например, правило S123/B45 означает, что живая клетка остается живой, если у нее 1, 2 или 3 соседа, новая рождается, если у нее 4 или 5 соседей.

2) Правила истории для автоматов с двумя состояниями: в этом случае автомат эволюционирует по своим обычным правилам, однако дополнительно происходит "окрашивание" клеток в соответствии с развитием (особыми цветами обозначаются живые/мертвые клетки на месте начальных живых, бывшие живые и особые граничные клетки).

3) Псевдоцвета для автоматов с двумя состояниями: развиваются по своим обычным правилам, но дополнительными цветами показываются новые клетки на каждом шаге.

4) Мир проводов: принадлежит к табличным правилам, реализован отдельно для повышения эффективности.

5) Автоматы с N состояниями (т.н. поколения): эволюционируют подобно простым автоматам с двумя состояниями (в "размножении" участвуют только самые молодые клетки), но каждая клетка умирает не сразу, а по прошествии maxstates поколений. Правила представлены в виде S0..8/B0..8/N, где S и B аналогичны первому пункту, а N - maxstates - максимальное время жизни клетки (количество состояний), максимальное значение N = 256.

6) Особые табличные правила: задаются таблицами из файлов. В таблицах указаны условия (количество и вид соседей, вид соседства - по Фон Нейману или по

Муру, вид симметрии), по которым происходит эволюция клетки. Более подробно данный вид правил описан в разделе "Поддержка N состояний".

7) Дилемма заключенного (задача Ллойда): каждая клетка представляет собой объект (человека), который может либо сотрудничать, либо предавать своих соседей. На каждом шаге каждая клетка "играет" с соседями и получает от этой игры некоторую сумму денег (более подробно написано в разделе "Дилемма заключенного"). Затем в результате оценки прибылей соседей клетка решает как вести себя выгоднее (предавать или сотрудничать) и на основе этой оценки строит свою стратегию действий.

В XLife используются два основных алгоритма:

– плиточный (алгоритм поколений).

– хэш-алгоритм (новый алгоритм, добавленный в ходе данной дипломной работы) .

В первом алгоритме рассматривается все непустые клетки "игрового поля" и некоторая часть пустых, а затем происходит пошаговая эволюция клеточного автомата по текущим правилам.

Второй алгоритм действует по принципиально иным правилам: поле разбивается на квадраты со сторонами N, равными степеням двойки. В таких квадратах можно однозначно определить состояние центральной области размером (N/2)*(N/2) через N/4 шага. Если в некотором словаре будет записан исходный квадрат и результат его эволюции, то при дальнейшей встрече данного квадрата можно будет сразу определить его будущее состояние и не проводить расчет. Используя определенные преобразования (в этом и заключается основная суть алгоритма) можно вычислить результат эволюции квадрата 2N*2N. Более подробно хэш-алгоритм описан в пункте 2.2.2.

Хэш-алгоритм наиболее хорошо показывает себя при расчете эволюции образцов, содержащих большое количество повторяющихся или похожих частей. Наименее эффективен для больших хаотических образцов (здесь лучше использовать плиточный алгоритм).

Интерфейс программы.

Для того, чтобы ознакомиться с функциями системы, необходимо нажать клавишу '?'. В появившемся окне указаны основные команды (см. рис. 1).

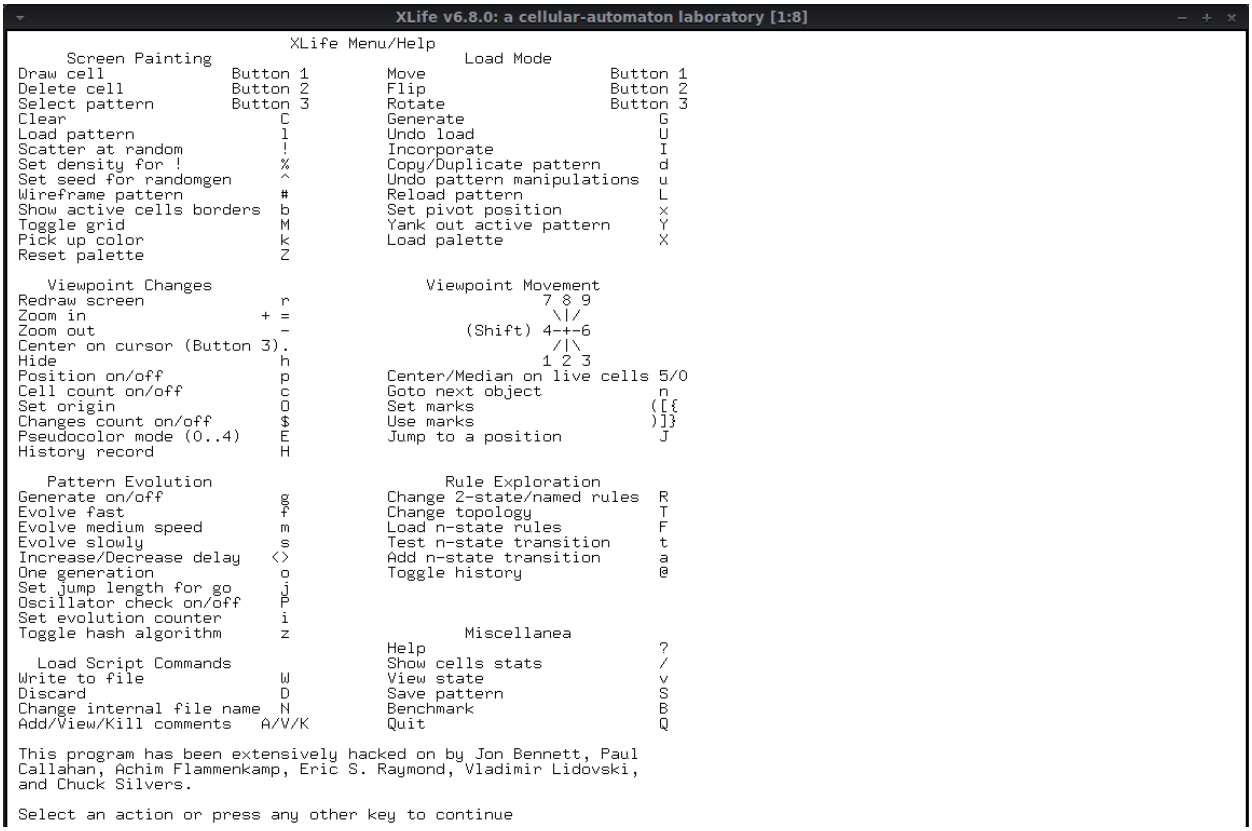

Рис. 1. Основные команды xlife.

Большинство команд, не представленных в окне справки, указаны в данном руководстве в таблице 6:

Таблица 6. Команды Xlife

Команда	**Описание**
!	Заполняет экран клетками случайным образом (по умолчанию "плотность" = 90%)
#	Изменяет режим отображения для предварительных (загружаемых) образцов. По умолчанию они показаны в нормальном режиме, при включении данной опции отображаются только границы клеток и не отображается общая граница образца.
$	Включает режим отображения количества изменений на каждом шаге.
%	Задает плотность для случайного заполнения (по

	умолчанию 90%).
([{	Устанавливает маркер на текущей позиции (всего 3 маркера, по одному на каждый из данных символов).
)] }	Переходит к соответствующему маркеру.
*****	Включает/выключает режим отображения текущей скорости (количество поколений в секунду).
+,=	Увеличивает масштаб отображения (масштаб 1:8 означает, что одна клетка представляет собой квадрат 8*8 пикселей, масштаб 2:1 – квадрат 2*2 клетки отображается как 1 пиксель и т.п.).
<	Замедляет эволюцию (увеличивает задержку на 10 мс).
>	Ускоряет эволюцию (уменьшает задержку на 10 мс).
-	Уменьшает масштаб.
.	Центрирует вид на текущем положении курсора.
/	Показывает количество клеток разных видов (цветов).
0	Центрирует вин на медианной позиции живых клеток.
1,End	Передвигает экран по диагонали влево-вниз.
2,Down	Передвигает экран вертикально вниз.
3,PageDown	Передвигает экран по диагонали вправо-вниз.
4,Left	Передвигает экран влево.
5	Центрирует вселенную на экране.
6,Right	Передвигает экран вправо.
7,Home	Передвигает экран по диагонали влево-вверх.
8,Up	Передвигает экран вверх.
9,PageUp	Передвигает экран по диагонали вправо-вверх.
Tab	Меняет цвет всех клеток одного типа на выбранный
?	Отображает окно справки по основным командам, используемым в xlife (см рис. 1).
@	Включает/выключает режим истории для автоматов с двумя состояниями. Также режим истории может быть включен добавлением '+' к строке, определяющей правила.
A	Добавляет комментарии.
B	Тест производительности. Введите количество поколений и получите время генерации.
C	Очищает текущий образец (или вселенную, если ни один образец не выбран).
D	Прерывает текущий сценарий загрузки, оставляя нетронутым текущую популяцию.

E	Переключает режим "псевдоцветов" для автоматов с двумя состояниями.
F	Загружает правила из выбранного файла.
G	Выполняет генерацию загруженного или активного образца на указанное количество шагов.
H	Переключает режим записи истории.
I	Встраивает предварительный (загружаемый или редактируемый) образец в основной образец.
J	Быстро перемещается к выбранным координатам (делает их текущими координатами курсора).
K	Удаляет комментарии, связанные с текущим образцом. Также очищает строку #K.
L	Быстро загружает копию предыдущего загруженного образца в позицию курсора.
M	Переключает режим отображения сетки.
N	Меняет внутреннее имя файла.
O	Устанавливает начало координат в текущее положение курсора.
P	Переключает режим проверки осциллятора. Эволюция остановится, когда будет обнаружено, что текущий образец - это осциллятор. Команда сравнивает исходный образец со следующими.
Q	Выход. Завершает работу программы.
R	Задает новые правила для текущей "вселенной". Правила могут задаваться как в качестве строк вида "B0..8/S0..8", так и именами вида life, seeds, wireworld и т.п. Существует возможность задания правил для дилеммы заключенного вида <float>$<float>. Подробнее указано далее в разделе "Правила".
S	Сохраняет вселенную в файле с расширением ".l". Если выделен образец, то сохраняется только он. Возможно выбрать формат для сохраняемого образца.
T	Устанавливает топологию. Возможно выделить прямоугольную область со связанными гранями (тор) или прямоугольную плоскость. 'Q' означает очень большой тор,

	режим по умолчанию, 'T' означает ограниченный тор, 'P' – ограниченную плоскость. Необходимо ввести размеры прямоугольной области (они будут округлены до кратных 8). Размер 0 означает псевдобесконечность. Примеры топологий: `T16,0', `T8x8', `P0,64', `T4096,4096', `T8X512', `P400*400'. В режиме истории можно установить плоские границы, просто обведя их 6-м цветом.
U	Отменяет загрузку образца.
V	Используется для просмотра комментариев.
W	Записывает скрипт (и удаляет его из памяти) из загруженных образцов в файл с расширением ".l"
X	Загружает палитру из файла с расширением ".colors" (по умолчанию). Файл состоит из текстовых строк. Любые пустые строки или строки, начинающиеся с "#" игнорируются. Спецификация RGB цветов должна использовать подобные строки: **color=1 0 0 255 #0000ff** **color=2 255 0 0** **color=4 255 255 0 cyan** **color=5 255 0 255**
Y	Если существует предварительный образец, то удаляет основной образец (вселенную) и перемещает на его место предварительный образец.
Z	Восстанавливает палитру по умолчанию.
^	Устанавливает "зерно" для генератора случайных чисел.
a	Добавляет переход с n состояниями - 6 цифр (одна для старого состояния, по одной для каждой ортогональной клетки-соседа и одна для нового состояния). Новое правило будет записано в файл new.transitions (требуется разрешение на запись в директорию с последним загруженным образцом).
b	Отображает значения xmin/xmax и ymin/ymax для живых клеток ("границы жизни").
c	Переключает режим отображения количества живых

	клеток (или плиток). Подсчет живых клеток замедляет эволюцию.
d	Дублирует предварительный образец. Позволяет интерактивно создавать множественные копии одного и того же объекта.
f	Устанавливает высокую скорость эволюции для 'g' (без задержки между шагами эволюции), режим по умолчанию.
g	Переключает исполнение игры (выполнять/остановить). Встраивает все предварительные образцы в основной, затем пошагово выполняет эволюцию до тех пор, пока не произойдет что-либо исключительное (достижение стабильного неменяющегося образца, нажатие клавиши, обнаружение осциллятора (только в режиме 'P') или окончание таймера, установленного командой 'i').
h	Прячет (останавливает) показ результата после каждого шага. Это режим может быть в несколько раз быстрее, чем установленный режимом 'g'. Однако, команда 'j' с надлежащим шагом демонстрирует лучшую скорость и более удобный вывод.
i	Устанавливает таймер: устанавливает максимальное число поколений (шагов) для выполнения последующей командой 'g' либо другой командой для эволюции. Значение по умолчанию (0) интерпретируется как никогда не останавливающийся расчет. Эта команда очень похожа на команду 'G'.
j	Устанавливает длину прыжка (шага) для команды 'g'. Значения 0 и 1 означают отсутствие прыжка. Большее значение означает более высокую скорость.
k	Подбирает цвет с текущей ячейки (ячейки, на которую указывает указатель мыши).
l	Загружает образец во "вселенную" из файла. Поддерживают расширения '.l' (по умолчанию), '.life', '.rle',

	'.mcl', '.lif', '.cells'. Позволяет накладывать несколько сохраненных состояний друг на друга для создания различных эффектов. Загруженный образец по умолчанию считается предварительным и может быть подвергнут линейным преобразованиям до встраивания его в основной образец (для распознавания он окружен ограничивающей рамкой). Возможно выбрать файл в списке в графическом окне либо напечатать его имя, при этом будет произведен поиск в папке-библиотеке и в текущей папке. Существует 2 режима загрузки. В первом режиме образец напрямую добавляется в пространство активного образца. Второй режим делает загружаемый образец "предварительным" и позволяет совершать с ним преобразования.
m	Устанавливает среднюю скорость эволюции для режима 'g' (задержка 250 мс между каждым шагом).
n	Переходит к следующему объекту. Эта команда позволяет находить удаленные корабли и т.п.
o	Встраивает все предварительные образцы в основной и совершает один шаг эволюции.
p	Переключает режим отображения окна с координатами мыши.
q	Позволяет пользователю выбрать некоторые опции (многоцветный/монохромный режим для хэша, вкл/выкл режим отображения полной информации).
r	Перерисовывает экран.
s	Устанавливает медленную скорость эволюции для режима 'g' (задержка 500 мс перед каждым шагом).
u	Отменяет преобразования над предварительным образцом, но не прерывает загрузку.
v	Отображает значения внутренних переменных и другую информацию.
x	Делает текущую позицию мыши точкой опоры для предварительного образца.

z	Предлагает пользователю выбрать алгоритм генерации (плиточный, гипер-хэш с быстрым ростом шага (шаг зависит от размера "обитаемого мира"), "быстрый" хэш с постоянным ростом шага (шаг зависит), хэш-режим с постоянным шагом, задаваемым пользователем).

Команды в shift-режиме.

PageUp, PageDown, Home, End, 1, 2, 3, 4, 6, 7, 8, 9 и клавиши-стрелки передвигают изображение быстрее.

Выделение правой кнопкой мыши с нажатой клавишей shift генерируют случайный образец.

Таблица 7. Команды в виджетах

PageUp, **PageDown**, **Home**, **End**	Прокручивают список в виджете.
Esc	Выход (закрывает виджет).
Shift-PageUp	Переходит в верхнюю директорию.
Left, **Right**, **Tab**	Передвигают текущую позицию к другому элементу в окне (режим загрузки, выбор текущей директории и т.п.).
Up, Down	Передвигают текущую позицию к другому элементу в списке (файлу или директории).
Печатные символы	Используются для печати имени файла.
Enter	Активирует текущий пункт в окне.
Backspace, Delete	Удаляют вводимые символы (по аналогии с любым текстовым редактором).

С помощью колесика мыши можно прокручивать "вселенную" по вертикали либо прокручивать список в виджете.

Таблица 8. Кнопки мыши в нормальном режиме

Левая кнопка	Активирует клетку под курсором, устанавливает ей значение выбранного состояния. В автоматах с двумя

	состояниями клетки всегда активируются состоянием 1. В автоматах с N состояниями вы можете выбрать текущий цвет в окне состояний (цветов клеток).
Нажатие на колесико	Удаляет клетку под курсором (устанавливает ее состояние в ноль). Чтобы очистить область, выделите ее с помощью мыши и нажмите 'C'.
Правая кнопка	Удаляет все предыдущие границы выделения. Если вы передвинете курсор с зажатой правой кнопкой, а затем отпустите ее, то прямоугольная область между точками нажатия и отпускания кнопки станет предварительной (как если бы вы загрузили новый образец и еще не встроили его в основной). Предварительный образец может быть передвинут, перевернут и отражен. Если вы просто щелкните правой кнопкой на клетке, то она станет центром экрана.

Таблица 9. Кнопки мыши при работе с предварительным образцом

Левая кнопка	Передвигает образец на текущую позицию.
Нажатие на колесико	Отражает образец относительно оси X.
Правая кнопка	Поворачивает образец по часовой стрелке относительно его точки опоры.

При работе с окном состояний (цветов):

– нажатие левой кнопкой: выбирает состояние;

– нажатие правой кнопкой: устанавливает цвет для данного состояния.

Опции виджетов.

Загрузка образца:

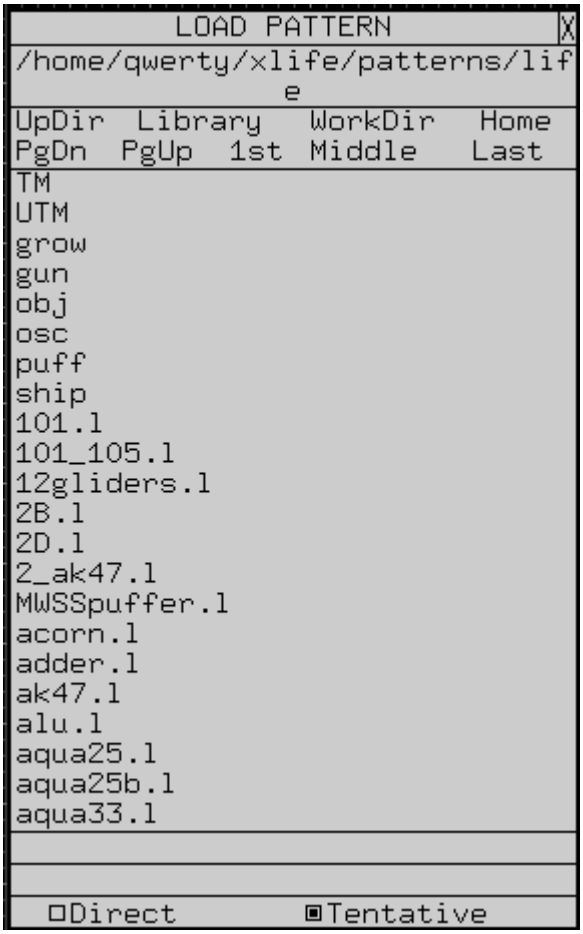

Рис. 2. Окно загрузки образца.

Радиокнопки позволяют выбрать прямой и предварительный режим загрузки. Прямой режим загружает образец напрямую в активное пространство, удаляя предыдущий образец. Предварительный режим (выбран по умолчанию) загружает образец в предварительное пространство, что позволяет применять к образцу различные модификации до встраивания в основной образец.

Сохранение образца:

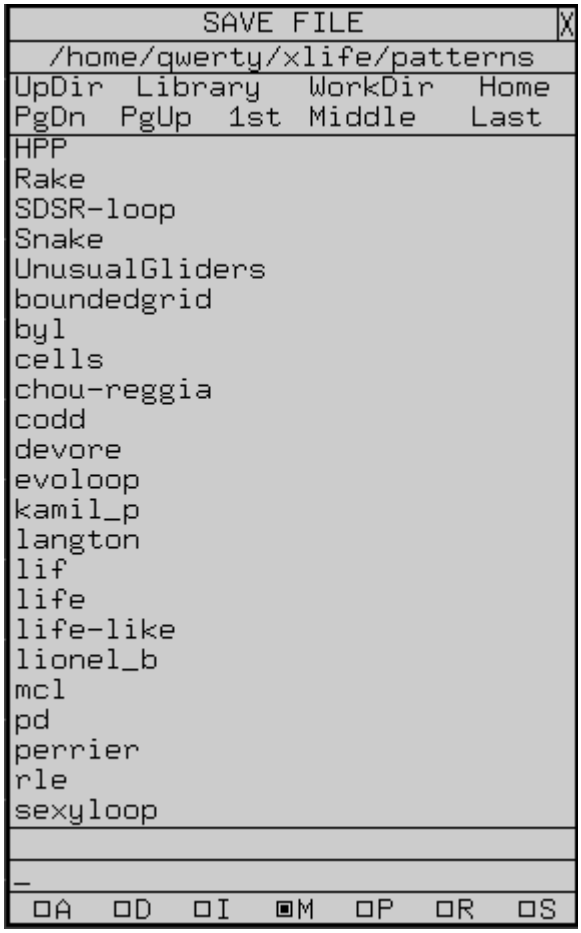

Рис. 3. Окно сохранения образца.

Радиокнопки позволяют выбрать один из следующих режимов: **A D I M P R S**. Формат 'S' представляет собой последовательность из P-блоков, по одному для каждого "пузыря" в изображении с соответствующими смещениями для каждого. Информация об остальных форматах размещена далее.

Основной формат загружаемых файлов.

Файл формата '.l' – это обыкновенный текстовый файл, состоящий из строк, разделенных символом перевода строки. Он интерпретируется как одна или несколько секций изображения, разделенных строками, начинающимися с '#'. Строки, начинающиеся с '##' считаются скрытыми комментариями и игнорируются.

```
#C This pattern was AQUA25.LIF in the Hensel archive
#C c/4 orthogonal spaceships
#C By Hartmut Holzwart and David Bell.
#C Artistic arrangement by Al Hensel.
#C Converted to structured format by ESR, January 1998
#C
#C Orthogonal c/4's are used in
#C SAWTOOTH, STRETCH.

#B 0
#K p4[c/4]
#P
.........................**
.........................**
.........................*
.......................*..*
.....................*....*
.....**............**.***.*
.....*..***.*...***......**
.....*........*..**
....****.*.*....*
....**........*.**
...*.........*****
.***.*...........***
*..*.............***
.***.*...........***
...*.........*****
....**........*.**
....****.*.*....*
.....*........*..**
.....*..***.*...***......**
.....**............**.***.*
.....................*....*
.......................*..*
.........................*
```

Рис. 4. Пример '.l' файла.

Некоторые указания формата могут использовать смещения, которые полезны при прямом режиме загрузки и с I-форматом. Эти смещения игнорируются в предварительном режиме загрузки.

Каждая секция изображения интерпретируется в соответствии с буквой формата, следующей за '#'. Существуют следующие буквы формата:

A - абсолютный.

Каждая следующая строчка интерпретируется как абсолютная (x, y) пара координат для клетки в состоянии 1 (по умолчанию). Этот формат является

устаревшим и его не следует использовать с новыми образцами. Однако, он используется по умолчанию – при отсутствии каких-либо указаний формата.

B и **E –** имя и блоки образцов.

Образцы, закрытые строками #B<имя> и #E "перепрыгиваются" при загрузке целого файла, но могут быть доступны при добавлении <имя> к имени файла. Это может быть полезно при создании связанных образцов внутри одного файла. Доступ осуществляется последовательным перепрыгиванием строк не в блоке, поэтому чрезмерно большое количество блоков в одном файле может замедлить процесс загрузки. Блоки образцов не могут быть вложенными.

Относительные секции изображения (D, I, M, P, ..) при нормальных условиях отрисовываются с центром вселенной в точке (0, 0). Это может быть изменено добавлением пары целых чисел, разделенных пробелом после буквы формата. Тогда они будут интерпретироваться как пара (x, y) смещений, и секция изображения будет нарисована с таким смещением левого верхнего угла.

Ведущая секция изображения без строки заголовка считается имеющей заголовок #A.

C – комментарий.

Строки, начинающиеся с 'C' – это комментарии, которые пользователь может автоматически записать в сохраняемы файл, и которые могут быть прочитаны с помощью Xlife.

D [xoff [yoff]] – относительно к предыдущей координатной паре.

Каждая следующая строка интерпретируется как (x, y) координатная пара относительно предыдущей (для непустой клетки). Первая относительна (0, 0) по умолчанию или, если указаны, к (xoff, yoff).

H currentHashMode p=population g=generations x=rootX y=rootY n=rootN d=rootDepth r=root - загрузка файла в хэш-режиме.

currentHashMode – режим хэша (1 - гипер-хэш, 2 - быстрый хэш, 3 - хэш с постоянным шагом), **population –** население (количество непустых клеток) загружаемого образца, **generations** – количество поколений, **rootX** и **rootY –** координаты загружаемого образца, **rootN –**размер "обитаемого мира", **rootDepth –** "глубина" мира, **root –** номер загружаемого квадрата в хэш-словаре. Далее в файле должны следовать строки в формате:

n sw se nw ne, где n – номер квадрата, добавляемого в словарь, sw, se, nw и ne – номера квадратов-предков квадрата n (юго-западного, юго-восточного, северо-западного и северо-восточного, соответственно). Более подробно о квадратах и хэш-режиме указано в пункте 2.2.2.

I [xoff [yoff [rotate [flip [delay]]]]] – включение.

Строка #I должна иметь поля, разделенные пробелами, после #I, состоящие из имени образца и пяти необязательных целых параметров (x, y offsets, rotation, flip, delay). Именованный образец загружается как, если бы он был включен в изображение в данной точке со всеми примененными преобразованиями. Смещения, если указаны, сдвигают точку загрузки образца относительно центра пространства образцов. Данный режим полезен для сборки шаблонов коллекций интересных образцов.

K – результирующая строка **–** специальный комментарий.

Эта строка должна определять результаты эволюции образца.

M [xsize, ysize] [xoff [yoff]] – групповое кодирование с

необязательными размерами и смещениями. Эти размеры игнорируются текущей версией Xlife. Только символы **b**, **$**, **!**, буквы от **A** до **X** и от **o** до **y**, а также цифры от **0** до **9** должны использоваться (помимо пробелов, которые игнорируются). Данные представляют собой упорядоченный ряды сверху вниз и, в то же время, каждая строка упорядочена слева направо. **$** представляет конец каждой строки, а **!** представляет собой конец образца. Для автоматов с двумя состояниями **b** – это

"мертвая" клетка, **o** – "живая" клетка. Для правил с большим количеством состояний **. –** "мертвая" клетка, состояния 1 - 24 представлены как **A - X**, состояния 25 - 48 как **pA - pX**, 49 - 72 как **qA - qX**, ..., 241 - 255 как **yA-yO**. Любой неверный символ интерпретируется как пробел и игнорируется.

N – имя.

Эта строка содержит внутренне имя образца 9которое может отличаться от имени файла).

O – владелец.

Строка содержит информацию об авторе данного файла. Записывается в форме id "name"@machine date, например:

#0 jb7m "Jon C. R. Bennett"@sushi.andrew.cmu.edu Fri Jan 12 18:25:54 1990

Примечание: вместо **O** может использоваться символ **0** (ноль).

P [xoff [yoff]] – изображение с необязательной координатной парой смещения.

Каждая строка в секции интерпретируется как строка изображения. Каждый символ ***** оживляет соответствующую клетку. Все другие символы оставляют соответствующую клетку "мертвой". Для автоматов со многими состояниями символами могут быть буквы, цифры, @, ~.

R [xoff [yoff]] – относительный с необязательными смещениями.

Каждая следующая строка интерпретируется как (x, y) координатная пара, относительная центру пространства образцов. Эти относительные секции изображения нормально отрисовываются с координатами 0, 0 в центре пространства образцов. Может быть изменено добавлением пары целых чисел, разделенных пробелом, после символа формата. если это сделано, то эти числа будут интерпретироваться как пара x и y смещений. Секция изображения будет отрисована с указанными смещениями.

T survivedigits/borndigits – устанавливает новые правила переходов.

Эта строка определяет новый набор правил с двумя состояниями. **survivedigits** и **borndigits –** это списки цифр от 0 до 8, означающих количество соседей, при которых центральная клетка соответственно выживает/рождается. Например, правил по умолчанию (жизнь Конвея):

#T 23/3. Также могут быть использованы именованные правила ('wireworld', 'seeds', 'life', 'lifehistory' or 'life+', 'brian' и т.п.). Возможно добавлять топологию после ':'. например, 'life+:T200,200' означает жизнь с историей на сетке торе размером 200*200. Можно использовать определения правил вида B0...8/S0...8, цифры после B означают количество соседей, при которых центральная клетка рождается, после S – выживает.

U filename - использование**.**

Формат – это #U с последующим именем файла. Загружает файл с набором также как и при наборе пользователем имени файла с клавиатуры. Если файл уже был загружен, он не будет перезагружен. Возможно добавлять топологию после ':' как в директиве #T.

X filename – палитра для пикселей.

Загружает палитру из файла с указанным именем.

Команда #I как было указано выше имеет следующий формат:

#I <pattern> <x> <y> <rotate> <flip> <delay>

<pattern> – имя образца (описано ниже).

<x>,<y> – целые числа, описывающие горизонтальное и вертикальное смещения.

<rotate> – целое число, которое показывает количество раз, которое образец будет повернут по часовой стрелке на 90° относительно точки опоры. Любое

значение (положительное или отрицательное) допустимо, так как оно берется по модулю 4.

<flip> – это множитель (1 или -1) для y координаты, который показывает отражение относительно оси X. Другие целые значения допустимы , они автоматически приводятся к значению 1 или -1.

<delay> – целое число, означающее количество генераций, выполняющихся до загрузки образца (отрицательные значения дают тот же эффект, что и ноль).

Примечание: все преобразования, применяемые к включаемому образцу, относительно применяются и к образцу, включающему его. Таким образом, загрузка сборки включенных образцов работает так же, как и следует ожидать.

Имя образца может быть в одной из следующих форм:

<file> –включает целиком файл <file> (аналогично старому формату).

<file>:<name> – включает блок образцов <name> в <file>.

:<name> – включает блок образцов <name> в текущем файле.

Примечание: формат <file>: не позволяется.

Файл может включать буквальные образцы или блоки образцов. Блок образцов – это образец, данный в любом приемлемом формате между строкой, содержащей "#B <name>", и другой строкой, содержащей "#E". Блоки образцов "перепрыгиваются" при включении целого файла. Блоки образцов не могут быть вложенными.

При сохранении образца в режиме 'I' программа автоматически производит анализ по выявлению "пузырей" (полностью соединенных областей живых клеток). Повторяющиеся пузыри распознаются даже если они повернуты или отражены. Если существует только один пузырь, то сохранение будет идентично формату #P. В противном случае выход будет в виде списка пузырей со строками #I, которые пересоздают сохраненный образец. Это имеет два преимущества: дублированные пузыри записываются только один раз, а общие элементы образцов распознаются и именуются (таким образом сохранение файла является еще и своеобразной переписью).

Поддержка N состояний.

XLife поддерживает автоматы с количеством состояний до 64 при использовании четырехклеточного симметричного соседства (окрестность Фон Неймана). Также возможно использовать восьмиклеточное соседство (окрестность Мура) для автоматов поколений(до 256 состояний), WireWorld и др. Более подробно правила будут объяснены ниже.

Примечание: при перевороте и отражении образцов с несимметричными правилами могут происходить некорректные преобразования.

При использовании программой таких правил используются цвета для индикации состояний. Файлы с образцами могут содержать цифры, чтобы обозначить состояния клеток, ***** интерпретируется как 1. Цветовые кнопки переключатели расположены в правом верхнем углу основного окна. При нажатии курсором на такую кнопку устанавливается цвет для левой кнопки мыши.

Синтаксис правил определения переходов выглядит следующим образом. Каждая строка содержит либо директивы, либо 6 цифр (букв) представленных как:

<old-state><neighbor><neighbor><neighbor><neighbor><new-state>

Для <old-state> или <neighbor> можно также использовать набор состояний: цифры или буквы в квадратных скобках. Следует использовать буквы вместо цифр для состояний, больших 9: A вместо 10, B вместо 11 и т.д. Буква 'a' означает 36, 'b' - 37 и т.д., '@' означает 62, '-' - 63. Комментарии, начинающиеся с # разрешены в файле.

Возможно разрешить автозагрузку правил при загрузке файла, добавив директиву #U в файл.

Директива **states <maxstates>** устанавливает максимальное количество состояний для данного правила.

Директива **passive <maxstate>** показывает, что все комбинации состояний клетки и соседей до maxstate включительно оставляют клетку неизменной. Данная директива может быть переопределена поздними правилами, указывающими переходы.

Директива **nosymmetries** устанавливает отсутствие симметрии в правилах. Может использоваться в правилах вида Langton Ants, Perrier loops и т.п.

Директива **rotate4reflect** устанавливает тип симметрии. Используется в правилах Bank I, II, III, IV и т.п.

Директива **Moore** устанавливает вид соседства. Эта директива должна предшествовать декларациям состояний и симметрии.

Также возможно указать правила, зависящие от количества соседей. Строка:

S(N*C)R

где S, N, C и R –это цифры интерпретируется так: если клетка имеет состояние S и ровно N соседей состояния C, то ее состояние-результат будет R. Не работает с `nosymmetries' или `Moore'.

wireworld-4.r

```
# Rules file for Wireworld-4
#
# More on this automaton may be found in the January 1990 issue of
# Scientific American (Computer Recreations, p. 146).

# 0 is space, 1 is wire, 2 is tail, 3 is head
states 4

passive 2
0[0123][0123][0123][0123]0      # No way to make a space into signal or wire

# Signal propagation
2[0123][0123][0123][0123]1      # tail -> wire
3[0123][0123][0123][0123]2      # head -> tail

# 1 or 2 heads adjacent to a wire makes it a head
1(3*1)3                 # wire with 1 head adjacent -> head
1(3*2)3                 # wire with 2 heads adjacent -> head
1(3*3)1                 # wire with 3 heads adjacent -> wire (overpopulation)
1(3*4)1                 # wire with 4 heads adjacent -> wire (overpopulation)

# End of ruleset
```

Рис. 5. Пример файла правил.

Правила проверяются в порядке, данном в файле: применяется первое подходящее правило. Когда функция эволюции встречает соседство, для которого нет определенного перехода, она останавливается и выделяет данное соседство. Будет предложено ввести результирующее состояние. После ввода эволюция продолжается и новое правило перехода записывается в новый файл переходов в данной директории. Эта особенность не может быть активирована, если все переходы определены.

Дилемма заключенного.

Описывает необычный класс клеточных автоматов, которые моделируют модифицированную дилемму заключенного. Такие игры иллюстрируют стабильные и выигрышные стратегии сотрудничества/предательства для ситуаций, в которых каждый "агент" неоднократно взаимодействует с ближайшими соседями.

Таблица 10. Платежная матрица для дилеммы заключенного

Выплата	Сотрудничество	Предательство
Сотрудничество	1	a
Предательство	b	0

Для более интересной игры b должно быть больше 1 (симуляция Ллойда обрабатывает случай только если a = 0). В каждом раунде каждая клетка вначале играет в игру со всеми своими соседями. Затем каждая клетка смотрит на прибыль своих соседей (и на свою собственную) и переходит на стратегию, обеспечивающую наивысший доход.

Чтобы выбрать такой тип правил нужно использовать команду R в следующей форме:

R<b>$<a>

Также возможно использовать директиву #T в файле.

В следующей таблице указаны состояния и их значения для данных правил:

Таблица 11. Значения состояний для дилеммы заключенного.

0	Неподвижный (неживая ячейка).
1	Живая, всегда сотрудничает.
2	Живая, всегда предает.
3	Переход от сотрудничающей к предающей (ее поведение аналогично предающей).
4	Переход от предающей к сотрудничающей (ее поведение аналогично сотрудничающей).

Окружение.

Программа имеет встроенную библиотеку образцов, ее адрес определяется содержимым переменной LIFEDIR, по умолчанию /usr/share/xlife (для ОС MS Windows используется другой адрес).

```
XLIFEPATSDIR    = xpatsdir root/patterns
CCOPTIONS       = -DLIFEDIR=\"$(XLIFEPATSDIR)\" -DHASHBITS=20 -DVFREQ=56
CDEBUGFLAGS     = -O5 -g
CXXDEBUGFLAGS   = $(CDEBUGFLAGS)
EXTRA_LIBRARIES = -lm -lstdc++
DOCDIR          = /usr/share/doc/xlife/
INSTBINFLAGS    = -m 0755
INSTPGMFLAGS    = -s $(INSTBINFLAGS)
INSTMANFLAGS    = -m 0444
INSTDATFLAGS    = -m 0644
MANDIR          = /usr/share/man/man6
MANSUFFIX       = 6
```

Рис. 6. Содержимое переменных окружения по умолчанию.

Если переменная LIFEDIR является списком директорий, разделенных запятыми, то все эти директории (и их поддиректории первого уровня) просматриваются для поиска файлов и наборов правил.

По умолчанию библиотека содержит следующие директории:

Таблица 12. Содержание библиотеки образцов.

life	Содержит обширную библиотеку интересных образцов Игры Жизнь Конвея.
life-like	Содержит большую библиотеку образцов, похожих на GoL (seeds, life without death, highlife replicator, brian's brain, starwars, ...).
codd	Содержит правила перехода и некоторые демонстрации.
devore	Вариации самовоспроизводящихся компьютеров CODD.
HPP	Ранняя модель движения молекул газа.
perrier	Содержит самовоспроизводящийся калькулятор.
wireworld	Содержит правила перехода и образцы Мира Проводов.
wireworld-4	Содержит правила перехода и образцы WireWorld-4. WireWorld-4 – это вариант классического мира проводов для окрестности Фон Неймана.
byl, chou-reggia	Содержит небольшие репликаторы с N

	состояниями.
langton	Знаменитые правила (муравей Лэнгтона) и образцы муравьев.
evoloop, sexyloop, SDSR	Содержит эволюционирующие репликаторы.
pd	Содержит образцы для варианта Ллойда игр дилеммы заключенного.

Текущая версия XLife для распространения может иметь большее количество шаблонов и правил.

1.1.2. Описание хэш-алгоритма

В 1984 году Билл Госпер (настоящее имя – Ralph William Gosper, Jr.) опубликовал статью "Exploiting regularities in large cellular spaces" (Эксплуатация закономерностей в больших клеточных пространствах), в которой предложил новый алгоритм для расчета эволюции клеточных автоматов – алгоритм hashlife.

В алгоритме существуют два ключевых компонента – хэш-механизм и макро-ячейки. Хэш-механизм позволяет предотвратить повторное вычисление уже известных сценариев. Макро-ячейка представляет собой блок клеток размером 2^n на 2^n, где n – неотрицательное целое. Каждая ячейка определяет свой результат (макро-ячейку размером 2^{n-1} на 2^{n-1}, которая представляет собой результат эволюции исходной ячейки через 2^{n-2} шага). Макро-ячейки позволяют экономить память, поскольку для хранения каждой ячейки размером 2^n на 2^n требуется только пять указателей: на четыре ячейки-предка (квадранты) размером 2^{n-1} на 2^{n-1} и на ячейку-результат такого же размера.

Обычно требуется меньше макро-ячеек, чем может показаться на первый взгляд, в связи с некоторыми ограничениями при создании новой ячейки. Во-первых, макро-ячейка никогда не создается, если уже существует ячейка с такими же квадрантами. Это правило также рекурсивно распространяется на все квадранты. Когда алгоритм пытается сгруппировать четыре квадранта, чтобы получить уже существующую ячейку, хэш-механизм замечает совпадение и возвращает старую ячейку. Что более важно, старая ячейка уже может знать свой результат. Второе

важное ограничение: макро-ячейки размером 2^n на 2^n могут создаваться только если их пространственные и временные координаты являются множителями 2^{n-2}. Благодаря этому значительно сокращается количество вновь создаваемых ячеек.

Скорость эволюции может экспоненциально возрастать в регионах с повторяющимся "поведением" из-за повторного использования многих результатов при построении больших ячеек-результатов.

Самая маленькая ячейка, которая может обладать результатом, имеет размер 4 на 4. Такие и все большие ячейки возвращают свой результат в ответ на сообщения (обычно при создании большего результата). Если опрашиваемая макро-ячейка уже знает свой результат, она просто возвращает его. Если ячейке размером 4 на 4 неизвестен свой результат, то он вычисляется непосредственным применением правил клеточного автомата к каждой из четырех центральных клеток (в данной реализации для экономии времени все макро-ячейки размером 4 на 4 для автоматов с двумя состояниями вычисляются заранее на стартовом этапе). Большие ячейки определяют свой результат рекурсивно, используя результаты ячеек-предков.

Рассмотрим подробно рекурсивный алгоритм поиска результата большой макро-ячейки размером 2^n на 2^n. Назовем ячейки-предки северо-западной, северо-восточной, юго-западной и юго-восточной (СЗ, СВ, ЮЗ и ЮВ соответственно). Каждая такая ячейка в свою очередь состоит из меньших ячеек-предков (см. рис. 7).

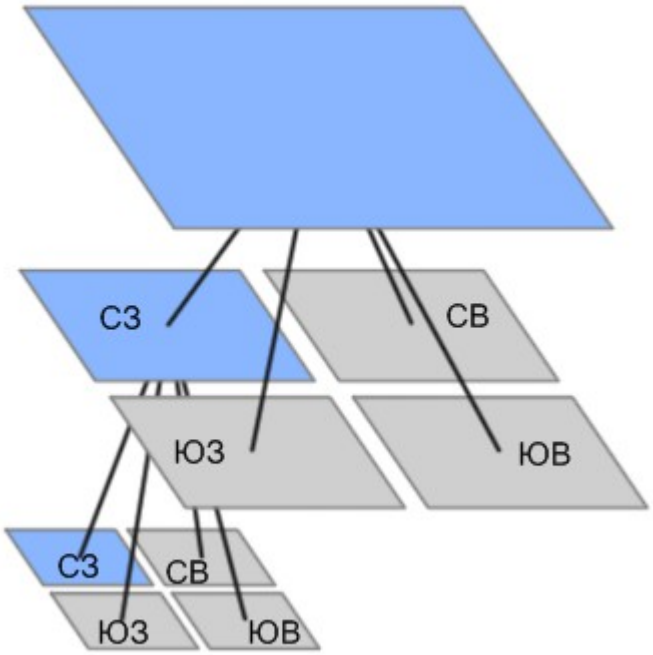

Рис. 7. Ячейки-предки.

Первоначально алгоритм находит результат для каждой из четырех ячеек-предков (ЮЗ, ЮВ, СЗ и СВ, см. рис. 8, здесь и далее макро-ячейки показаны в псевдотрехмерном формате, где по оси Z обозначается время, т.е. более "глубокими" являются более "поздние" ячейки – результаты).

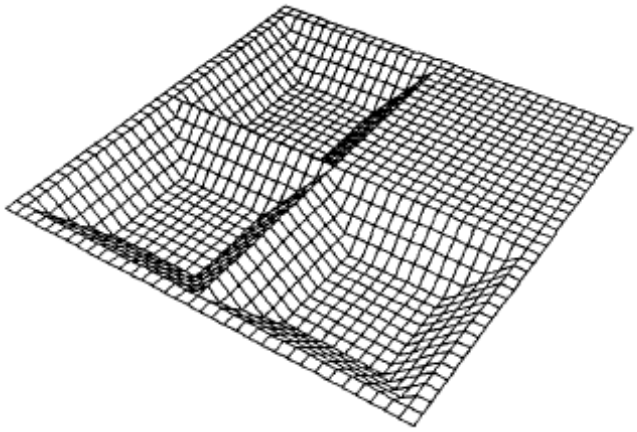

Рис. 8. Поиск результата ячеек-предков.

Далее необходимо найти ячейки-результаты размером 2^{n-2} на 2^{n-2} "искусственных" ячеек, образованных на стыках квадрантов исходной ячейки (например, одна из этих ячеек состоит (слева направо сверху вниз) из СВ-квадранта ячейки СЗ, СЗ-квадранта ячейки СВ, ЮВ-квадранта ячейки СЗ, ЮЗ-квадранта

ячейки СВ, остальные формируются схожим образом). Также нужно найти центральную ячейку-результат объединения частей сразу четырех квадрантов исходной клетки (см. рис. 9).

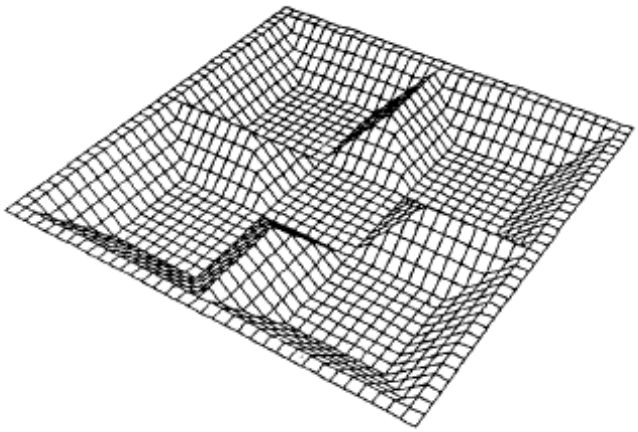

Рис. 9. Поиск результата ячеек-объединений квадрантов.

Следующим шагом необходимо найти будущие квадранты искомого результата, для этого происходит поиск результатов объединений граничащих между собой ячеек, полученных на предыдущих шагах (см. рис. 10).

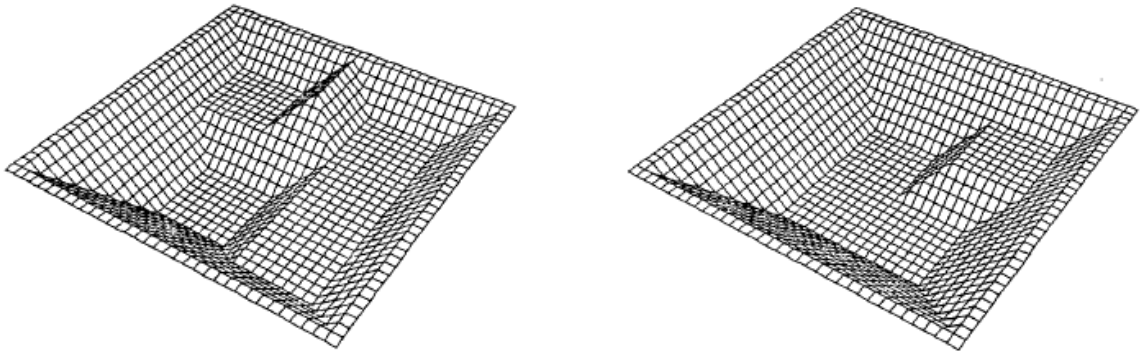

Рис. 10. Поиск квадрантов искомого результата.

Искомым результатом является объединение четырех ячеек, полученных на предыдущем шаге. Данный алгоритм применяется рекурсивно для всех ячеек, у которых неизвестен результат (при использовании правил с двумя состояниями для ячеек размером 4 на 4 результат вычисляется заранее, для остальных правил результат ячеек размером 4 на 4 вычисляется по этим правилам при необходимости).

1.1.3. Реализация хэш-алгоритма

Исходя из того, что ключевыми объектами в алгоритме Билла Госпера являются хэш-механизм и макро-ячейки, именно их исполнение и явилось главенствующим в реализации хэш-алгоритма системы моделирования клеточных автоматов Xlife.

Важную роль в быстродействии играет структура макро-ячеек и словаря – объекта, в котором хранятся уже известные ячейки.

Основными функциями в реализации хэш-алгоритма в программе Xlife являются recursiveUnite, findInDictionary, getResult, unite2, addToDictionary. Во всех функциях, если не указано особо, под ячейкой подразумевается ее номер в массиве hashVector.

2.2.1. Макро-ячейки

В данной реализации макро-ячейки, являющиеся одной из основ хэш-алгоритма, представляют собой класс (объект) node, содержащий несколько статических переменных (hashVector, hashTable, maxSize, curSize, prevMaxSize, curPop, populationVector), несколько переменных – полей каждого объекта (nw, ne, sw, se, result, hashNext, popPointer), а также методы класса и «дружественные» функции:

```
class node
{
        static node *hashVector;
        static int *hashTable;
        static int maxSize, curSize, prevMaxSize, curPop;
        static vector<bigNumber> populationVector;
        int nw, ne, sw, se, result;
        int hashNext;//for dictionary
        int popPointer;
public:
        …
```

}

Статический указатель на ячейку hashVector – это динамический массив, предназначенный для хранения уже известных макро-ячеек. Играет одну из ключевых ролей в реализации словаря.

Статический указатель на целое число hashTable – динамически изменяемая хэш-таблица, предоставляющая возможность быстрого доступа к ячейке по известным квадрантам (более подробная информация содержится в пункте 2.2.3.2).

Статические целочисленные переменные maxSize, curSize, prevMaxSize, curPop означают, соответственно, максимально возможное число хранимых в словаре ячеек, текущее число ячеек в словаре, предыдущее максимально число ячеек (необходимо для т.н. «сборщика мусора», т.е. при удалении более неиспользуемых ячеек) и текущее количество ячеек в «векторе населения» (численность населения запоминается не для всех ячеек).

Статический вектор длинных чисел (чисел с неограниченным количеством знаков) populationVector содержит в себе численности населения для макро-ячеек, у которых известен этот параметр («вектор населения»).

Целочисленные переменные nw, ne, sw, se, result – это, соответственно, адреса в массиве ячеек северо-западного, северо-восточного, юго-западного, юго-восточного квадранта и ячейки-результата, определенных для каждой макро-ячейки.

Целочисленная переменная hashNext – это адрес ячейки, имеющей такое же значение хэш-функции, применяемое в случае коллизии в хэ-таблице (более подробная информация содержится в пункте 2.2.3.2).

Целочисленная переменная popPointer – это адрес в «векторе населения», в котором содержится значение численности населения для данной макро-ячейки.

2.2.2. Словарь макро-ячеек

Словарь макро-ячеек – это структура, в которой хранятся уже известные ячейки с целью избежать повторного расчета уже вычисленного результата.

Реализуемый словарь должен отвечать нескольким ключевым требованиям:

– словарь должен быть быстрым (от скорости его работы во многом зависит быстродействие всего хэш-алгоритма);

– словарь должен эффективно расходовать память (он должен занимать наименьший возможный объем памяти);

– необходимо иметь механизм изменения размера словаря (при достижении максимального размера нужна возможность расширения словаря, а при нахождении в нем избыточного количества неиспользуемых ячеек нужно средство «сборки мусора»);

– должны быть предусмотрены функции добавления новых ячеек в словарь и поиска ячейки по ее квадрантам.

Были рассмотрены следующие варранты реализации словаря: использование стандартного контейнера C++ std::map (представляет собой ассоциативный массив), использование контейнера стандарта C++11 std::unordered_map (представляет собой хэш-таблицу), либо разработка собственной хэш-таблицы.

Ассоциативный массив (также часто называется словарь, карта или хэш) — структура данных, которая позволяет хранить пары вида «ключ, значение» (k, v). В нем определены следующие базовые операции:

INSERT – операция добавления пары в массив;

DELETE – операция удаления пары из массива;

SEARCH – операция поиска пары в массиве.

Иногда к этому списку добавляют операцию изменения пары.

Предполагается, что в ассоциативном массиве не могут храниться две пары с одинаковыми ключами.

В паре «ключ, значение» (k, v) значение v обычно называют значением, ассоциированным с ключом k.

С точки зрения интерфейса ассоциативный массив часто рассматривается в качестве обычного массива, в котором в качестве индексов используются не только целые числа, но и значения других типов — например, строки или объекты. С точки зрения реализации ассоциативные массивы представляют собой интерфейс к различным деревьям поиска (в библиотеке STL – красно-черные деревья) или к хэш-таблицам.

Хэш-таблица – структура данных, которая представляет собой реализацию ассоциативного массива, т.е. имеет возможность хранить пары вида «ключ, значение» и предоставляет те же базовые функции, что и ассоциативный массив. Для адресации используется хэш-функция – функция, которая преобразует ключ в индекс в хэш-таблице.

Хэш-таблицы реализуют основные функции за время O(1), т.е. имеют существенное преимущество перед другими структурами при хранении большого количества значений. Однако, имеются некоторые ограничения: во-первых, не гарантируется, что время выполнения каждой операции мало, во-вторых, существует ненулевая вероятность возникновения коллизий (поскольку множество возможных значений ключей больше, чем размер таблицы).

Коллизия (от лат. collisio — столкновение) – ситуация, при которой несколько блоков данных (или значений, в случае с хэш-таблицей) имеют одинаковое значение хэш-функции, т.е. попадают в одну и ту же ячейку хэш-таблицы. Существует несколько способов разрешения коллизий.

Метод цепочек (корзин).

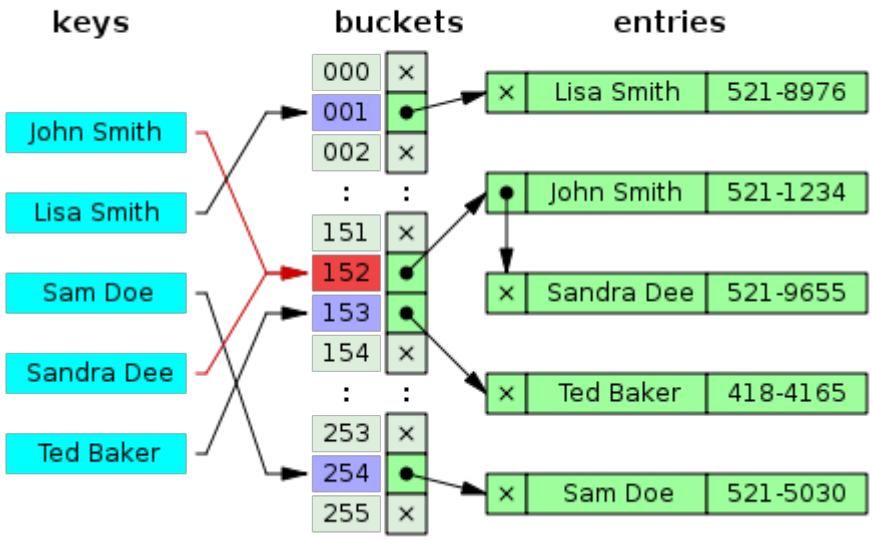

Рис. 11. Метод цепочек.

При возникновении коллизии в ячейку таблицы записывается не само значение, а ссылка на список значений, имеющих одно и то же значение хэш-функции. Если при добавлении в хэш-таблицу в заданную ячейку встречается

ссылка на элемент списка, происходит коллизия. Тогда происходит простая вставка элемента как узла в этот список. При поиске происходит проход по списку до тех пор, пока не будет встречен необходимый ключ/значение. При удалении ситуация аналогична.

Открытая адресация.

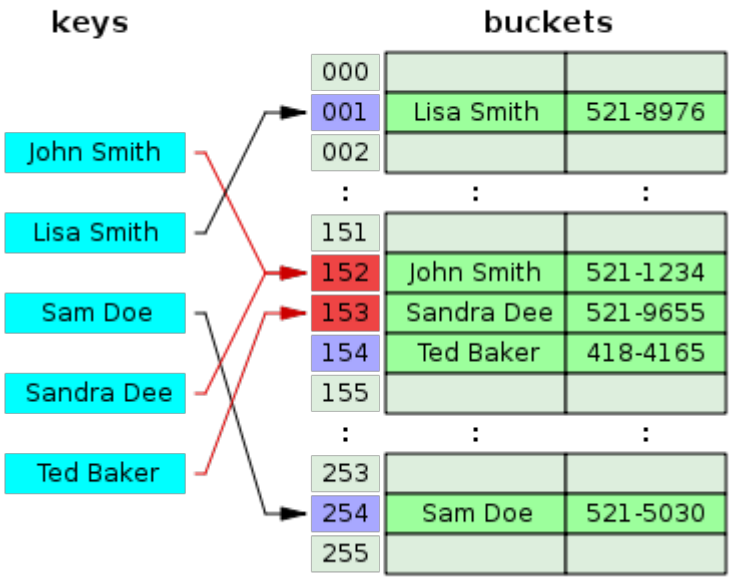

Рис. 12. Метод открытой адресации.

В случае метода открытой адресации (или закрытого хеширования) все элементы хранятся в самой хэш-таблице, без использования списков. В отличие от метода цепочек может возникнуть ситуация, когда таблица окажется полностью заполненной, и будет невозможно добавлять в неё новые элементы. Решением может быть динамическое изменение размеров хэш-таблицы с полной перестройкой.

Для разрешения коллизий применяются несколько подходов. В частности, это метод линейного исследования. В этом случае при возникновении коллизии следующие за текущей ячейки проверяются до тех пор, пока не найдётся свободная ячейка, в которую и будет помещен новый элемент. При достижении конца таблицы, поиск будет циклически продолжен с начала.

Линейное хеширование довольно просто реализуется, однако с ним связана значительная проблема – кластеризация. Это явление создания длинных последовательностей занятых ячеек, из-за чего увеличивается среднее время поиска

в таблице. Для снижения эффекта кластеризации используется еще один метод разрешения коллизий – двойное хеширование. Его основная идея заключена в том, что вместо линейного смещения на одну позицию происходит смещение на число позиций, определенных другой хэш-функцией.

Сложный вопрос в реализации хэширования с открытой адресацией – это операция удаления элемента. Дело в том, что если будет просто удален элемент из таблицы, то станет невозможен поиск ключа, в процессе вставки которого данная ячейка оказалась заполненной. Одним из способов решения этой проблемы является «нанесение меток» на очищенные ячейки таким образом, чтобы отличать их от занятых.

Операции вставки нового элемента и поиск по ключу в ассоциативном массиве std::map обладают логарифмической временной сложностью, т.е. при увеличении количества объектов в массиве эти операции будут выполняться медленнее. В то же время основные функции в хэш-таблицах имеют константную сложность, что означает, что время их выполнения не зависит от размеров таблицы (только в худших вариантах, когда необходимо произвести перерасчет таблицы, время будет довольно велико по сравнению с остальными случаями). Поэтому за основу словаря была взята хэш-таблица. В ходе данной работы выяснилось, что, несмотря на многие достоинства, реализация хэш-таблицы из стандартной библиотеки std имеет ряд недостатков: в частности, неоптимальный расход памяти и время поиска/вставки (поскольку стандартная реализация не заточена под конкретные нужды этого проекта). В связи с этим было принято решение разработать хэш-таблицу, которая была бы наиболее эффективна для работы с хэш-алгоритмом Билла Госпера.

В результате в данной работе была создана реализация хэш-таблицы с использованием метода цепочек с целью разрешения коллизий. Созданный словарь включает в себя динамически изменяемый массив hashVector, в котором непосредственно хранятся макро-ячейки и динамически изменяемый целочисленный массив hashTable, который является таблицей, используемый для быстрого доступа к ячейкам (в нем хранятся номера ячеек в hashVector). С целью

экономии оперативной памяти не используются пары вида «ключ, значение», вместо чего хэш-функция вычисляется непосредственно по макро-ячейке:

```
inline int node::hashF()
{
	return (unsigned)(65537*nw + 257*ne + 17*sw + 5*se)%maxSize;
}
```

Хэш-функция возвращает индекс в хэш-таблице, соответствующий данной ячейке.

При значительной заполненности таблицы с целью уменьшить количество коллизий производится увеличение размера hashVector и hashTable, а также полный пересчет таблицы. При возникновении коллизии организуется список ячеек с использованием поля hashNext.

2.2.3. Функция findInDictionary

Синтаксис функции:

inline int findInDictionary (n0, n1, n2, n3)

int n0;

int n1;

int n2;

int n3;

Аргументы:

Таблица 13. Аргументы функции findInDictionary

n0	Указывает на юго-западный квадрант искомой ячейки.
n1	Определяет юго-восточный квадрант искомой ячейки.
n2	Указывает на северо-западный квадрант искомой ячейки.
n3	Определяет северо-восточный квадрант искомой ячейки.

Описание функции:

Функция findInDictionary() производит поиск макро-ячейки в словаре по четырем указанным квадрантам. Если ячейка с такими квадрантами существует в словаре, возвращается ее адрес, иначе возвращается код ошибки (-3).

Исходный код функции:

```
inline int findInDictionary(int n0, int n1, int n2, int n3)
{
	if (n0 == zero && n1 == zero && n2 == zero && n3 == zero)
		return zero;
```

```
	int hash = hashF(n0, n1, n2, n3), i = node::hashTable[hash], prev = i;
	if (i == 0)
		return -3;//search failed: this node hasn't been hashed yet
	while (i != -1)
	{
		if (node::hashVector[i].sw == n0 && node::hashVector[i].se == n1 && node::hashVector[i].nw ==
n2 && node::hashVector[i].ne == n3)
		{
			if (prev != i)//move to "top"
			{
				node::hashVector[prev].hashNext = node::hashVector[i].hashNext;
				node::hashVector[i].hashNext = node::hashTable[hash];
				node::hashTable[hash] = i;
			}
			return i;
		}
		prev = i;
		i = node::hashVector[i].hashNext;
	}
	return -3;//search failed: this node hasn't been hashed yet
}
```

2.2.4. Функция unite2

Синтаксис функции:

```
inline int unite2 (int a0, int a1, int a2, int a3, int depth, int pop = 0)
    int a0;
    int a1;
    int a2;
    int a3;
    int depth;
    int pop;
```

Аргументы:

Таблица 14. Аргументы функции unite2

a0	Указывает на юго-западный квадрант искомой ячейки.
a1	Определяет юго-восточный квадрант искомой ячейки.
a2	Указывает на северо-западный квадрант искомой ячейки.
a3	Определяет северо-восточный квадрант искомой ячейки.
depth	Показывает т.н. «глубину» возвращаемой макро-ячейки (двоичный логарифм ее размера).
pop	Флаг численности населения ячейки (равен населению, если оно известно, иначе 0).

Описание функции:

Функция unite2() объединяет макро-ячейку из четырех указанных квадрантов. Если ячейка с такими квадрантами уже существует в словаре, возвращается ее адрес, иначе производится добавление в словарь и возвращается вновь созданный адрес. Результат ячейки не рассчитывается. Если pop равен 0, то функция рассчитывает население макро-ячейки, иначе население устанавливается равным pop.

Исходный код функции:

```
inline int unite2(int a0, int a1, int a2, int a3, int depth, int pop = 0)
{
	int res = findInDictionary(a0, a1, a2, a3);
	if (res != -3)
		return res;
	node tmp(a0, a1, a2, a3);
	tmp.result = -3;
	return tmp.addToDictionary(depth, pop);
}
```

2.2.5. Функция getResult

Синтаксис функции:

int getResult (int n, int depth)

int n;

int depth;

Аргументы:

Таблица 15. Аргументы функции getResult

n	Указывает на адрес макро-ячейки, результат которой необходимо найти.
depth	Показывает т.н. «глубину» возвращаемой макро-ячейки (двоичный логарифм ее размера).

Описание функции:

Функция getResult() находит результат макро-ячейки n и возвращает адрес этого результата в словаре. Если результат уже известен, то он непосредственно возвращается. Если неизвестен, то находится по рекурсивному алгоритму с помощью функции recursiveUnite(). Если ячейка имеет глубину 2, т.е. размер 4 на 4 клетки, а ее результат неизвестен (это возможно для автоматов с количеством состояний, большим двух), то применять рекурсивный алгоритм нельзя. В этом

случае результат вычисляется простым использованием текущих правил клеточного автомата.

Исходный код функции:

```
int getResult(int n, int depth)//depth is depth of n node
{
	if (n == zero)
		return zero;
	if (node::hashVector[n].result != -3)
		return node::hashVector[n].result;
	if(eventHandlerCounter++ == 50000)
	{
		eventHandler();
		eventHandlerCounter = 0;
		if (insideHashAlgorithm == 2)
			throw 1;
	}
	if (depth == 2)//Situation when depth = 2 and node hasn't been hashed yet
	{
		cell_t p[4][4], nw = 0, ne = 0, sw = 0, se = 0;
		p[0][0] = node::hashVector[node::hashVector[n].nw].nw;
		p[0][1] = node::hashVector[node::hashVector[n].nw].ne;
		p[0][2] = node::hashVector[node::hashVector[n].ne].nw;
		p[0][3] = node::hashVector[node::hashVector[n].ne].ne;
		p[1][0] = node::hashVector[node::hashVector[n].nw].sw;
		p[1][1] = node::hashVector[node::hashVector[n].nw].se;
		p[1][2] = node::hashVector[node::hashVector[n].ne].sw;
		p[1][3] = node::hashVector[node::hashVector[n].ne].se;
		p[2][0] = node::hashVector[node::hashVector[n].sw].nw;
		p[2][1] = node::hashVector[node::hashVector[n].sw].ne;
		p[2][2] = node::hashVector[node::hashVector[n].se].nw;
		p[2][3] = node::hashVector[node::hashVector[n].se].ne;
		p[3][0] = node::hashVector[node::hashVector[n].sw].sw;
		p[3][1] = node::hashVector[node::hashVector[n].sw].se;
		p[3][2] = node::hashVector[node::hashVector[n].se].sw;
		p[3][3] = node::hashVector[node::hashVector[n].se].se;
		nw = returnOneCell(p, 1, 1);
		ne = returnOneCell(p, 1, 2);
		sw = returnOneCell(p, 2, 1);
		se = returnOneCell(p, 2, 2);
		return unite2(sw, se, nw, ne, 1);
	}
	int z = recursiveUnite(n, depth);
	return node::hashVector[n].result;
}
```

2.2.6. Функция recursiveUnite

Синтаксис функции:

int recursiveUnite (int n, int depth)

int n;

int depth;

Аргументы:

Таблица 16. Аргументы функции recursiveUnite

n	Указывает на адрес макро-ячейки, результат которой необходимо найти по рекурсивному алгоритму.
depth	Показывает т.н. «глубину» возвращаемой макро-ячейки (двоичный логарифм ее размера).

Описание функции:

Функция recursiveUnite() находит результат макро-ячейки n с использованием рекурсивной части хэш-алгоритма. При необходимости функция рекурсивно вызывается для ячеек-предков и ячеек, состоящих из ячеек-квадрантов предков (в соответствии с описанием алгоритма в пункте 2.2.2). Функция возвращает адрес искомого результата в словаре.

Исходный код функции:

```
int recursiveUnite(int n, int depth)
{
	int a0 = node::hashVector[n].sw, a1 = node::hashVector[n].se, a2 = node::hashVector[n].nw, a3 = node::hashVector[n].ne;
	int a0se = node::hashVector[a0].se, a1sw = node::hashVector[a1].sw, a0ne = node::hashVector[a0].ne, a1nw = node::hashVector[a1].nw;
	int a0nw = node::hashVector[a0].nw, a2sw = node::hashVector[a2].sw, a2se = node::hashVector[a2].se, a3sw = node::hashVector[a3].sw;
	int a1ne = node::hashVector[a1].ne, a3se = node::hashVector[a3].se, a2ne = node::hashVector[a2].ne, a3nw = node::hashVector[a3].nw;
	int b00 = getResult(a0, --depth);
	int b01 = getResult(unite2(a0se, a1sw, a0ne, a1nw, depth), depth);
	int b02 = getResult(a1, depth);
	int b10 = getResult(unite2(a0nw, a0ne, a2sw, a2se, depth), depth);
	int b11 = getResult(unite2(a0ne, a1nw, a2se, a3sw, depth), depth);
	int b12 = getResult(unite2(a1nw, a1ne, a3sw, a3se, depth), depth);
	int b20 = getResult(a2, depth);
	int b21 = getResult(unite2(a2se, a3sw, a2ne, a3nw, depth), depth);
	int b22 = getResult(a3, depth);
	int c00 = 0, c01 = 0, c10 = 0, c11 = 0;
	if (currentHashMode == 1 || stepPower >= depth - 1)//normal recursive conditions
	{
		c00 = getResult(unite2(b00, b01, b10, b11, depth), depth);
		c01 = getResult(unite2(b01, b02, b11, b12, depth), depth);
		c10 = getResult(unite2(b10, b11, b20, b21, depth), depth);
		c11 = getResult(unite2(b11, b12, b21, b22, depth), depth);
	}
	else
	{
		if (depth == 2 && ev_mode == VALENCE_DRIVEN)//n is 4x4 node
		{
			int b00ne = getCell(b00, 2), b01nw = getCell(b01, 1), b10se = getCell(b10, 512), b11sw = getCell(b11, 256);
```

```
                int b01ne = getCell(b01, 2), b02nw = getCell(b02, 1), b11se = getCell(b11, 512), b12sw =
getCell(b12, 256);
                int b10ne = getCell(b10, 2), b11nw = getCell(b11, 1), b20se = getCell(b20, 512), b21sw =
getCell(b21, 256);
                int b11ne = getCell(b11, 2), b12nw = getCell(b12, 1), b21se = getCell(b21, 512), b22sw =
getCell(b22, 256);
                find2x2node(c00, b10se + (b11sw << 1) + (b00ne << 8) + (b01nw << 9), live1, live2,
maskNW, pos);
                find2x2node(c01, b11se + (b12sw << 1) + (b01ne << 8) + (b02nw << 9), live1, live2,
maskNW, pos);
                find2x2node(c10, b20se + (b21sw << 1) + (b10ne << 8) + (b11nw << 9), live1, live2,
maskNW, pos);
                find2x2node(c11, b21se + (b22sw << 1) + (b11ne << 8) + (b12nw << 9), live1, live2,
maskNW, pos);
            }
            else
            {
                c00 = unite2(node::hashVector[b00].ne, node::hashVector[b01].nw,
node::hashVector[b10].se, node::hashVector[b11].sw, depth - 1);
                c01 = unite2(node::hashVector[b01].ne, node::hashVector[b02].nw,
node::hashVector[b11].se, node::hashVector[b12].sw, depth - 1);
                c10 = unite2(node::hashVector[b10].ne, node::hashVector[b11].nw,
node::hashVector[b20].se, node::hashVector[b21].sw, depth - 1);
                c11 = unite2(node::hashVector[b11].ne, node::hashVector[b12].nw,
node::hashVector[b21].se, node::hashVector[b22].sw, depth - 1);
            }
        }
        int result = unite2(c00, c01, c10, c11, depth);
        node::hashVector[n].result = result;
        return n;
}
```

2.2.7. Функция addToDictionary

Синтаксис функции:

int node::addToDictionary (int depth, int pop = 0)

int depth;

int pop;

Аргументы:

Таблица 17. Аргументы функции addToDictionary

depth	Показывает «глубину» макро-ячейки (двоичный логарифм ее размера).
pop	Флаг численности населения ячейки (равен населению, если оно известно, иначе 0).

Описание функции:

Функция addToDictionary() добавляет макро-ячейку в словарь. Если текущее количество ячеек в словаре превысило 0,75 от максимального, происходит

изменение размеров словаря и пересчет хэш-таблицы. Если известна «численность населения» ячейки (pop не равен 0), то значением численности населения данной макро-ячейки считается pop, иначе населенность вычисляется (либо напрямую, либо через население меньших ячеек). Функция возвращает адрес добавленной ячейки в словаре.

Исходный код функции:

```
int node::addToDictionary(int depth, int pop = 0)
{
	if ((float)curSize/maxSize >= 0.75)
		rehash();
	int hash = this->hashF(), i = hashTable[hash];
	hashVector[curSize] = *this;
	if (depth > minSaveDepth && (depth - minSaveDepth) % depthStep == 0)//calculate population of node with depth = minSaveDepth + depthStep*N; //N > 0
	{
		bigNumber population(pop);
		getPopulation(hashVector[curSize].nw, population, depth - 1);
		getPopulation(hashVector[curSize].ne, population, depth - 1);
		getPopulation(hashVector[curSize].sw, population, depth - 1);
		getPopulation(hashVector[curSize].se, population, depth - 1);
		hashVector[curSize].popPointer = curPop++;
		populationVector.push_back(population);
	}
	if (depth == minSaveDepth)
	{
		int population = pop;
		if(strcmp(active_rules, "wireworld") && !pop)//not wireworld rules or we already know node's population (pop)
			calculatePopulation(curSize, population);
		hashVector[curSize].popPointer = curPop++;
		populationVector.push_back(population);
	}
	if (i == 0)
	{
		hashTable[hash] = curSize;
		return curSize++;
	}
	int p = hashVector[i].hashNext;
	while (p != -1)
	{
		i = p;
		p = hashVector[p].hashNext;
	}
	hashVector[i].hashNext = curSize;
	return curSize++;
}
```

2.3. Тестирование скорости хэш-алгоритма

Поскольку одной и основных задач создания данного программного модуля являлось повышение скорости работы, очень важным представляется вопрос тестирования эффективности алгоритма и его сравнение с плиточным алгоритмом эволюции.

Для сравнения скорости использовалась стандартная Unix-команда time, которая возвращает время выполнения программы, в следующем формате:

time ./xlife filename, где filename – имя файла с загружаемым образцом.

После загрузки файла производился выбор алгоритма и запуск расчета. Результаты сравнения занесены в следующую таблицу :

Таблица 18. Сравнение скорости хэш-алгоритма и пошагового алгоритма

Выбранный образец	Выбранные правила	Плиточный алгоритм		Алгоритм hashlife	
		Количество поколений	Время расчета	Количество поколений	Время расчета
Metacatacryst	Game of Life	50000	4m19s	22906437632	1m25s
Primes	Wireworld	66250	2m32s	48224256	1m13s
Tempesti loops	Tempesti	8700	4m00s	27040	52s
Golly ants	Langton ant	1201340	54s	$1.9*10^{21}$	37s
10-1	Evoloop	19200	4m25s	19200	1m4s
byl	Byl	8134	4m44s	196608	1m1s
Loop2	Chou-reggia2	6150	3m20s	8388624	59s

Как следует из вышеприведенной таблицы алгоритм hashlife показывает значительно более высокую скорость для большого количества правил и образцов.

Однако, несмотря на большое количество достоинств хэш-алгоритм имеет некоторые существенные недостатки: в частности, не реализована поддержка топологий, отличных от бесконечного поля. Также нет возможности проводить проверку осцилляторов. Не реализованы некоторые вспомогательные возможности плиточного алгоритма.

3. Организационно-экономическая часть

3.1. Технико-экономическое обоснование научно-исследовательской работы

В связи с широким практическим применением клеточных автоматов в различных прикладных областях деятельности, например, в криптографии, моделировании экономических, химических и физических процессов (более подробно указано в пункте 1.3) возникла необходимость в возникновении программного обеспечения, которое позволяет вычислять эволюцию клеточных автоматов.

Обычно при расчете используется простой плиточный алгоритм (он применяет правила ко всей решетке ячеек клеточного автомата). Однако, хэш-алгоритм, разработанный Биллом Госпером в 1984 году, позволяет существенно увеличить скорость расчета для образцов, содержащих повторяющиеся элементы, благодаря использованию однократно просчитанного результата эволюции для таких элементов.

В ходе данной работы было принято решение разработать модуль поддержки хэш-алгоритма для существующей программы xlife, поскольку подобное решение обеспечит интеграцию нового программного кода в уже имеющееся программное обеспечение.

Данная работа имеет коммерческую направленность. Помимо этого, она может принести еще и косвенный экономический эффект из-за того, что существенное увеличение скорости работы с хэш-алгоритмом позволит значительно сократить время моделирования клеточных автоматов.

3.2. Затраты на разработку программы

Затраты на разработку (З) складываются из затрат на заработную плату всех исполнителей ($З_{ЗП}$), затрат на электроэнергию ($З_{эл}$), затрат на основные материалы ($З_{о.м}$), а также затрат на вспомогательные материалы ($З_{в.м}$).

$$З_{р} = З_{ЗП} + З_{эл} + З_{о.м} + З_{в.м}$$

Разработка осуществлялась на протяжении 5 месяцев.

Расчет заработной платы исполнителей (ЗЗП)

В данном дипломном проекте исполнителями являются:

– руководитель дипломной работы;

– консультант по экономической части;

– консультант по экологической части.

$З_{ЗП} = З_{ЗПраз} + З_{ЗПрук} + З_{ЗПэкон} + З_{ЗПэкол}$

При расчете затрат на заработную плату необходимо учитывать обязательные отчисления на социальное страхование, которые рассчитываются на основе планового фонда заработной платы основных исполнителей и составляют 30%. Исходя из этого все затраты на заработную плату в расчетах должны умножаться на коэффициент $k = 1,3$.

Заработанная плата руководителя:

Размер оплаты труда дипломного руководителя проекта принимается равным 207 руб./ч. На одного дипломника приходится 19,5 часов рабочего времени, следовательно:

$З_{ЗПрук} = З_{ЗПрук.час.} * Т * k = 207 * 19,5 * 1,3 = 5247,45$ руб,

где $\mathbf{З_{ЗПрук.час.}}$ - стоимость рабочего часа дипломного руководителя.

k — коэффициент отчислений на социальное страхование;

Т – количество часов, приходящихся на одного дипломника.

Заработанная плата консультанта по экономической части дипломного проекта:

Стоимость рабочего часа консультанта по экономической части дипломного проекта составляет 300 руб. Консультант затрачивает на одного студента-дипломника 4 часа рабочего времени и получает за это $З_{ЗПэкон1.} = 4 * 300 = 1200$ руб.

$З_{ЗПэкон} = З_{ЗПэкон.1} * k,$

где **k** — коэффициент отчислений на социальное страхование;

$\mathbf{З_{ЗПэкон.1}}$ **—** заработная плата консультанта за период разработки дипломного проекта.

$З_{ЗПэкон} = 1200 * 1,3 = 1560$ руб.

Заработанная плата консультанта экологической части дипломного проекта:

Заработная плата консультанта по экологической части дипломного проекта составляет 207 руб./час. Консультант затрачивает на одного студента-дипломника 4 часа рабочего времени и получает за это $З_{ЗПэкол1.} = 4 * 207 = 828$ руб.

$З_{ЗПэкол} = З_{ЗПэкол.1} * k$,

где **k** — коэффициент отчислений на социальное страхование;

$З_{ЗПэкол.1}$ — заработная плата консультанта по экологической части за период разработки дипломного проекта.

$З_{ЗПэкон} = 828 * 1,3 = 1076,40$ руб.

Итого, расходы на заработанную плату основных исполнителей за 4 месяца разработки составили:

$З_{ЗП} = 5247,45 + 1560 + 1076,40 = 7883,85$ руб.

Затраты на электроэнергию

Затраты на электроэнергию за один месяц – $З_{эл.мес.}$:

$З_{эл.мес.} = N_у * а * F_Д * n_k * n_{ст}$, где:

$N_у$ – установленная мощность оборудования, кВт;

а – тариф за 1 кВт*ч, $а = 4,01$ руб.;

$F_Д$ – действительный фонд времени работы оборудования, час

($F_Д$ = 8 часов в день * 22 дня в месяц = 176 ч);

n_k – коэффициент использования оборудования по мощности и во времени, принимаемый равным 0,75;

$n_{ст}$ – коэффициент, который учитывает потери электроэнергии в сети, $n_{ст}=1,04$;

Затраты на освещение:

$F_Д$ = 4 часа в день* 22 дня в месяц = 88 часов;

$N_у$ = 3 лампы по 60 Вт = 0,18 кВт;

$З_{осв} = 0,18 * 4,01 * 88 * 0,75 * 1,04 = 49,54$ руб.

Затраты на работу оборудования:

$F_Д$ = 8 часов в день * 22 дня в месяц = 176 часов;

$N_у$= 0,5 кВт (системный блок, монитор и принтер);

$З_{об} = 0,5 * 4,01 * 176 * 0,75 * 1,04 = 275,25$ руб.

Итого суммарные затраты на электроэнергию (в месяц):

$З_{сум} = З_{осв} + З_{об} = 49,54 + 275,25 = 324,79$ руб.

Итого, затраты на электроэнергию за 5 месяцев разработки составили:

$З_{эл} = З_{сум} * 4 = 324,79 * 5 = 1623,95$ руб.

Затраты на основные материалы

Для разработки проекта использовалось оборудование, приобретенное ранее для выполнения других работ, поэтому $\mathbf{З_{о.м.}} = 0$

Затраты на вспомогательные материалы

Поскольку разработка программного обеспечения не является производством, а выполняется исключительно с использованием программных средств, к вспомогательным материалам можно отнести только такие, которые затрачиваются при окончательной печати дипломного проекта. Затраты на них приведены в таблице 19.

Таблица 19. Затраты на вспомогательные материалы

Наименование вспомогательных материалов	Единица измерения	Количество материалов	Цена, руб./ед.	Стоимость вспомогательных материалов, руб.
Упаковка 500 листов А4	шт.	1	150	150
Картридж для принтера	шт.	1	850	850
Итого	-	-	-	1'000

В итоге, суммарные затраты на разработку составляют:

$З_{р} = З_{ЗП} + З_{эл} + З_{о.м} + З_{в.м} = 7883,85 + 1623,95 + 0 + 1000 = 10507,80$ руб.

1.2.

1.3.

3.1. Затраты на эксплуатацию

Разработанный модуль хэш-алгоритма представляет собой программное обеспечение, которое суммарно будет выполняться около 1 часа в сутки. Количество рабочих часов в месяце = 22 рабочих дня * 1 час работы программы = 22 часа.

Затраты на месяц эксплуатации можно подсчитать по формуле:

$З_{экс} = З_{эл}$,

где $\mathbf{З_{эл}}$ — затраты на электроэнергию.

$З_{эл} = N_у * а * F_Д * n_к * n_{ст}$,

где $\boldsymbol{N_y}$ – установленная мощность оборудования, кВт;

а – тариф за 1 кВт*ч, *а* = 4,01 руб.;

$\mathbf{F_Д}$ – действительный фонд времени работы оборудования, час

($F_Д$ = 1 час в день * 22 дня в месяц = 22 ч);

$\mathbf{n_к}$ – коэффициент использования оборудования по мощности и во времени, обычно принимается в пределах 0,7 – 0,9;

$\mathbf{n_{ст}}$ – коэффициент, учитывающий потери электроэнергии в сети, $n_{ст}$ = 1,04.

Затраты на освещение:

$F_Д$ = 1 час в день * 22 дня в месяц = 22 часа;

N_y = 3 ламп по 60 Вт = 0,18 кВт;

$З_{осв}$ = 0,18 * 4,01 * 22 * 0,7 * 1,04 = 11,56 руб.

Затраты на работу оборудования:

$F_Д$ = 1 час в день * 22 дня в месяц = 22 часа;

N_y = 0,6 кВт (системный блок и монитор);

$З_{об}$ = 0,6 * 4,01 * 22 * 0,7 * 1,04 = 37,60 руб.

$З_{экс} = З_{эл} = З_{осв} + З_{об}$ = 11,56 + 37,60 = 49,16 руб. (в месяц)

Расчет годовых затрат

Годовыми затратами при разработке программного обеспечения является сумма затрат на разработку ($З_р$) и эксплуатацию анализатора ($З_{экс}$) в течение 12 месяцев:

$З = З_р + З_{экс} * 12 = 10507,80 + 49,16 * 12 = 11097,72$ руб.

3.4. Определение ожидаемого экономического эффекта от реализации проекта

Данный проект может быть востребованным на рынке, поскольку внедрение хэш-алгоритма позволяет существенно увеличить скорость расчетов эволюции клеточных автоматов, содержащих повторяющиеся элементы, вследствие чего возникает возможность значительно сократить время работы программы.

В среднем выручка от одной проданной копии программного обеспечения составляет порядка 5'000 руб. Если программу приобретут хотя бы 3 клиента, то экономический эффект будет:

$S = 5'000 * 3 - 11097,72 = 3902,78$ руб.

Прибыль оказывается выше затрат, следовательно, данный дипломный проект экономически оправдан.

4. Экология и безопасность

4.1. Экологический анализ производства

Данный дипломный проект разрабатывался с использованием персонального компьютера. Компьютерная техника может представлять некоторую опасность для здоровья человека. Вредными факторами, воздействующим на пользователя ПК, являются воздействие вредных компонент материалов комплектующих (бром, соединения хлора, свинец и др.), электромагнитное излучение, условия рабочей среды (метеорологические условия, освещенность). Немаловажным фактором, влияющим на утомляемость, также является эргономика рабочего места, дисплея и клавиатуры.

В настоящее время существует несколько международных стандартов, предъявляющих жесткие требования к технологии изготовления компьютеров и периферийного оборудования. Самый известный стандарт безопасности в этой области — TCO — группа стандартов добровольной сертификации на эргономичность и безопасность электронного оборудования (в основном, компьютерного), разработанная комитетом TCO Development, который является частью Шведской конфедерации профсоюзов. Сертификат TCO получают те продукты, которые соответствуют требованиям к экологичности используемых материалов, энергопотреблению, уровню электромагнитных полей, а также эргономике и т.д. TCO сейчас поддерживается большим количеством производителей. Персональный компьютер, на котором велась работа над данным проектом, имеет сертификат TCO'06.

4.2. Анализ условий труда

Метеорологические условия

Под метеорологическими условиями производственной среды понимают сочетание таких параметров как температура, относительная влажность, скорость движения воздуха и его запыленность. Вышеперечисленные параметры оказывают значительное влияние на физиологическую деятельность человека, его здоровье и

самочувствие, а также на надежность работы средств вычислительной техники. В производственных условиях большое значение имеет суммарное действие многих микроклиматических параметров.

Температура – это один из ключевых параметров, которые характеризуют тепловое состояние микроклимата. Значительное изменение температуры в помещении отрицательно сказывается на самочувствии человека. В душном помещении человек быстрее утомляется, у него может возникнуть головная боль. Это ведет к снижению работоспособности.

Другим важным параметром микроклимата является относительная влажность воздуха. Она оказывает существенное влияние как на организм человека, так и на работу технологического оборудования. При высокой относительной влажности снижается сопротивление изоляции, поэтому человек, который находится рядом или непосредственно пользуется таким оборудованием может подвергнуться поражению электрическим током.

Запыленность воздуха также отрицательно сказывается на здоровье человека. Наэлектризованный экран дисплея притягивает частицы взвешенной в воздухе пыли, так что вблизи него качество воздуха ухудшается, и оператору приходится работать в более запыленной атмосфере, что может вызвать кожные заболевания. Запыленный воздух так же может содержать различные сильнодействующие примеси (соли, кислоты, газы), которые при контакте с организмом человека вызывают отравления, раздражение органов дыхания и слизистых, аллергические заболевания, и даже могут привести к раковым заболевания.

Скорость движения воздуха тоже влияет на деятельность человека. В плохо проветренном помещении самочувствие человека ухудшается из-за того, что в организм человека с вдыхаемым воздухом не поступает нужного количества кислорода, необходимого для нормального функционирования мозга. Это ведет к снижению мысленной деятельности, а долгое пребывание в не проветренном помещении вызывает сильные головные боли и даже обмороки. В помещении с большой скоростью движения воздуха возникают сквозняки, что может привести к возникновению простудных заболеваний с различными осложнениями.

Электромагнитное излучение

Электронно-вычислительная техника является источником ЭМИ. Систематически воздействуя на организм человека, ЭМИ приводит к изменению нервной, сердечной и других систем человека. Благодаря современным технологиям (снижение потребляемой мощности, ЖК-дисплеи), ЭМИ ПК и периферийного оборудования не превышает установленных стандартами безопасных для здоровья значений.

Электрический ток

Электрические установки, к которым относится всё оборудование ЭВМ, представляют для человека большую потенциальную опасность. Основная опасность электроустановок такова: токоведущие проводники корпуса ЭВМ и прочего оборудования, оказавшегося под напряжением в результате повреждения изоляции, не подают каких-либо сигналов, которые предупреждали бы человека об опасности.

Действие электрического тока на организм человека:

Термическое действие заключается в нагреве тканей и биологических сред организма, что приводит к перегреву всего организма и, как следствие, нарушению обменных процессов.

Электролитическое воздействие заключается в разложении крови, плазмы и прочих физиологических растворов организма на составляющие, после чего они уже не могут корректно выполнять свои функции.

Биологическое воздействие связано с раздражением и возбуждением нервных волокон и других органов.

Различают два основных вида поражений электрическим током: электрические травмы и удары.

Электротравмами являются:

электрический ожог — результат теплового воздействия электрического тока в месте контакта;

электрический знак — специфическое поражение кожи, выражающееся в затвердевании и омертвении верхнего слоя;

электроофтальпия — воспаление наружных оболочек глаз из-за воздействия ультрафиолетового излучения дуги;

металлизация кожи — внедрение в кожу мельчайших частичек металла;

механические повреждения, вызванные непроизвольными сокращениями мышц под действием тока.

Электрический удар – это поражение организма электрическим током, при котором возбуждение живых тканей сопровождается судорожным сокращением мышц.

В зависимости от тяжести последствий, электроудары делят на четыре степени:

I — судорожное сокращение мышц без потери сознания;

II — судорожное сокращение мышц с потерей сознания, но с сохранившимися дыханием и работой сердца;

III — потеря сознания и нарушение сердечной деятельности или дыхания (или того и другого);

IV — состояние клинической смерти.

Основной фактор, обусловливающий ту или иную степень поражения человека, - *сила тока*. Для характеристики его воздействия на человека установлены три критерия (см. таблицу 20):

пороговый ощутимый ток — наименьшее значение тока, вызывающего ощутимые раздражения;

пороговый неотпускающий ток — значение тока, вызывающее судорожные сокращения мышц, не позволяющие пораженному освободиться от источника поражения;

пороговый фибрилляционный ток — значение тока, вызывающее фибрилляцию сердца (хаотические и разновременные сокращения волокон сердечной мышцы, полностью нарушающие её работу).

Таблица 20. Средние значения пороговых токов

Ток	Значение тока		
	порогового ощутимого, мА	порогового неотпускающего, мА	порогового фибрилляционного, мА
Переменной частотой 50 Гц	0,5 … 1,5	6 … 10	50 … 100
Постоянный	5,0 … 20	50 … 80	300

Потенциальными источниками возможного поражения электрическим током в помещении с ЭВМ могут быть поврежденные провода, разъемы, розетки и сами части ЭВМ.

Требования к помещениям для эксплуатации ПК

Площадь, отводимая на одно рабочее место с ПК, должна составлять не менее 6,0 $м^2$, а объем не менее 20,0 $м^3$. Это связано с тем, что уровень электромагнитных полей может оказаться существенно выше из-за отсутствия общего заземления и земляной шины сети питания ПК, из-за оплетающих рабочие места жгутов проводов. При расстановке компьютеров необходимо также учитывать электромагнитные поля ПК, излучаемые на соседние рабочие места.

При выполнении основной работы на ПЭВМ с монитором уровень шума не должен превышать 60 дБ. Снизить уровень шума в помещении с монитором и ПЭВМ можно использованием звукопоглощающих материалов с максимальными коэффициентами звукопоглощения в области частот 63 — 8000 Гц для отделки помещений, подтвержденных специальными акустическими расчетами.

Освещение рабочего места

К освещению помещений с ПК предъявляются высокие требования. Освещение рабочего места должно быть организовано так, чтобы полностью исключалась возможность попадания света в глаза, появление тени, неравномерность распределения яркости освещения.

Помещение с ПК оборудуют системами как естественного, так и искусственного освещения. Естественное освещение должно осуществляться через

световые проемы, ориентированные преимущественно на север и северо-восток, и обеспечивать коэффициент естественной освещенности (КЕО) не ниже 1,2% в зонах с устойчивым снежным покровом и не ниже 1,5% на остальной территории России.

Для искусственного освещения помещения с ПК используется система общего равномерного освещения. Допускается использование местного освещения, предназначенного для освещения зоны расположения документов. При этом освещенность экрана на рабочем месте с ПЭВМ должна быть 200 люкс; клавиатуры, документов и стола — 400 люкс.

Для соблюдения норм освещенности необходимо рассчитать количество приборов освещения, при котором обеспечиваются оптимальные условия работы.

Схема помещения:

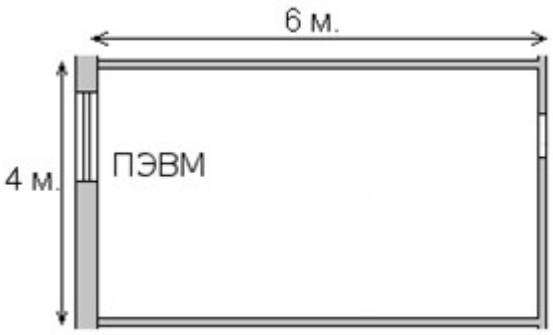

Рис. 13. Схема помещения

Исходные данные:

Ширина помещения: a = 4 м;

длина помещения: b = 6 м;

высота помещения: H = 3 м;

высота подвески светильников: H_p = 2,9 м.

Расчет:

В качестве источника освещения используются лампы накаливания. Определение количества светильников (N) общего освещения при использовании ламп накаливания:

$$N = \frac{E * S * K * Z}{F * k * n} \qquad (1),$$

где ***E*** — нормируемая освещенность, лк;

S — площадь освещаемого помещения, кв.м.;

K — коэффициент запаса;

Z — поправочный коэффициент;

F — световой поток лампы, лк;

k — коэффициент использования светового потока;

n — количество ламп в светильнике.

Площадь помещения равна: $S = 4 * 6 = 24$ кв.м.

По данным из справочной таблицы коэффициент запаса для типа помещения $K = 1,5$, а поправочный коэффициент $Z = 1,3$.

По описанию в сопроводительной документации к лампе, световой поток равен не менее 2500 лм. Количество ламп в светильнике равно 5.

Согласно справочным данным, а также согласно отраслевым нормам освещенности для помещений с ПЭВМ при работе с экраном в сочетании с работой над документами минимальная освещенность $E = 300$ лк.

Индекс помещения определяется по формуле:

$$i = \frac{a * b}{(a + b) * H_p},$$

где ***a*** — ширина помещения, м;

b — длина помещения, м;

$\boldsymbol{H_p}$ — высота подвеса светильников над рабочей поверхностью, м.

$i = 24 / ((4 + 6) * 2,9) = 0,828$

Коэффициент использования светового потока зависит от индекса помещения, типа светильника и коэффициента отражения потолка и стен. Установлено, что при коэффициенте отражения потолка 50% и стен 30% коэффициент использования светового потока при $i = 0,828$ имеет следующее значение: $k = 0,3$

Подставляя полученные данные в формулу (1) определяем:

$N = (300 * 24 * 1,5 * 1,3) / (2500 * 0,3 * 5) = 3,744$

Для хорошего уровня освещенности необходимо порядка 4 светильников.

Меры противопожарной безопасности в помещении с ПЭВМ

Как и для любого электрического устройства, для ПК велика предрасположенность к возгоранию. Высокая плотность размещения элементов электронных схем, запыленность и плохое охлаждение могут стать причиной пожара.

Напряжение к электроустановкам помещений с ПЭВМ подается по кабельным линиям, которые представляют особую пожарную опасность. Наличие горючего изоляционного материала, вероятных источников зажигания в виде электрических искр и дуг, разветвленность и труднодоступность делают кабельные линии местом наиболее вероятного возникновения и развития пожара.

Для снижения вероятности пожара при производстве компьютерной техники стараются не использовать легковоспламеняющиеся материалы.

4.3. Безопасность при чрезвычайной ситуации

В случае пожара при работе с ПЭВМ необходимо:

прекратить работу на персональном компьютере;

отключить щит электропитания;

вызвать к месту пожара начальника или администратора, вызвать пожарную помощь;

по возможности вынести легковоспламеняющиеся, взрывоопасные материалы и наиболее ценные предметы;

приступить к тушению пожара с помощью расположенных в помещении пожарных кранов и углекислотных огнетушителей ОУ-5 (расположение указано на плане эвакуации);

в случае угрозы жизни людей срочно эвакуироваться из здания.

Особое значение при пожаре имеют вопросы эвакуации людей. Для этого необходимо использовать штатные эвакуационные выходы. Эвакуационными выходами считаются дверные проемы, если они ведут из помещений либо непосредственно, либо через вестибюль. К эвакуационным путям относят такие,

которые ведут к эвакуационному выходу и обеспечивают безопасное движение людей в течение определенного времени.

Пути эвакуации:

все двери эвакуационных выходов должны свободно открываться в сторону выхода из помещений. При пребывании людей в помещении, двери могут запираться только на внутренние, легко открывающиеся запоры;

запрещается применять на путях эвакуации горючие материалы отделки, облицовки, окраски стен и потолков;

запрещается загромождать проходы, коридоры, лестничные площадки мебелью, оборудованием, а также забивать двери эвакуационных выходов;

коридоры и проходы для эвакуации, должны иметь как можно меньшую длину и минимальное количество поворотов.

Все виды путей эвакуации должны иметь естественное или искусственное освещение, работающее как от обычной электросети, так и от сети аварийного освещения.

На всех этажах здания должен быть план эвакуации, согласно которому люди должны выходить из здания при пожаре.

Эвакуация из помещения осуществляется по линиям, изображенным на плане (рис. 14).

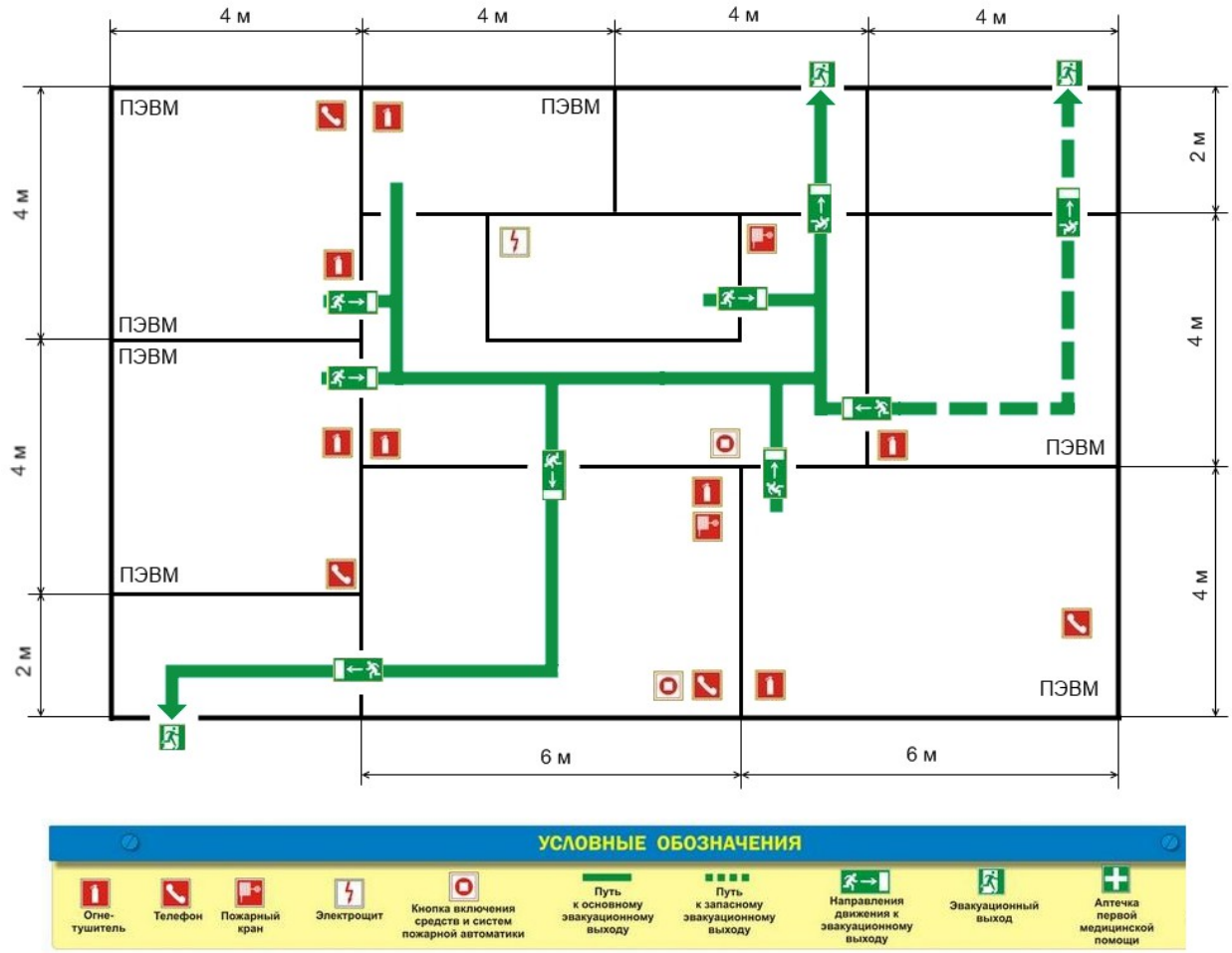

Рис. 14. План эвакуации

5. Заключение

В результате работы над данным дипломным проектом был разработан модуль хэш-алгоритма для системы моделирования клеточных автоматов Xlife, а также было произведено портирование программы в операционную систему Microsoft Windows.

Реализация хэш-алгоритма позволяет значительно ускорить расчет эволюции образцов клеточных автоматов, содержащих повторяющиеся элементы. На данный момент поддерживаются все правила с двумя состояниями вида B/S, правила поколений, мир проводов, а также т.н. табличные правила, которые задаются таблицей возможных состояний и переходов. Полученные преимущества в скорости могут быть применены в тех расчетах, где время выполнения имеет первостепенное значение: в частности, моделирование реальных процессов с целью получения практического результата, математическое моделирование, моделирование автоматов с целью выявления закономерностей.

Также в ходе данной работы было произведено портирование исходной программы в операционную систему MS Windows, что позволит увеличить количество пользователей Xlife, поскольку ОС Windows являются одними из самых распространенных, особенно среди пользователей персональных компьютеров.

Сделанные в исходной программе Xlife изменения позволяют ей сравняться с лидерами в своем классе, а по некоторым показателям даже опередить их.

6. Список литературы

1) R. Wm. Gosper. Exploiting Regularities in Large Cellular Spaces / R. Wm. Gosper – North-Holland, Amsterdam: Physica D (Elsevier), 1984.

2) Астафьев Г. Б. Клеточные автоматы; учебно-методическое пособие / Г.Б. Астафьев, А.А. Короновский, А.Е. Храмов – Саратов, 2003.

3) Tomas G. Rokicki. An Algorithm for Compressing Space and Time / Tomas G. Rokicki // Dr Dobb's Journal. – 2006. – №4.

4) Owen, Jennifer. Complexity Measures on the Game of Life / Jennifer Owen. // MSc Dissertation, University of York – 2008.

5) HashLife на коленке // Хабрахабр [Электронный ресурс]: http://habrahabr.ru/post/136616/

6) Евсютин О.О. Использование клеточных автоматов для решения задач / Евсютин О.О., С.К. Россошек. // Доклады ТУСУРа. – 2010. – №1 (21).

7) Hashlife // Википедия – свободная энциклопедия [Электронный ресурс]: http://en.wikipedia.org/wiki/Hashlife