Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение   
высшего образования

”Алтайский государственный технический университет им. И. И. Ползунова”

Факультет информационных технологий

наименование подразделения

Кафедра информатики, вычислительной техники

и информационной безопасности

наименование кафедры

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Попкова

(подпись руководителя от вуза) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

ОТЧЕТ

о преддипломной практике

(вид практики)

в ФГБОУ ВО АлтГТУ

(название предприятия, организации, учреждения)

ПрП\_09.04.01.04.000\_ОТ

(обозначение документа)

Студент группы \_\_8ИВТ-71\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С. В. Горских

(инициалы, фамилия)

Руководитель практики

от предприятия \_\_профессор каф. ИВТиИБ, д.т.н. Л.И. Сучкова

(должность, ученое звание) (инициалы, фамилия)

Руководитель практики от вуза доцент каф. ИВТиИБ, к.п.н А.И. Попкова

(должность, ученое звание) (инициалы, фамилия)

Барнаул 2019

ФГБОУ ВО «Алтайский государственный технический университет

им. И. И. Ползунова»

Кафедра «Информатика, вычислительная техника и информационная безопасность»

**Индивидуальное задание**

на производственную практику

(преддипломная практика) студенту 2 курса Горских С.В. группы 8ИВТ-71

Профильная организация:

ФГБОУ ВО «Алтайский государственный технический университет им. И. И. Ползунова»

Сроки практики: 24.05.2019 г. - 21.06.2019 г.

Тема: Разработка синтаксически-ориентированной инструментальной системы для клеточно-автоматного моделирования.

**Рабочий график (план) проведения практики:**

|  |  |  |  |
| --- | --- | --- | --- |
| **№**  **п/п** | **Содержание раздела (этапа) практики** | **Сроки выполнения** | **Планируемые результаты практики** |
| 1 | Получить индивидуальное задание на практику.  Пройти инструктаж по охране труда и технике безопасности.  Завершение написания 1 главы диссертации | 1 неделя | Итоговое формирование компетенции:  - ОК-5: использование результатов освоения фундаментальных и прикладных дисциплин магистерской программы для решения задач в своей предметной области, самостоятельное решение научных, исследовательских, проектных и технологических задач в области информатики и вычислительной техники;  - ОК-7: применить знания и умения в решении конкретных практических задач;  - ОПК-5: владение методами и средствами получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе в глобальных компьютерных сетях;  - ОПК-6: способность анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями. |
| 2 | Завершить разработку синтаксически-ориентированной инструментальной системы для клеточно-автоматного моделирования.  Выполнение тестирования и подготовка к опытной эксплуатации разработанного программно-технического обеспечения  Завершение написания 2 и 3 глав диссертации  Написание 4 главы диссертации  Анализ результатов проведенных исследований | 2-3 недели | Итоговое формирование компетенций:  - ОК-8: закрепить навыки работы на современном оборудовании и в средах разработки, применяемых для создания программно-технического обеспечения автоматизированных систем;  - ПК-7: применение перспективных методов исследования и решения профессиональных задач на основе знания мировых тенденций развития вычислительной техники и информационных технологий;  - ПК-12: способность выбирать методы и разрабатывать алгоритмы решения задач управления и проектирования объектов автоматизации. |
| 3 | Оформить и сдать отчет по практике. В отчете отразить полученные результаты | 4 неделя | Итоговое формирование компетенции  ОК-9: оформление отчета по проведенной работе и подготовка публикации по результатам исследования |

Руководитель практики от вуза \_\_\_\_\_\_\_\_\_\_\_\_\_ Попкова А.И., к.п.н., доцент каф. ИВТиИБ

(подпись) (Ф.И.О., должность)

Руководитель практики от

профильной организации Сучкова Л.И, д.т.н., профессор каф.ИВТиИБ

(подпись) (Ф.И.О., должность)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горских С.В.

(подпись) (Ф.И.О.)

**Инструктаж по ОТ, ТБ, ПБ, ПВТР**

Инструктаж обучающегося по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка проведен «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Руководитель практики от

профильной организации \_\_\_\_\_\_\_\_\_\_\_ Сучкова Л.И, д.т.н., профессор каф.ИВТиИБ

(подпись) (Ф.И.О., должность)

**Содержание**

[**Введение** 6](#_Toc12253723)

[**1** **Обзор систем клеточно-автоматного моделирования** 8](#_Toc12253724)

[**1.1** **Понятие клеточно-автоматной модели** 8](#_Toc12253725)

[1.2 Существующие системы клеточно-автоматного моделирования 17](#_Toc12253726)

[1.2. Golly 17](#_Toc12253727)

[1.2.2 CAME&L 18](#_Toc12253728)

[1.2.3 SoftCAM 20](#_Toc12253729)

[1.2.4 Система имитационного моделирования WinALT 21](#_Toc12253730)

[**2** **Проектирование синтаксически-ориентированной среды клеточного-автоматного моделирования** 23](#_Toc12253731)

[2.1 Разработка синтаксиса описания клеточно-автоматной модели 23](#_Toc12253732)

[2.1.1 Описание клеточных автоматов 23](#_Toc12253733)

[2.1.2 Системы параллельных подстановок 28](#_Toc12253734)

[2.1.3 Параллельные подстановки 29](#_Toc12253735)

[2.1.4 Контекстные подстановки 33](#_Toc12253736)

[2.1.5 Функции переходов 34](#_Toc12253737)

[2.1.6 Механизм внешних воздействий 35](#_Toc12253738)

[2.2 Проектирование среды моделирования 37](#_Toc12253739)

[2.2.1 Проектирование интерпретатора 38](#_Toc12253740)

[2.2.2 Проектирование пользовательского интерфейса 49](#_Toc12253741)

[**3** **Программная реализация синтаксически-ориентированной среды клеточно-автоматного моделирования** 50](#_Toc12253742)

[3.1 Структура интерпретатора 50](#_Toc12253743)

[3.1.1 Лексический анализатор 50](#_Toc12253744)

[3.1.2 Синтаксический анализатор 54](#_Toc12253745)

[3.1.3 Подпрограммы семантического контроля 55](#_Toc12253746)

[3.1.4 Подпрограммы интерпретации 58](#_Toc12253747)

[3.2 Пользовательский интерфейс 59](#_Toc12253748)

[**4** **Исследования поведения клеточно-автоматных моделей с «внешними воздействиями» в разработанной системе** 64](#_Toc12253749)

[**Заключение** 73](#_Toc12253750)

[**Список использованных источников** 74](#_Toc12253751)

**Введение**

Клеточно-автоматное моделирование находит себе применение во многих сферах. Исследования, основанные на моделировании с помощью клеточных автоматов, дают новые возможности для нахождения алгоритмической разрешимости тех или иных задач. Проектирование и разработка новых функциональных решений для такого моделирования является актуальным. На данный момент существует множество программных продуктов, осуществляющих моделирование с помощью клеточных автоматов. Большинство из них имеют функционал для демонстрации множества стандартных шаблонов клеточных автоматов и не предоставляют инструментарий для разработки пользовательской клеточно-автоматной модели.

В большинстве случаев состояние системы на протяжении всего процесса моделирования описывают правилами поведения, которые относятся только лишь к внутренним объектам процесса, что идеализирует модель, отбрасывая внешние факторы. В реальном мире такое поведение процесса встречается очень редко. В данной работе предложена концепция моделирования, при которой предполагается возможность описания так называемых “внешних воздействий” и правил их влияния на состояние клеток. Это позволит взглянуть на поведение модели с новой стороны, учитывая различные внешние факторы, при исследовании того или иного процесса.

Конечной целью данной работы является разработка синтаксически-ориентированнной среды клеточно-автоматного моделирования, предоставляющей новые возможности в описании моделей клеточных автоматов, главной из которых является описание поведения «внешних воздействий».

Для достижения этой цели должны быть выполнены следующие задачи:

1. Исследование существующих решений для клеточно-автоматного моделирования;
2. Проектирование и разработка собственного синтаксиса описания моделей;
3. Проектирование и разработка среды моделирования с применением разработанного языка.
4. Проведение экспериментов при моделировании клеточно-автоматных моделей с использованием «внешних воздействий»

# **Обзор систем клеточно-автоматного моделирования**

* 1. **Понятие клеточно-автоматной модели**

Клеточный автомат (КА) был предложен фон-Нейманом в середине прошлого века [1]. На основе этой модели подтверждалась мысль о том, что человек может создать устройство, обладающее присущими ему самому свойствами, в частности, способностью воспроизводить себе подобного. Поэтому КА долгое время рассматривался как модель самовоспроизведения и толковался как упрощенная модель некоторого биологического сообщества, состоящего из множества клеток. Каждой клетке ставится в соответствие конечный автомат, называемый *элементарным автоматом*, который может находиться в одном из двух состояний: 0 и 1 (черное или белое) и изменять это состояние 0 — 1 или 1 — 0 в зависимости от состояний клеток некоторого своего окружения, называемого *соседством*. Алгоритм вычисления следующего состояния в зависимости от состояний соседей (функция переходов элементарного автомата) у всех клеток одинаковый. Все клетки выполняют переход в новое состояние одновременно, т.е. синхронно и параллельно. При этом наблюдается изменение *глобальной* черно-белой картины распределения состояний по пространству КА. Такая картина называется *конфигурацией* КА.

В более строгом виде основные свойства КА формулируются в следующем виде [2]:

* Дискретность пространства, времени и состояний.
* Однородность (все клетки организованы в регулярную пространственную структуру).
* Синхронный режим изменения состояний.
* Пространственная локальность (аргументами функции переходов каждого элементарного автомата являются состояния элементарных автоматов из его ограниченной окрестности).
* Временная локальность (аргументами функции переходов являются состояния соседних автоматов только на ограниченном числе предыдущих итераций).

Итеративная смена конфигурации при переходах всех элементарных состояний в новые состояния называется *эволюцией* КА. Эволюционируя КА моделирует пространственную динамику, которая может иметь завершение, повторяться периодически или изменяться хаотично. Исследования этой "модели мира" показали, что несмотря на простоту каждой клетки, их кооперативная работа моделирует очень сложные и разнообразные процессы, которые иногда невозможно (или по крайней мере, неизвестно как) описать другим способом. В таком понимании КА теперь называется *классическим* КА.

Первое время КА оставался объектом для игры ума. Сначала эта игра происходила на бумаге. Потом она была перенесена на компьютер. Самой известной из таких "игрушек" для начинающих программистов до сих пор является так называемая "Игра Жизнь" [3]. Клеточный автомат, клетки которого расположены в ячейках двумерной декартовой решетки и имеют только по восемь соседей, может иметь бесконечное множество эволюций, воспроизводящих разные картины: движение пули по какой-либо траектории, распространение волны, раскрутку спирали, рисовать причудливые орнаменты и т.д., то есть строить самые различные черно-белые меняющиеся во времени образы. Задача в этой игре состоит в том, чтобы найти такую начальную конфигурацию, при которой КА, эволюционировал бы желаемым образом. Все попытки решить эту задачу формально не увенчались успехом. Только опыт, навыки и терпение при испытаниях множества проб могут помочь в этом деле.

Большое разнообразие модификаций КА, предназначенных для моделирования пространственной динамики привело к появлению понятия мелкозернистого параллелизма, который объединил в себе все модели вычислений, обладающие двумя главными свойствами КА:

1. Итерационный процесс вычисления глобальных конфигураций при неограниченном виртуальном параллелизме вычисления новых состояний клеток.
2. Локальность межклеточных взаимодействий, когда каждая клетка вычисляет функцию перехода, используя в качестве переменных состояния ограниченное число близлежащих клеток.

При этом на алфавит состояний клеток не накладывается ограничений. Он может быть булевым, целочисленным, вещественным и символьным. Более того, мелкозернистый параллелизм допускает не только синхронный режим смены состояний клеток, но и асинхронный. Асинхронный режим предполагает, что смена состояний клеток происходит не "сразу все" по тактовому сигналу, а в любом порядке. Этот порядок может быть случайным или заранее заданным. Для моделирования пространственной динамики во многих случаях используются вероятностные КА, а также КА с самыми разными функциями переходов. Такое расширенное понятие КА-моделирования включает в себя также явные численные методы решения дифференциальных уравнений в частных производных, которые иногда рассматриваются как "непрерывные КА".

Режим работы КА определяет порядок применения локальных операторов к клеткам массива за время одной итерации. Основными режимами являются: синхронный и асинхронный. Названные два режима являются двумя крайними случаями. Существуют также смешанные и производные режимы, например, блочно-синхронный и асинхронный упорядоченный [3].

При синхронном режиме изменение состояний происходят только после того, как все функции fk для всех клеток вычислены, т.е. известны новые состояния всех клеток массива. В машине САМ-8, которая является аппаратурным воплощением КА, все клетки выполняют вычисления функций параллельно и синхронно меняют свои состояния, реализуя потенциальный внутренний параллелизм КА, далее называемый клеточным параллелизмом, при котором итерация равна одному такту.

В обычном последовательном компьютере синхронная параллельная смена состояний клеток имитируется путем вычисления новых состояний всех клеток последовательно при сохранении их в другом массиве, который служит глобальным состоянием на следующей итерации. Такая параллельность (одновременность) называется виртуальной.

При асинхронном режиме никакой одновременности смены состояний клеток не предполагается. Внутренний параллелизм проявляется в том, что смены состояний клеток происходят в произвольном порядке (так называемый "истинный параллелизм" или "true concurrency"). Каждая клетка вычисляет функцию перехода от тех состояний соседей, которые на данный момент имеют место, и сразу производит замену текущего состояния на новое. Это значит, что функции переходов fk могут иметь в качестве аргументов как текущие, так и новые состояния. Аппаратные реализации, позволяющие реализовать истинную параллельность асинхронных КА, неизвестны. В последовательном компьютере каждая итерация выполняется за |M| тактов. Множество возможных последовательностей, к которым применяется правила перехода для перехода в новое глобальное состояние, называются трассами.

Клеткой называется пара (x, m) A х М , где х А - символ алфавита A, обозначающий состояние клетки, а m М - имя клетки из множества имен М, обозначающее местоположение клетки в дискретном пространстве. Алфавит может быть булевым, Ab = {0,1}, вещественным, Ar = [0,1], и символьным Аs = {а,b, с, d,....}. Состояния могут быть заданы переменными, определенными на одном из этих алфавитов.

Множество имен может быть любым (не более чем счетным) множеством. При моделировании пространственной динамики его элементами обычно являются наборы пространственных координат точек дискретного пространства конечных размеров.

Множество клеток = {(х,m)}, в котором нет клеток с одинаковыми именами, называется *клеточным массивом*. Клеточные массивы, в которых состояния определены на множестве A, а имена на множестве М, составляют класс клеточных массивов, обозначаемый (A, M). На множестве имен определены именующие функции : М —> А, которые определяют пары взаимодействующих клеток.

Конечное множество именующих функций называется *шаблонам соседства*.

Т{m) = { (m), (m),..., (m)}.

Для каждой клетки m М шаблон Т{m) определяет множество ее соседей. Нумерация клеток в Т(m) может быть любой. При этом принято считать, что (m) = m.

Выражение вида

S(m) = {( , (m)), ( , (m)),..., ( , (m))}

называется *локальной конфигурацией*, если существует такое r, что расстояние от m до любого (m) Т(m), d(m, (m)) < r, где Т(m) -множество входящих в S(m) именующих функций, называемое определяющим шаблоном локальной конфигурации.

*Параллельной подстановкой* называется выражение вида

(m) : S(m) \* S"{m) -> S'(m),

где S(m) = {( , (m)),...,( , (m))},

S'(m) = {( , (m)),...,( , (m))},

S"{m) = {( , (m)),..., ( , (m))}.

Результат применения локального оператора ко всем клеткам массива Q называется *глобальным оператором* ( ).

Существует два основных режима применения глобального оператора вычисления следующего глобального состояния (t + 1): синхронный и асинхронный.

Синхронный режим предполагает, что аргументы функции переходов - это состояния клеток-соседей на текущей итерации t. На каждой итерации клетки вычисляют значения нового состояния и, затем, все клетки вместе синхронно заменяют старые состояния на новые. Результатом применения параллельной подстановки к клеточному массиву может быть представлен как вычитание из базовых клеток всех применимых подстановок и добавление их правых частей.

При асинхронном режиме каждая клетка вычисляет функцию перехода от тех значений состояний соседей, которые на данный момент имеют место и сразу меняет свое состояние. Таким образом, каждая итерация КА состоит из | | шагов, на которых только одна клетка меняет свое состояние.

Конечное множество параллельных подстановок Ф = { 1, .. , n}, определенных на одном и том же классе клеточных массивов (A, M), называется *системой параллельных подстановок*. Система параллельных подстановок Ф применима к клеточному массиву , если хотя бы одна из подстановок Ф к нему применима. Одной итерацией эволюции системы Ф к является применение всех его подстановок ко всем клеткам массива.

Непротиворечивая система подстановок Ф = { 1,..., n} совместно с режимом их применения, который может задаваться с помощью контекстов второго рода, называется *Алгоритмом Параллельных Подстановок* (АПП).

Любой клеточный автомат и все его известные модификации могут быть описаны в терминах АПП, поскольку эта формальная система обладает вычислительной универсальностью. Кроме того, для конструирования сложных клеточно-автоматных моделей очень важно наличие такого средства как контекста второго рода.

Разделение левой части параллельной подстановки на две части: базу и контекст, сделано для того, чтобы при применении ее к любой конкретной клетке М отделить множества клеток S( ), состояния которых она изменяет, от множества клеток S"( ), состояния которых она не меняет. При этом состояния клеток из контекста S"( ) могут быть изменены на той же итерации применением этой же подстановки к другой клетке , если S"( ) S( ) 0 или другой подстановкой ' из системы Ф = { , '}.

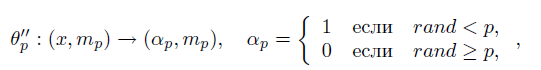
Контексты параллельных подстановок играют двоякую роль, различающуюся своим влиянием на эволюцию. Соответственно, они подразделяются на контексты первого рода и контексты второго рода.

* Контексты первого рода содержат клетки вида (, (m)), в которых (m) определены на базовом множестве имен, т.е. в области эволюции КА. Такие контексты характерны для классических КА;
* Контексты второго рода выполняют управляющие функции. Локальная конфигурация контекста второго рода имеет вид

где  - функция, значения которой играют роль предиката, делающего подстановку применимой. Для вычисления значений функции необходимы дополнительные контекстные подстановки ", определенные на " (A" х М"). Переменные могут быть внешними переменными, например, случайными числами от генератора случайных чисел, или значениями тактирующих импульсов, или указателями области значений имен клеток и др.

Следующие контексты второго рода считаются типовыми:

* Вероятностный контекст , применяемый в вероятностных КА. Его контекстная подстановка имеет вид:



где  М", rand - случайное число в интервале [0,1], р - вероятность применимости подстановки.

* Тактирующий контекст применяется для организации n-тактного синхронного режима применения подстановок.



где М".

* Пространственный контекст  применяется для выделения тех областей клеточного массива, где подстановка должна быть применима.

Параллельная подстановка , определенная на  (A,M), называется непротиворечивой, если результат ее  () над любым  (A,М) является клеточным массивом, т.е. не содержит двух разных клеток с одним и тем же именем. Проще говоря, непротиворечивая подстановка при одновременном применении в разных клетках никогда не пытается записать разные состояния в клетку с одним и тем же именем, т.е.



Ясно, что если правая часть подстановки содержит только одну клетку, как это имеет место у классического КА, то она всегда непротиворечива. Если же |S'(m)| > 1, то существует опасность коллизии при синхронном режиме.

Установить заранее противоречивость очень сложно, поэтому, обычно удовлетворяются достаточным условием непротиворечивости, которое для одной подстановки имеет следующий вид



где Т'() и T'() - шаблоны локальной конфигурации S'(m) при m = и m =, соответственно.

Противоречивая подстановка не может выполняться синхронно. Однако, ее можно заменить на систему непротиворечивых подстановок, применимых на разных подтактах итерации, использовав тактирующий контекст. При асинхронном режиме коллизии невозможны, так как никакие две клетки одновременно не меняют своих состояний. Представленные одной и той же системой параллельных подстановок КА при синхронном, многотактном синхронном и асинхронном режимах функционирования имеют различные эволюции и, следовательно, не могут быть моделями одного и того же явления.

* 1. Существующие системы клеточно-автоматного моделирования
     1. Golly

Golly позиционируется как кросплатформенное приложение с открытым исходным кодом для изучения различных типов клеточных автоматов [4]. Основными авторами явлются Эндрю Треворроу и Том Рокицки, авторами кода – Тим Хаттон, Дейв Грин, Джейсон Саммерс.

Ключевые особенности Golly:

* Поддерживает ограниченные и неограниченные вселенные с ячейками до 256 состояний;
* Поддерживает несколько алгоритмов, в том числе сверхбыстрый (по мнению авторов) алгоритм Билла Госпера;
* Большая библиотека готовых моделей клеточных автоматов;
* Интерфейс для создания собственных правил перехода – ключевая особенность в рамках исследований.

Правила перехода в случае моделирования игры «Жизнь» должны соответствовать следующему шаблону:

*R r , C c , M m , S smin .. smax , B bmin .. bmax , N n*

*R r* - указывает диапазон соседства ( r от 1 до 500);

*C c* - указывает количество состояний ( c от 0 до 255);

*M m* - указывает, включена ли средняя ячейка в шаблон соседства ( m равно 0 или 1);

*S smin .. smax* - определяет предельные значения для живой ячейки, необходимые для выживания в следующей итерации;

*B bmin .. bmax* - определяет предельные значения для мертвой клетки, необходимые для возрождения в следующей итерации;

*N n* - указывает расширенный тип окрестности ( n - M для Мура, N для фон Неймана и C для циркуляра).

Недостатком с точки зрения синтаксической составляющией данной системы является функциональная ограниченность в описании клеточно-автоматных моделей. Перечисление клеток, учавствующих в правиле перехода, или другии словами тип окрестности осуществляется из набора заданных типов, таких как окрестность Мура, фон Неймана, циркуляра и т.д. Также стоит понимать, что ключевые слова синтаксиса разработаны для описания моделей основных уже существующих клеточных автоматов, что исключает возможность для описания сколь угодно различных моделей.

На рисунке 1.1 представлен интерфейс системы Golly.

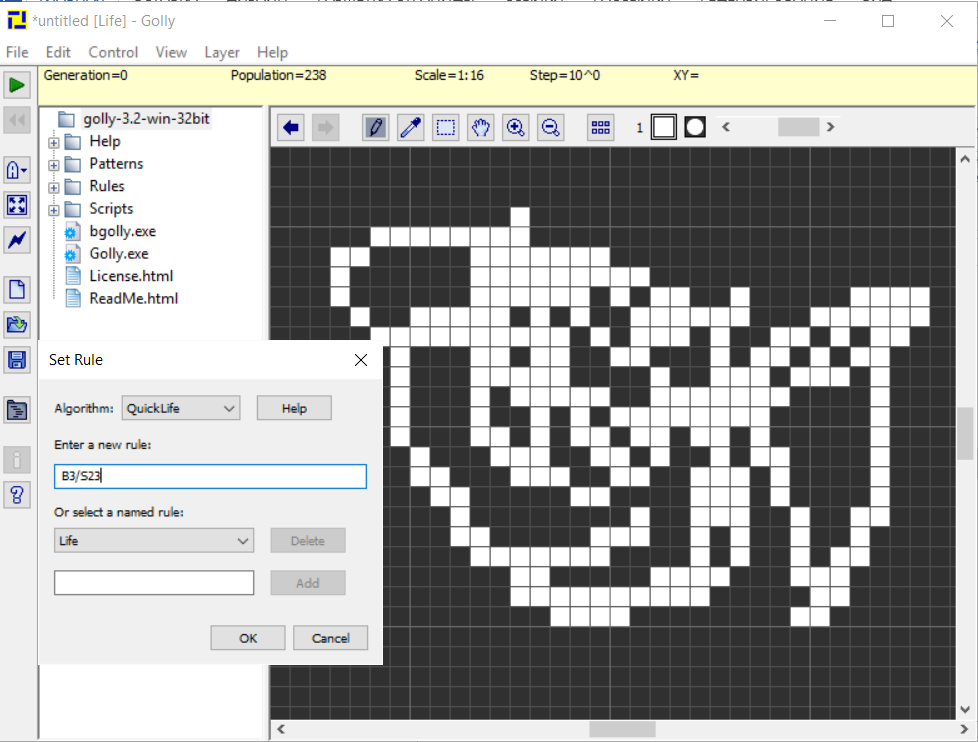


Рисунок 1.1 – интерфейс Golly

* + 1. CAME&L

Л. Наумов и А. Шалыто разработали собственную среду моделирования КА, называемую CAM&L [17]. Данная среда разрабатывалась, как простое, расширяемое рабочее пространство для сложных клеточных вычислений. На рисунке 1 представлен интерфейс приложения.

В данном приложении функциональность моделируемого автомата реализуется набором взаимодействующих компонентов.

* Grid – визуализация состояния решетки;
* Metrics ­– определяет координаты клеток, расстояние между ними и отношение соседства;
* Data – обеспечивает хранение состояния решетки в памяти и сохранение его в файл;
* Rules – реализует правила автомата и другие преобразования состояния решетки.

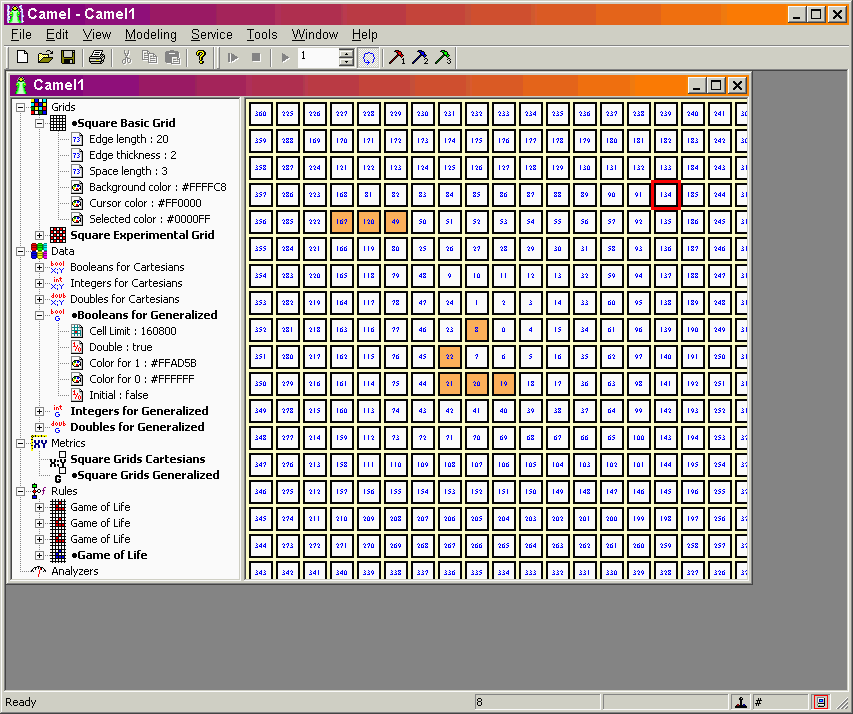


Рисунок 1.2 – Интерфейс CAM&L

Достоинства данного решения:

* Обширный набор правил, позволяющих строить разнообразные модели клеточных автоматов;
* Богатый инструментарий для наблюдения за экспериментом;
* Сохранение состояния эксперимента в XML-файл.

Недостатки:

* Достаточно старая реализация с неэффективными алгоритмами вычислений;
* Интерфейс, не соответствующий современным требованиям;
* Достаточно сложный язык описания модели КА;
  + 1. SoftCAM

И. Матюшкин предложил свой вариант программы для моделирования КА, называемый SoftCAM [18]. На рисунке 2 представлена её архитектура в виде UML-диаграммы.

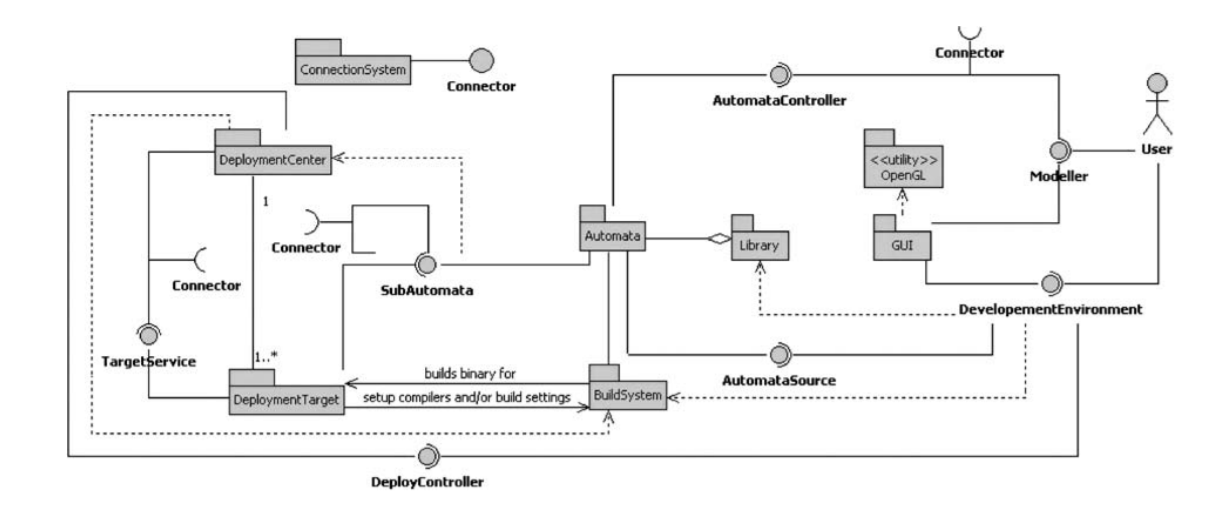


Рисунок 1.3 – Архитектура SoftCAM

Данное решение расширяет функционал моделирования такой МКА как Golly. Сделан упор на дружественность и открытость к пользователю, удобный и понятный интерфейс, легкий язык моделирования КА. К недостаткам относится низкое быстродействие, являющееся результатом выбора скриптового языка Python как средство разработки. Легкий язык моделирования КА является недостатком в случае проектирования сложных автоматов, когда код становится громоздким. Данная проблема частично решается созданием готовых шаблонов моделирования и библиотек.

* + 1. Система имитационного моделирования WinALT

Программный комплекс WinALT, разработанный на кафедре параллельных вычислений НГУ и в Институте вычислительной математики и математики геофизики СО РАН, обеспечивает конструирование мелкозернистых параллельных алгоритмов (МЗП) и структур в самой широкой трактовке термина. Данная система базируется на алгоритме параллельных подстановок – алгоритмической системе, являющейся пространственной моделью для представления параллельных мелкозернистых алгоритмов и структур [19]. Алгоритм параллельных подстановок (АПП) концептуально объединяет в себе подстановочный характер алгоритма Маркова с пространственной параллельностью клеточного автомата и основывается на общем для них ассоциативном механизме выполнения операций [20]. АПП позволяет описать вычисления в максимально параллельном виде, когда на каждом шаге выполняются все допустимые действия над всеми имеющимися данными. На рисунке 1.3 представлен графический интерфейс системы.

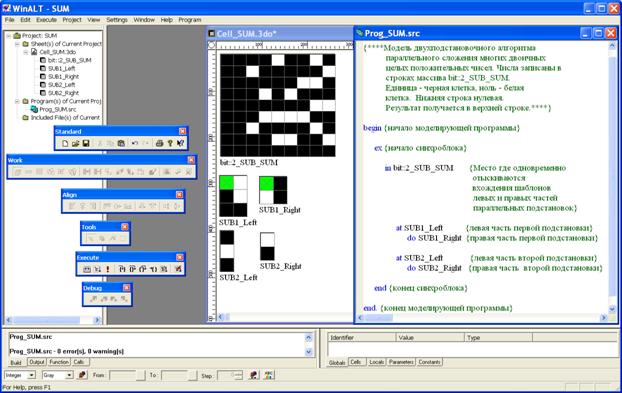


Рисунок 1.3 – Графическая среда системы WinALT

Ключевые достоинства WinALT:

* Возможность конструирования алгоритмов и структур с разнообразными видами МЗП;
* Полностью открытая система;
* Конструирование и модификация как данных модели, так и моделирующих программ, позволяет следить за динамикой изменения преобразуемых в модели данных;
* Многофункциональный графический интерфейс;
* Работа с объектами данных больших размеров.

К недостаткам можно отнести язык для описания работы клеточно-автоматных моделей, используемый в рассмотренной среде WinALT, имеющий не тривиальный синтаксис и не предоставляющий возможности использовать весь потенциал клеточных автоматов.

# **Проектирование синтаксически-ориентированной среды клеточного-автоматного моделирования**

Среда клеточно-автоматного моделирования состоит из следующих основных компонентов:

* Клиентский интерфейс для описания клеточно-автоматных моделей на спроектированном синтаксисе;
* Модуль визуализации процесса моделирования;
* Интерпретатор для спроектированного языка;
  1. Разработка синтаксиса описания клеточно-автоматной модели
     1. Описание клеточных автоматов

Предполагаемый язык является Си-подобным интерпретируемым языком и основан на математической модели «Алгоритм Параллельных Подстановок» АПП, позволяющий описывать формально КА-модели. Условно, синтаксис языка можно разделить на две составляющие – синтаксические конструкции языка Си и синтаксис описаний КА, основанный на модели АПП.

Текст программы пользователя может состоять из объявлений или определений данных, описаний функций, описаний клеточных автоматов. Также допускаются описания пользовательских типов, таких как структуры. В качестве данных могут выступать переменные, константы, статические и динамические массивы, которые могут быть стандартного или пользовательского типа, а также специализированные КА-типы.

Язык допускает описание идентификаторов, которые могут состоять из букв английского алфавита, цифр, символов подчеркивания и всегда начинаются с буквы. Комментарии могут быть однострочными и многострочными.

Типы данных, поддержваемые данным языком представлены в таблице 2.

Таблица 2.1 – Типы данных

| **Тип** | **Краткое описание** |
| --- | --- |
| Стандартные типы данных | |
| int | Целый (4 байта) |
| long | Целый (8 байт ) |
| bool | Булев (1 байт) |
| double | Вещественный с двойной точностью (8 байт) |
| void | - |
| string | Строковый (переменный размер) |
| Специализированные типы данных | |
| cell | Указатель на клетку (4 байта) |
| classArray | Клеточный массив (переменный размер) |
| neig | Шаблон соседства (переменный размер) |

В таблице 2.2 приведен список поддерживаемых операторов языка Си. Помимо представленных операторов язык содержит и некоторые собственные операторы.

Таблица 2.2 – Список поддерживаемых операторов языка Си

|  |  |
| --- | --- |
| **Оператор** | **Краткое описание** |
| [выражение]; | Оператор «выражение» |
| while | Цикл с предусловием |
| for | Цикл с параметром (for) |
| if/else | Условный оператор if |
| break | Немедленное завершение цикла |
| continue | Переход к следующей итерации цикла |
| return | Возврат из функции |

В таблице 2.3 представлен список поддерживаемых унарных, бинарных и тернарных операций языка Си.

Таблица 2.3 – Список поддерживаемых операций языка Си

| **Операция** | **Краткое описание** |
| --- | --- |
| Унарные операции | |
| ++ | Инкремент |
| -- | Декремент |
| ! | Логическое отрицание |
| + | Унарный плюс |
| - | Унарный минус (арифметическое отрицание) |
| new | Выделение памяти |
| delete | Освобождение памяти |
| Бинарные и тернарные операции | |
| \* | Умножение |
| / | Деление |
| % | Остаток от деления |
| + | Сложение |
| - | Вычитание |
| < | Меньше |
| > | Больше |
| == | Равно |
| <= | Меньше или равно |
| >= | Больше или равно |
| != | Не равно |
| && | Логическое И |
| || | Логическое ИЛИ |
| = | Присваивание |
| . | Операция доступа |

Рассмотрим более подробнее реализацию функциональных возможностей языка по описанию клеточных автоматов.

Для клеточного автомата требуется клеточное пространство, на котором он моделируется. Клеточные пространства описываются с помощью специальных классов клеточных массивов, имеющих тип classArray. Ниже представлен синтаксис описания класса клеточных массивов:

*classArray имя <тип\_решетки, форма\_пространства>*

*{*

*поля\_данных*

*шаблоны соседства*

*};*

*тип\_решетки* – структура решетки, на которой описывается класс. Тип решетки задает количество измерений пространства и структуру соседства;

*форма\_пространства* – задает граничные условия. К примеру, клеточное пространство может задаваться формой тора;

*поля\_данных* – элементы данных, расположенные в узлах решетки. Другими словами, это описание переменных (частный случай), которые будут хранить значения клеток, связанных с ними;

*шаблоны соседства* – объявление шаблона соседства, определяющего набор соседних клеток, которые будут задействованы в вычислении очередного значения клетки.

Ниже представлен листинг кода описания класса клеточного массива:

*classArray SimpleClassArray <CA.Lattice.HEXAGONAL\_2D,CA.Surface.FLAT>*

*{*

*bool isAlive;*

*neig template = {#[0], #[1], #[2], #[1, 2], #[1, 3]};*

*};*

В классе SimpleClassArray, который представляет двумерную квадратную сетку с граничными условиями типа FLAT ( противоположные грани соединяются друг с другом), определено поле клетки типа bool, хранящее определенную информацию, а также шаблон соседства template, состоящий из пяти элементов, причем первый элемент указывает на центровую клетку, а последний указывает на соседа через одну клетку (сосед соседа).

Для создания клеточного массива определенного класса клеточных массивов используется следующий синтаксис:

*имя\_класса имя = new имя\_класса(размеры\_клеточного\_массива);*

Где *имя\_класса* – имя класса клеточных массивов, описанного пользователем, *размеры\_клеточного\_массива* – численное значение размеров измерений создаваемого клеточного массива.

Ниже представлен пример кода объявления и инициализации клеточного массива на 2D пространстве с размером 500 на 500 клеток:

*SimpleClassArray simpleArray = new SimpleClassArray(500, 500);*

Синтаксис описания непосредственно клеточного автомата представляет собой описание функции, которая по заданным правилам преобразует данные в указанных клеточных массивах. Она содержит в себе описание систем параллельных подстановок, а также может содержать описание контекстных подстановок и функций переходов. Клеточный автомат может принимать параметры, а также возвращает значение типа integer ­– число завершенных глобальных итераций эволюции. Описание клеточного автомата должно содержать, по крайне мере, одну систему подстановок, которая всегда идет в описание первой. Клеточные автоматы работают только с теми клеточными массивами, которые были переданы в функцию в качестве параметра. Ниже представлен синтаксис описания клеточного автомата:

*CellAuto имя (список\_параметров)*

*{*

*вектор\_внешних\_воздействий*

*системы\_подстановок*

*[контекстные\_подстановки]*

*[функции\_переходов]*

*}*

*список\_параметров* – список клеточных массивов, с которыми будет работать клеточный автомат;

*системы\_подстановок* – системы параллельных подстановок. См. «2.1.2 Системы параллельных подстановок»;

*контекстные подстановки* – см. «2.1.3 Контекстные подстановки»;

*функции\_переходов* – см. «2.1.4 Функции переходов»;

*вектор\_внешних\_воздействий* – см. «2.1.5 Внешние воздействия»

Ниже представлен листинг кода, демонстрирующий описание клеточного автомата с именем SimpleCAЮ, который имеет единственный параметр – клеточный массив класса SimpleClassArray. Клеточный автомат состоит из системы параллельных подстановок S1 с единственной подстановкой PP1, при помощи которой каждая клетка массива SCA сможет обменяться состоянием (переменной isAlive) с левой клеткой-соседом.

*CellAuto SimpleCA(SimpleClassArray SCA)*

*{*

*System : S1*

*{*

*PP1: (var x1, SCA.isAlive), (var x2, SCA.template[0].isAlive)* ->

*(x2, SCA.isAlive), (x1, SCA.template[0].isAlive);*

*}*

*}*

* + 1. Системы параллельных подстановок

Система параллельных подстановок представляет собой множество правил переходов клеточного автомата и может состоять из одной или нескольких параллельных подстановок. Подстановки из одной системы постановок применяются последовательно к каждой клетке указанного клеточного множества и, в конечном счете, после применения всех подстановок ко всем клеткам, образуют выполнения глобального оператора.

Каждая система подстановок допускает описание подстановок, которые могут применяться к клеточным массивам различных классов. При этом стоит учитывать, что система подстановок может выполняться только на единственной области значений имен клеток, которая указывается при объявлении системы подстановок. По этой причине, клеточные массивы, к которым применяются подстановки из одной системы параллельных подстановок, должны быть одной мерности и одного размера (это не относится к контексту). В противном случае, корректная работа клеточного автомата не гарантирована, и может привести к критическим ошибкам памяти. Ниже представлен синтаксис описания системы параллельных подстановок:

System [имя\_системы] : указатель\_области\_значений [определяющий\_шаблон]

{

параллельные\_подстановки

}

Рассмотрим составные части подробнее:

*имя\_системы* – имя описываемой системы параллельных подстановок, которое допускается не указывать, если оно не требуется;

*указатель\_области\_значений* – указывает область значений имен клеток, которая будет использоваться в подстановках данной системы. В качестве указателя может выступать клеточный массив или клеточная область.

*параллельные\_подстановки* – список параллельных подстановок системы (см. «2.1.3 Параллельные подстановки»);

*определяющий шаблон* – необязательный параметр, который используется при задании блочно-синхронных клеточных автоматов.

* + 1. Параллельные подстановки

Каждая параллельная подстановка состоит из базы и правой части подстановки, а также может содержать контекст. В свою очередь база, правая часть подстановки и контекст состоят из описаний клеток базы, правой части и контекста соответственно.

Подстановки также могут выполняться в циклах, при этом все итерации цикла выполняются на одном подтакте последовательно и не считаются по отношению друг к другу параллельными. Контекст подстановки вычисляется на каждой итерации цикла. Следует также помнить, что при синхронном режиме работы клеточного автомата, значения клеток будет изменено только после применения всей системы подстановок ко всем клеткам. По этой причине на каждой итерации цикла клетки будут иметь свое первоначальное значение.

Каждая подстановка заканчиваются точкой с запятой и должна быть описана раньше функций переходов и контекстных подстановок, применяемых в данной подстановке. Контекст в подстановках выполняется в первую очередь. Если хоть один предикат в контексте ложен, то данная подстановка не будет выполнена. Ниже представлен синтаксис описания параллельной подстановки:

имя: база [ & контекст ] –> правая\_часть [~ циклы]

Рассмотрим подробнее описание параллельной подстановки:

*база* – определяет локальную конфигурацию, клетки которой учавствуют в подстановке. Локальная конфигурация задается клетками базы, перечисленными через запятую;

*контекст* – обычно исполняет роль предиката, но также может быть использован для вычисления конкретного индекса из области значений индексов клеток (пространственный контекст), или играть роль вероятностного срабатывания подстановки (вероятностный контекст). Контекстом могут быть как контекстные клетки (вызовы контекстных подстановок), так и клетки клеточных массивов, которые были переданы в клеточный автомат в качестве параметров. Элементы контекста разделяются запятыми;

*правая\_часть* – должна соответствовать локальной конфигурации базы.

Синтаксис описания клетки базы представлен ниже:

([ var ] имя, поле\_клетки\_базы)

*поле\_клетки\_базы –* указатель на поле клеточного массива. Нельзя использовать клетки клеточного массива, который не был явно указан в принимаемых клеточным автоматом параметрах.

*[var] имя* – указание модификатора var при описании означает, что будет объявлена локальная переменная с именем имя, тип которой выводится в соответствии с типом указанного поля клетки. При выполнении подстановки данная переменная будет содержать значение соответствующего поля. Если ключевое слово var отсутствует , то клетка базы играет роль предиката, а в качестве имени можно указать выражение, локально описанную функцию, константу или переменную, значения которых будут сопоставляться со значением исползуемого поля клетки. Предикаты базы проверяются после анализа контекста, и в случае, если хоть один из них ложен – подстановка не будет выполнена.

Синтаксис описания клетки контекста:

([ var ] имя, поле\_клетки\_контекста)

Рассмотрим составные части описания клетки контекста подробней:

*поле\_клетки\_контекста –* поле клетки клеточного массива, имя переменной, либо имя контекстной подстановки (см «2.1.4 Контекстные подстановки»). Нельзя указывать клетки клеточного массива, который не был явно передан в клеточный автомат в качестве параметра. В отличие от базы, здесь не допускается указывать поле клетки, которое является структурой или массивом (а не их конкретным элементом);

*[ var ] имя –* аналогично описанию базы.

Синтаксис описание клетки правой части:

*(значение, поле\_клетки\_правой\_части)*

Рассмотрим составные части описания клетки правой части подробнее:

поле\_клетки\_правой\_части – локальная конфигурация правой части подстановки должна полностью соответствовать локальной конфигурации базы;

значение – переменная, значение которой при выполнении подстановки будет присвоено указанному полю клетки. Также допускается указывать выражения или вызовы функций переходов. Тип поля клетки и тип значения должны совпадать. В отличие от базы и контекста, здесь не допускается указывать модификатор var. Если полем клетки является массив или структура (а не их конкретные элементы), то в качестве значения может быть только функция переходов, которая возвращает значение типа void. В этом случае изменять поле клетки следует непосредственно в теле указанной функции переходов, корректность чего полностью возлагается на плечи пользователя.

Синтаксис описания циклов:

*for (инициализация; выражение; модификация)*

Описание циклов аналогично описанию циклов for в языке С/С++. Указание оператора цикла означает, что подстановка будет выполняться в цикле по всем правилам языка С/С++. Подстановка может иметь несколько операторов цикла, разделенных запятыми. В этом случае циклы стоящие правее являются вложенными по отношению циклов стоящих левее.

Ниже представлен листинг кода, демонстрирующий описание параллельной подстановки с циклом:

CellAuto SimpleAuto(SimpleClassArray ARR, SimpleClassArray OTH)

{

System: sys

{

PP1: (var x1, ARR.#[i].f), (var y1, OTH.#[i].v) &

(true, cnt) –>

(y1, ARR.#[i].f), (x1, OTH.#[i].v) ~

for(int i = 0; i<8; i++);

}

(bool, cnt)

{

if (randf() > 0.5) return true;

else return false;

}

}

Подстановка с именем PP1 имеет базу, локальная конфигурация которой состоит из двух клеток, а также контекст (в виде вызова контекстной подстановки cnt), который выполняет роль предиката. Подстановка выполняется в цикле, который позволяет клеткам из массивов ARR и OTH обменяться значениями клеток из своих локальных конфигураций.

* + 1. Контекстные подстановки

Ранее уже было отмечено, контекст в подстановках может играть роль предигата, служить для вероятностного срабатывания подстановки, а также использоваться для вычислений имен клеток из области значений. Для того чтобы не усложнять этими операциями тело контекста, вычисления выносятся в контекстные подстановки, которые аналогичны функциям, за исключением того, что они не могут принимать параметров. Контекстые подстановки, как и обычные функции, должны возвращать значение (за исключением типа void), но в физическом смысле не явлются функциями, а их код непосредственно интерпретируется в точке вызова. Также в контекстные подстановки скрыто передается указатель на текущую клетку клеточной области, что дает возможность неявного обращения к данным. Вызов контекстной подстановки допускается до ее описания, но в любом случае она должна быть описана в теле клеточного автомата.

Синтаксис описания контекстной подстановки:

(тип, имя)

{

тело\_контекстной\_клетки

}

тип – тип возвращаемого контекстной подстановкой значения. В качестве типа могут использоваться любые стандартные типы (в том числе и void), кроме массивов и структур. Тип в описании контекстной подстановки, и тип переменной, которая принимает значение этой подстановки в контексте, должны совпадать. Это относится и к случаю, когда контекст играет роль предиката – контекстная подстановка и выражение в контексте должны быть одного типа.

тело\_контекстной\_клетки – аналогично телу обычной функции языка С/С++.

Ниже представлен листинг кода, демонстрирующий применение контекстной подстановки:

CellAuto NDiff(SimpleClassArray A)

{

System: A

{

Q1: (var x1, A.f), ( var x2, A.T[k].f) &

(var k, calc) –>

(x2, A.f), (x1, A.T.[k].f);

}

(int, calc)

{

if (0 < randf() <= 0.25) return 1;

else if (0.25 < randf() <= 0.5) return 2;

else if (0.5 < randf() <= 0.75) return 3;

else if (0.75 < randf() <= 1) return 4;

else return 0;

}

}

* + 1. Функции переходов

Функции переходов используются в правой части подстановок для вычисления новых значений клеток. Отличие функций переходов от обычных функций заключается в том, что в функцию переходов скрыто передается указатель на текущую клетку клеточной области, что позволяет обращаться к клеткам неявным способом.

Синтаксис описания функции переходов:

тип\_возвращаемого\_значения имя ([ параметры ])

{

тело\_функции

}

Ниже представлен листинг кода, демонстрирующий применение функции переходов:

CellAuto Live(TArray1 A)

{

System: A

{

Q1: ($, A.f) –>(calc(), A.f);

}

bool calc()

{

int sum = 0;

for (int i = 1; i < A.T.Size; i++) sum += A.T[i].f;

if (A.f == true)

if (sum == 2 || sum == 3) return true;

else return false;

else

if (sum == 3) return true;

else return false;

}

}

Пример описания клеточного автомата, реализующего игру Конуэна «Жизнь». Функция переходов рассчитывает новое состояние, которое будет присвоено клетке A.f.

* + 1. Механизм внешних воздействий

Предполагаемая возможность описания так называемых “внешних воздействий” и правил их влияния на состояние клеток достигается с помощью расширения функционала описания модели клеточного автомата. Основным объектом модернизации является описание клеточного автомата синтаксической конструкцией <CellAuto>, внутри которой разрешается объявлять переменные и описывать состояния, которые не являются частью информации клеточного пространства, но могут использоваться в вычислениях. Данные переменные или состояния, в дальнейшем будем их называть «параметры внешних воздействий», являются некими ключами управления над подстановками, описывающими поведение «внешних воздействий».

Параметры внешних воздействий могут объявляться как обычные переменные , так и как массивы.

Поведение внешнего воздействия может описываться локальной параллельной подстановкой в текущей системной подстановке, также можно описать отдельную системную подстановку.

В случаях, когда необходимо учитывать значение какого-либо параметра внешнего воздействия без описания отдельных правил его поведения в контексте существующей параллельной подстановки, есть возможность использовать ее в рамках локальной функции перехода данной подстановки.

Для объявления параметра внешнего воздействия в описании клеточного автомата необходимо указать ключевое слово external.

external тип имя параметры\_инициализации;

Объявление параметра внешнего воздействия схоже с простым объявлением переменной. *Данное ключевое слово разрешает доступ к этой переменной из пользовательского интерфейса, а также всем подстановкам и функциям, определенным с помощью ключевого слова external*. Остальные функциональные эллементы клеточного автомата не имеют доступа к данным параметрам.

Существуют различные варианты использования данного механизма. Он позволяет активировать пользователю какой-либо процесс, описанный специальными подстановками как внешнее воздействие, связанное со своим параметром внешнего воздействия, во время работы клеточного автомата с помощью изменения состояния параметра средствами пользовательского интерфейса. Другими словами, во время процесса моделирования пользователь изменяет значение параметра внешнего воздействия на такое, которое приводит к выполнению подстановок, описываемых необходимое внешнее воздействие.

В случае описания в клеточном автомате нескольких внешних воздействий пользователь может объявить вектор параметров внешних воздействий для управления ими.

Также возможен вариант описания внешнего воздействия, которое начинает работать по определенному событию, под которым подразумевается изменение состояния параметра внешнего воздействия, однако уже не пользователем, а каким либо процессом или вычислением, заранее описанным пользователем в виде подстановок. В этом случае пользователю также необходимо будет объявить такую подстановку с ключевым словом external.

* 1. Проектирование среды моделирования

Основными требованиями для среды клеточно-автоматного моделирования являются:

* наличие пользовательского интерфейса;
* возможность описания клеточно-автоматной модели на спроектированном языке;
* наличие виджета в интерфейсе, отвечающего за визуализацию описанной модели;
* наличие интерпретатора, который проверяет корректность описания модели и осуществляет выполнение моделирования.

На рисунке 2.1 представлена структурная схема среды моделирования.

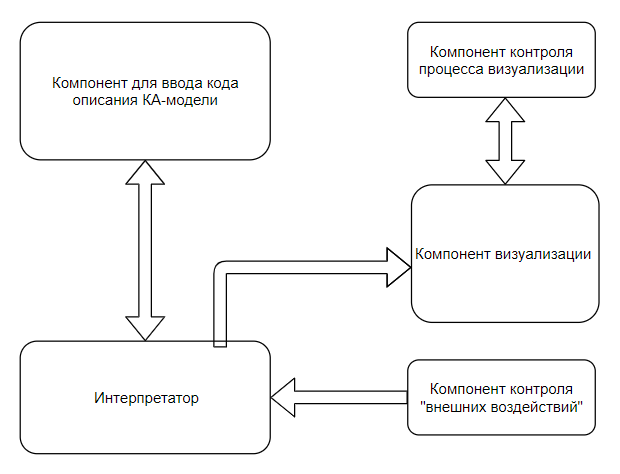


Рисунок 2.1 – Структурная схема среды моделирования

* + 1. Проектирование интерпретатора

Предполагаемый интерпретатор спроектированного языка является однопроходным, и его главной задачей является анализ и незамедлительная трансляция написанного пользователем кода описания моделируемого клеточного автомата.

На рисунке 2.2 представлена схема интерпретатора.

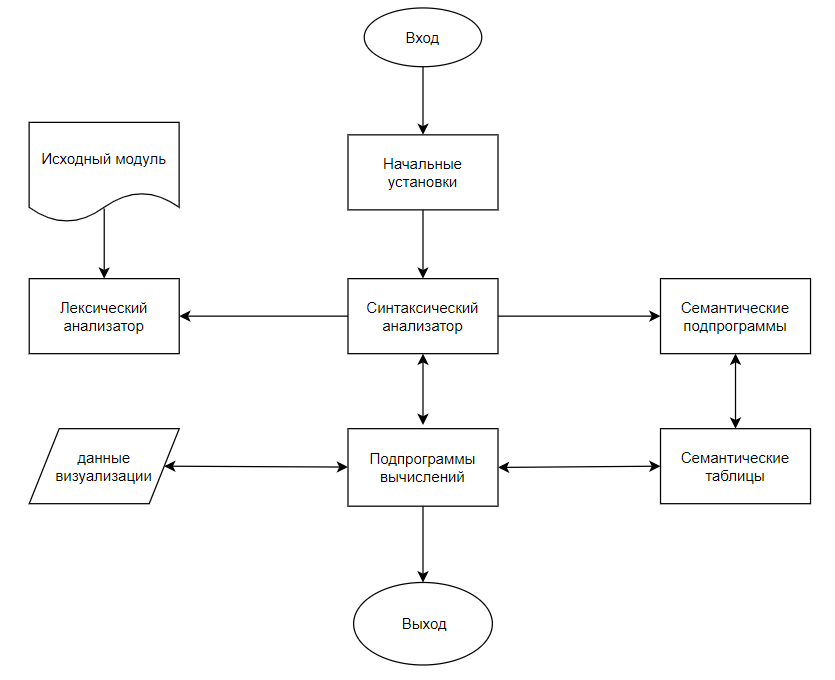


Рисунок 2.2 – Схема интерпретатора

Исходным модулем называется код, который пользователь написал при описании своей клеточно-автоматной модели. Он передается для анализа лексическому анализатору.

Лексический анализатор в свою очередь последовательно считывает лексемы текста исходного модуля. Если встречается лексема, не описанная в таблице лексем, то пользователю выдается лексическая ошибка, указывающая на неизвестную лексическую единицу.

Управление лексическим анализатором осуществляет синтаксический анализатор, который проверяет корректность синтаксиса введенного пользователем кода, анализируя набор считываемых лексическим анализатором лексем.

В качестве синтаксического анализатора был выбран LL-анализатор, поэтому спроектированный синтаксис описан при помощи LL-грамматики. Ниже представлена LL-грамматика:

<программа> → <конструкции> | e

<конструкции> → <конструкции> <элемент> | <элемент>

<элемент> → # <директива> | <КА> | <данные>; | const <данные>;

| classArray <ид-р> <<выражение>, <выражение>>

{ <тело кл.массива> };

<директива> → include <конст.строков.> | pragma <ид-р>

<данные> → <тип> <объект>;

<тип> → int | float | double | char | bool | string | cell | area | <ид-р>

<объект> → [ <измерения> ] <список мас-вов> | <список>

<измерения> → <измерения>, <выражение> | e

<список> → <список>, <переменная> | <переменная>

<переменная> → <ид-р> = <знч пер-ной> | <ид-р>

<знч пер-ной> → <выражение> | new <тип> ( <фак.параметры> )

<фак.параметры> → <фак.параметры>, <выражение> | <выражение>

<список мас-вов> → <список мас-вов>, <массив> | <массив>

<массив> → <ид-р> = <тип.иниц.м.> | <ид-р>

<тип.иниц.м.> → { <иниц.мас.> } | new <тип> [ <выражение> ] |

<выражение>

<иниц.мас.> → <ниц.мас.>, <выражение> | <выражение>

<ф-я> → <тип ф-и> <ид-р> ( <форм.пар-ры> ) <тело функции>

<тип ф-и> → void | <тип> <фун.мас.> | <тип> <фун.мас.>

<фун.мас.> → [ <измерения> ] | e

<форм.пар-ры> → <форм.пар-ры>, <форм.пар-р> | e

<форм.пар-р> → <тип> <пар-р> | const <тип> <пар-р>

<пар-р> → [ <измерения> ] <ид-р> | <ид-р> <иниц.пар-ра> | <ид-р>

<иниц.параметра> → = <выражение>

<тело функции> → { <опер-ры> } | ;

<тело кл.массива> → <тело кл.массива> <э-т т.кл.мc-ва> | e

<э-т т.кл.мc-ва> → <поле>; | neig [] <ид-р> = { <эл.шаблона> };

<эл.шаблона> → <эл.шаблона>, #[ <ук.эл.шаблона> ] |

#[ <ук.эл.шаблона> ]

<ук.эл.шаблона> → <ук.эл.шаблона>, <конст.целая> | <конст.целая>

<КА> → CellAuto <ид-р> (<форм.пар-ры>){<тело КА>};

<тело КА> → <тело КА> <эл-т тела КА> | e

<эл-т тела КА> → external <данные> | <сист.подстан.> | <ф-я>

<сист.подстан.> → <external> System <ид-р> : <имя> <опр.шаблон>

{<парал.подст.>}

<опр.шаблон> → < <имя> > | e

<парал.подст.> → <парал.подст.> <подстановка>; | <подстановка>;

<подстановка> → <external><ид-р> : <база> <контекст> -> <пр.часть> <циклы> | ( <тип> <фун.мас.> , <ид-р> ) {<опер-ры>}

<база> → <база>, <эл-т базы> | <эл-т базы>

<эл-т базы> → ( <var> <выр.базы> , <имя> )

<выр.базы> → <выражение> | $

<var> → var | e

<контекст> → <контекст> <эл-т контекста>| & <эл-т контекста>

| e

<эл-т контекста> → ( <var> <выражение>, <имя> )

<пр.часть> → <пр.часть>, <эл-т пр.части> | <эл-т пр.части>

<эл-т пр.части> → ( <выражение> , <имя> )

<циклы> → <циклы>, <for КА> | : <for КА>

<for КА> → for (<п.цикла1> ; < п.цикла2> ; < п.цикла3>)

<сост.оператор> → { <опер-ры> }

<опер-ры> → <опер-ры> <оператор> | e

<оператор> → <сост.оператор> |

Parallel ( <фак.параметры> ) { <композиция КА> } | ;

<оператор1> → <присваивание> | <if> | <for> | return <возв.значение>|

<данные> | delete <имя> | break | continue

<if> → if ( <выражение> ) <оператор> <else>

<else> → else <оператор> | e

<for> → for (<п.цикла1> ; < п.цикла2> ; < п.цикла3>) <оператор>

<п.цикла1> → <данные> | <присваивание> | e

<п.цикла2> → <выражение> | e

<п.цикла3> → <присваивание> | e

<возв.значение> → <выражение> | e

<композиция КА> → <композиция КА> <эл-т композиции КА> | e

<эл-т композиции КА> → <ид-р> <в.системы> (<фак.параметры> : <выражение> , <выражение>) <домен>;

<в.системы> → . <ид-р> | e

<домен> → domain <конст.целая> | e

<присваивание> → <имя> <присваивание1> | ++ <имя> | -- <имя>

<присваивание1> → = <выражение> <new> | ++ | -- | e

<new> → new <new type> | e

<new type> → ( <фак.параметры> ) | [ <фак.параметры> ]

<имя> → <имя2> ( <фак.параметры> <запуск КА> ) | <имя2>

<имя2> → <имя2> . <имя3> <эл. массив> |

<ид-р> [ <иниц.мас.> ] | <ид-р>

<имя3> → <ид-р> | #

<запуск КА> → : <выражение> , <выражение> | e

<выражение> → + <сравнение> | - <сравнение> | <сравнение>

<сравнение> → <сравнение> > <слагаемое> |

<сравнение> >= <слагаемое> |

<сравнение> < <слагаемое> |

<сравнение> <= <слагаемое> |

<сравнение> == <слагаемое> |

<сравнение> != <слагаемое> |

<сравнение> & <слагаемое> |

<сравнение> && <слагаемое> |

<сравнение> | <слагаемое> |

<сравнение> || <слагаемое> | <слагаемое>

<слагаемое> → <слагаемое> + <множитель> |

<слагаемое> - <множитель> | <множитель>

<множитель> → <множитель> \* <эл.выр> |

<множитель> / <эл.выр> |

<множитель> % <эл.выр> | <эл.выр>

<эл.выр> → <имя> | <константа> | ( <выражение> )

<константа> → <конст.целая> | <конст.веществ.> |

<конст.символьн.> |

<конст.строков.> | true | false | NULL

Семантика кода проектируемого языка использует синтаксически-управляемый перевод. Напомним, что СУ-перевод представляет собой КС-грамматику, в которой к каждому правилу добавлены элементы перевода (дельта-правила). Всякий раз, когда правило учавствует в выводе выходной цепочки, с помощью элемента перевода вычисляется часть выходной цепочи, соответветствующая части входной цепочки, порожденной этим правилом.

В таблице 2.1 представлен СУ-перевод для LL-грамматики проектируемого языка. Префиксом **ΔS** обозначаются дельта-правила семантики.

Таблица 2.1 – СУ-перевод для LL-грамматики проектируемого языка

| LL(1)-грамматика | СУ-перевод |
| --- | --- |
| PRG → CTN INF | t(PRG) → t(CTN) t(INF) |
| FRM → id id; FRM |  id::id TFN FRM | e | t(FRM) → ΔS1 ΔS2 ΔS5 t(FRM)|  ΔS30 ΔS31 t(TFN)  t(FRM) | e |
| CTN → ELM CTN | e |  |
| ELM → # DIR | CLA | DAT; |  const DAT; | FUN |  classArray id < VR1, VR1>  { TCM }; | t(ELM) → DIR | CLA | DAT | ΔS6  t(DAT) | t(FUN) ΔS10 ΔS37 |  ΔS14 t(TCL) ΔS10 | ΔS27  t(VR1) t(VR1) t(TCM) ΔS10 |
| DAT → TYP OBJ | t(DAT) → ΔS1 t(TYP) t(OBJ) ΔS5 |
| TYP → int | float | double | char |  bool | string | cell | area | id |  |
| OBJ → [ IZM ] MS1 | SP1 | t(OBJ) → ΔS4 t(IZM) t(MS1) t(SP1) |
| IZM → , IZM | e | t(IZM) → ΔS4 t(IZM) | e |
| SP1 → PM1 SP2 |  |
| SP2 → , SP1 | e |  |
| PM1 → id PM2 | t(PM1) → ΔS2 t(PM2) |
| PM2 → **=** PM3 | e | t(PM2) → t(PM3) | e |
| PM3 → VR1 | new TYP ( FP1 ) | t(PM3) → t(VR1)|  ΔS24 t(TYP) t(FP1) ΔS22 |
| FP1 → VR1 FP2 | e | t(FP1) → ΔS20 ΔG4 t(VR1) ΔGp  t(FP2) | e |
| FP2 → , VR1 FP2 | e | t(FP2) → ΔS20 t(VR1) t(FP2) | e |
| MS1 → MV1 MS2 |  |
| MS2 → , MS1 | e |  |
| MV1 → id MV2 | t(MV1) → ΔS2 t(MV2) |
| MV2 → = TIN | e | t(MV2) → t(TIN) | e |
| TIN → { IM1 } | new TYP [ IM1 ] | VR1 | t(TIN) → t(IM1) | ΔS24 t(TYP) t(IM1) ΔS23 | t(VR1) |
| IM1 → VR1 IM2 | t(IM1) → ΔS21 t(VR1) t(IM2) |
| IM2 → , IM1 | e |  |
| FUN → TYP FBJ id ( FO1 ) TFN |  void id ( FO1 ) TFN | t(FUN) → ΔS1 t(TYP) t(FBJ) ΔS9  ΔS5 t(FO1) t(TFN) |  ΔS1 ΔS9 ΔS5 t(FO1)t(TFN) |
| FBJ → [ IZM ] | e | t(FBJ) → ΔS4 t(IZM) | e |
| FO1 → PR1 FO2 | e | t(FO1) → ΔS8 t(PR1) t(FO2) ΔS11 | e |
| FO2 → , PR1 FO2 | e |  |
| PR1 → TYP PR2 | const TYP PR2 | t(PR1) → ΔS1 t(TYP) t(PR2) ΔS5 |  ΔS6 ΔS1 t(TYP) t(PR2) ΔS5 |
| PR2 → [ IZM ] id | id INP | t(PR2) → ΔS4 t(IZM) ΔS2|  ΔS2 t(INP) |
| INP → = VR1 | e | t(INP) → t(VR1) | e |
| TFN → { OIO } | ; | t(TFN) → ΔS12 t(OIO) | ; |
| TCL → POL; TCL | FUN TCL |  id ( FO1 ) TFN TCL |  ~ id () TFN TCL | e | t(TCL) → t(POL) t(TCL) |  t(FUN) ΔS10 t(TCL) |  ΔS15 ΔS5 t(FO1) t(TFN)  ΔS10 t(TCL) |  ΔS16 ΔS5 t(TFN) ΔS10  t(TCL) | e |
| POL → TYP EPL | t(POL) → ΔS1 t(TYP) t(EPL) ΔS5 |
| EPL → [ IZM ] MP1 | PS1 | t(EPL) → ΔS4 t(IZM) t(MP1) | t(PS1) |
| PS1 → id PS2 | t(PS1) → ΔS2 t(PS2) |
| PS2 → , PS1 | e |  |
| MP1 → id MP2 | t(MP1) → ΔS2 t(MP2) |
| MP2 → , MP1 | e |  |
| TCM → POL; TCM | e |  neig id = { ET1 }; TCM | t(TCM) → t(POL) t(CM) | e |  ΔS1 ΔS4 ΔS2 t(ET1)  ΔS5 t(TCM) |
| ET1 → #[ c1 NEI ] ET2 | t(ET1) → t(NEI) t(ET2) |
| ET2 → , ET1 | e |  |
| NEI → , c1 NEI | e | t(NEI) → t(NEI) | e |
| CLA → CellAuto id ( FO1 ) { TC1 }; | t(CLA) → ΔS32 t(FO1) t(TC1)  ΔS10 ΔS37 |
| TC1 → SPT TC1 | FUN TC1 | e | t(TC1) → t(SPT) t(TC1) |  ΔS7 t(FUN) ΔS10  t(TC1) | e |
| SPT → System id : NM1 OPT { PP1 } | t(SPT) → ΔS34 t(NM1) t(OPT)  t(PP1) ΔS10 |
| OPT → < NM1 > | e |  |
| PP1 → POD; PP2 | CPT PP2 |  |
| PP2 → POD; PP2 | CPT PP2 | e |  |
| POD → id : BS1 CN1 -> PC1 CL1 | t(POD) → ΔS35 t(BS1) t(CN1) t(PC1)  t(CL1) |
| BS1 → ( VAR EBS , NM1 ) BS2 | t(BS1) → t(VAR) ΔS40 t(EBS) ΔS41  t(NM1) ΔS42 t(BS2) |
| BS2 → , ( VAR EBS , NM1 ) BS2 | e | t(BS2) → t(VAR) ΔS40 t(EBS) ΔS41 t(NM1) ΔS42 t(BS2) | e |
| EBS → VR1 | $ | t(EBS) → t(VR1) |
| VAR → var | e | t(VAR) → ΔS36 | e |
| CN1 → \* ( VAR VR1 , NM1 ) CN2 | e | t(CN1) → t(VAR) ΔS40  t(VR1) ΔS41 t(NM1)  ΔS42 t(CN2) | e |
| CN2 → , ( VAR VR1 , NM1 ) CN2 | e | t(CN2) → t(VAR) ΔS40 t(VR1) ΔS41  t(NM1) ΔS42 t(CN2) | e |
| PC1 → ( VR1 , NM1 ) PC2 | t(PC1) → t(VR1) ΔS43 t(NM1) ΔS44  t(PC2) |
| PC2 → , ( VR1 , NM1 ) PC2 | e | t(PC2) → t(VR1) ΔS43 t(NM1) ΔS44  t(PC2) | e |
| CL1 → : for ( PK1 ; PK2 ; PK3 )  CL2 | e |  |
| CL2 → , for ( PK1 ; PK2 ; PK3 )  CL2 | e |  |
| CPT → ( TYP FBJ , id ) { OIO } | t(CPT) → ΔS1 t(TYP) t(FBJ) ΔS33  ΔS5 t(OIO) ΔS10 |
| SOP → { OIO } | t(SOP) → ΔS13 t(OIO) ΔS10 |
| OIO → OPR OIO | e |  |
| OPR → PV1; | SOP | IF0 | FOR | ; |  return RET; | delete NM1; |  DAT; |  break; | continue;  Parallel ( FP1 ) { CC1 } | t(OPR) → t(PV1) | t(SOP) |  t(IF0) | t(FOR) | t(RET)|  t(NM1) ΔS26 |  t(DAT) | t(FP1) t(CC1) |
| IF0 → if ( VR1 ) OPR ELS | t(IF0) → t(VR1) t(OPR) t(ELS) |
| ELS → else OPR | e | t(ELS) → t(OPR) | e |
| FOR → for (PK1 ; PK2 ; PK3) OPR | t(FOR) → ΔS13 t(PK1) t(PK2) t(PK3)  t(OPR) ΔS10 |
| PK1 → DAT | PV1 | e | t(PK1) → t(DAT) | t(PV1) | e |
| PK2 → VR1 | e | t(PK2) → t(VR1)| e |
| PK3 → PV1 | e | t(PK3) → t(PV1) | e |
| RET → VR1 | e | t(RET) → t(VR1) | e |
| CC1 → id VSM (FP1 : VR1, VR1) DOM ;  CC2 |  |
| CC2 → CC1 | e |  |
| VSM → . id | e |  |
| DOM → domain c1 | e |  |
| PV1 → NM1 PV2 | ++ NM1 | -- NM1 | t(PV1) → t(NM1) t(PV2) |  t(NM1) ΔS25 |  t(NM1) ΔS25 |
| PV2 → = NEW | ++ | -- | e | t(PV2) → ΔS25 t(NEW) t(VR1) |  ΔS25 | ΔS25 | e |
| NEW → new TYP NWT | VR1 | t(NEW) → ΔS24 t(TYP) t(NWT) |  t(VR1) |
| NWT → ( FP1 ) | [ IM1 ] | t(NWT) → t(FP1) ΔS22 |  t(FP1) ΔS23 |
| NM1 → id EMS NM2 | t(NM1) → ΔS17 t(EMS)  t(NM2) ΔS19 |
| NM2 → . NM3 EMS NM2 | ( FP1 ZCA )  | e | t(NM2) → t(NM3) t(EMS) t(NM2) |  e | ΔS39 ΔS18 t(FP1)  t(ZCA) ΔS38 |
| NM3 → id | # | t(NM3) → ΔS18 | ΔS18 |
| EMS → [ IM1 ] | e | t(EMS) → ΔS39 ΔS18 t(IM1) ΔS38 | e |
| ZCA → : VR1 , VR1 | e | t(ZCA) → ΔS18 t(VR1) t(VR1) | e |
| VR1 → SL1 VR2 | + SL1 VR2 |  - SL1 VR2 | t(VR1) → t(SL1) t(VR2) | t(SL1)  t(VR2) | t(SL1) t(VR2) |
| VR2 → > SL1 VR2 | < SL1 VR2 |  >= SL1 VR2 | <= SL1 VR2 |  == SL1 VR2 | != SL1 VR2 |  & SL1 VR2 | | SL1 VR2 |  && SL1 VR2 | || SL1 VR2 | e | t(VR2) → t(SL1) t(VR2) | … |  t(SL1) t(VR2) | e |
| SL1 → MN1 SL2 |  |
| SL2 → + MN1 SL2 | - MN1 SL2 | e | t(SL2) → t(MN1) t(SL2) | … | e |
| MN1 → ELV MN2 |  |
| MN2 → \* ELV MN2 | / ELV MN2 |  % ELV MN2 | e | t(MN2) → t(ELV) t(MN2) | … | e |
| ELV → NM1 | CON | ( VR1 ) | t(ELV) → t(NM1) | t(CON) |  t(VR1) |

Описание дельта-правил семантики для СУ-перевода проектируемого языка представлены в таблице 2.2

Таблица 2.2 – Описание дельта-правил семантики

| **Дельта-правило** | **Краткое описание** |
| --- | --- |
| **ΔS1** | Сохранение указанного типа данных |
| **ΔS2** | Добавление переменной/массива в семантическое дерево |
| **ΔS3** | Проверка существования идентификатора в семантическое дереве |
| **ΔS4** | Увеличивает сохраненное количество измерений массива на 1 |
| **ΔS5** | Сбрасывает все сохраненные флаги (кроме флага параметра) |
| **ΔS6** | Выставляет сохраненный флаг константы |
| **ΔS7** | Выставляет сохраненный флаг признака tran-функции |
| **ΔS8** | Выставляет сохраненный флаг параметра |
| **ΔS9** | Добавление функции в семантическое дерево |
| **ΔS10** | Восстанавливает из стека указатель в семантическом дереве |
| **ΔS11** | Снимаем сохраненный флаг параметра |
| **ΔS12** | Выставляет флаг описания функции |
| **ΔS13** | Сохраняет в стек вершину дерева и создает нулевого правого потомка |
| **ΔS14** | Добавляет класс в семантическое дерево |
| **ΔS15** | Добавляет конструктор в семантическое дерево |
| **ΔS16** | Добавляет деструктор в семантическое дерево |
| **ΔS17** | Добавляет идентификатор в стек разбора операций доступа |
| **ΔS18** | Сравнивает верхушку стека разбора операций доступа указанным элементом |
| **ΔS19** | Выталкивает вершину стека разбора операций доступа |
| **ΔS20** | Сравнивает верхушку стека разбора операций доступа с признаком параметра функции |
| **ΔS21** | Сравнивает верхушку стека разбора операций доступа с признаком указателя измерения массива |
| **ΔS22** | Сравнивает верхушку стека разбора операций доступа с признаком инициализации переменной |
| **ΔS23** | Сравнивает верхушку стека разбора операций доступа с признаком инициализации массива |
| **ΔS24** | Проверяет корректность конструктора инициализируемого объекта |
| **ΔS25** | Проверяет объект на доступность операции присваивания |
| **ΔS26** | Проверяет объект на доступность оператора delete |
| **ΔS27** | Добавляет класс клеточных массивов в семантическое дерево |
| **ΔS32** | Добавляет клеточный автомат в семантическое дерево |
| **ΔS33** | Добавляет контекстную подстановку в семантическое дерево |
| **ΔS34** | Добавляет систему подстановок в семантическое дерево |
| **ΔS35** | Добавляет параллельную подстановку в семантическое дерево |
| **ΔS36** | Выставляет сохраненный флаг признака var |
| **ΔS37** | Снимает флаг неявного доступа к клеточному массиву |
| **ΔS38** | Выставляет флаг разбора элемента конструкции доступа |
| **ΔS39** | Снимает флаг разбора элемента конструкции доступа |
| **ΔS40** | Проверяет поле "значение" клетки базы/контекста параллельной подстановки (пролог) |
| **ΔS41** | Проверяет поле "значение" клетки базы/контекста параллельной подстановки (эпилог) |
| **ΔS42** | Проверяет поле "указатель клетки" клетки базы/контекста параллельной подстановки |
| **ΔS43** | Проверяет поле "указатель клетки" клетки правой части параллельной подстановки (пролог) |
| **ΔS44** | Проверяет поле "указатель клетки" клетки правой части параллельной подстановки (эпилог) |

Семантические таблицы интерпретатора представлены в виде семантического дерева, где каждый элемент этого дерева является объектом одного из классов, описывающего какой-либо элемент семантических данных. Типы семантичеких таблиц и описания их полей представлены в таблице 2.3.

Таблица 2.3 – Типы семантических таблиц интерпретатора

|  |  |
| --- | --- |
| **Семантическая таблица** | **Поля таблицы** |
| Родительский класс для всех семантических таблиц | * Тип таблицы * Изображение лексемы * Указатели на потомков и родителя в семантическом дереве |
| Переменная | * Тип данных * Флаг external * Ссылка на пользовательский тип в семантическом дереве * Флаг константы * Флаг параметра * Значение именованной константы |
| Массив | * Тип данных * Ссылка на пользовательский тип в семантическом дереве * Флаг константы * Флаг параметра * Флаг external * Значение именованной константы * Количество измерений |
| Функция | * Тип объекта возвращаемого значения * Тип данных возвращаемого значения * Ссылка на пользовательский тип в семантическом дереве * Количество измерений в возвращаемом значении (если массив) * Флаг описания функции * Флаг external |
| Конструктор | * Флаг описания конструктора |
| Деструктор | * Флаг описания деструктора |
| Класс клеточных массивов |  |
| Система подстановок |  |
| Параллельная подстановка |  |
| Контекстная подстановка | * Тип объекта возвращаемого значения * Тип данных возвращаемого значения * Ссылка на пользовательский тип в семантическом дереве * Количество измерений в возвращаемом значении (если массив) * Флаг external |
| Клеточный автомат |  |

* + 1. Проектирование пользовательского интерфейса

Пользовательский интерфейс должен содержать в себе следующие компоненты:

* удобное поле ввода кода описания клеточно-автоматной модели с возможностью работы с несколькими вкладками для описания различных моделей;
* виджет визуализации работы клеточного автомата;
* инструментарий контроля над внешними воздействиями;
* панель контроля за процессом визуализации.

Поле для ввода кода должно иметь изменяемые размеры, а также иметь возможность изменять размер шрифта, что является удобной функцией для пользователя.

Виджет визуализации должен иметь достаточные размеры для четкого отображения процесса моделирования клеточного автомата. При необходимости, для более детального наблюдения за процессом визуализации, требуется возможность открытия виджета в новом окне с изменяемыми размерами.

Для установки и изменения значений вектора внешних воздействий, а также специальных переменных внешних воздействий должно быть предусмотрено поле ввода данных и кнопка подтверждения действия над контролируемыми параметрами. В случаях, когда требуется тщательный пошаговый контроль над влиянием внешних воздействий, можно использовать данный инструментарий вместе функциями контроля за процессом визуализации.

Панель контроля за процессом визуализации должна предоставлять пользователю возможность запуска процесса, его остановку, продолжение и завершение.

# **Программная реализация синтаксически-ориентированной среды клеточно-автоматного моделирования**

Программа имеет модульную структуру, которая изображена на рисунке 3.1. Точки взаимодействия обозначены с помощью стрелок.

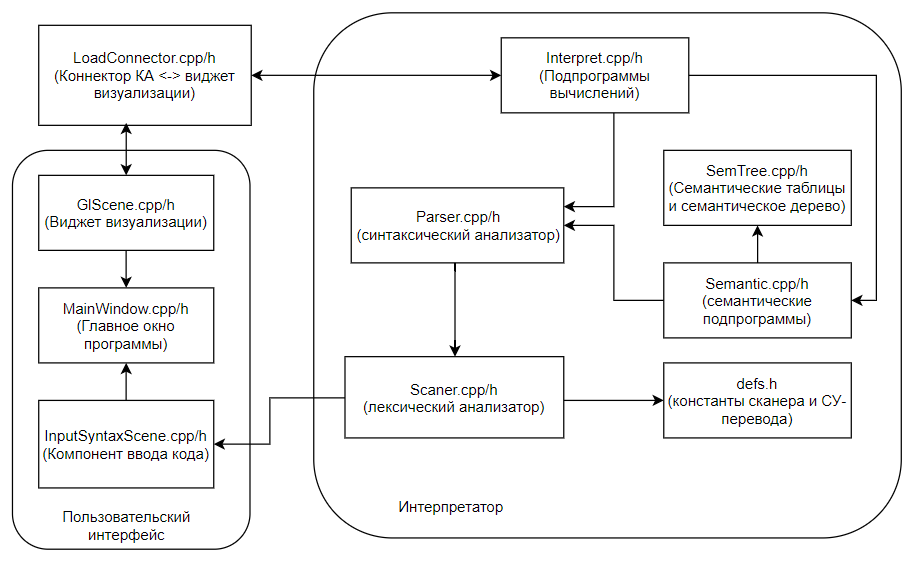


Рисунок 3.1 – Общая структура программы

* 1. Структура интерпретатора

При проектировании интерпретатора были определены основные его компоненты, такие как лексический анализатор и таблица лексем, синтаксический анализатор, процедуры семантического контроля, подпрограммы интерпретации. На рисунке 3.1 программные модули интерпретатора выделены отдельно.

* + 1. Лексический анализатор

Лексический анализатор за один свой проход считывает в исходном тексте одну лексему и выполняет поиск соответствия в таблице лексем, если соответствие найдено, он отдает все данные о лексеме другим модулям, в противном случае возвращается признак ошибочной лексемы.

В программной реализации модуль представлен основным классом *Scaner*, а также файлом, хранящим список лексем *defs.h*.

Класс *Scaner* содержит следующие поля и методы:

*char[] text* – объект для хранения исходного кода описания клеточно-автоматной модели;

*unsigned int pointer* – переменная для хранения номера очередного прочитанного символа исходного кода, является неким указателем на текущую точку анализа текста;

*int column* – переменная, хранящая номер столбца текущего символа анализируемого исходного кода;

*int row* – переменная, хранящая номер строки текущего символа анализируемого текста;

*Scaner(char\*), Scaner(const char\*)* – конструкторы класса с входным параметром в виде указателя на массив символов. Предполагает ипользование в двух случаях: считывание исходного кода из файла, либо получение исходного кода из текстового поля пользовательского интерфейса программной среды;

У полей *pointer, column* и *row* имеются методы получения и установки их значений.

*void loadFromFile(char\* path)* – метод считывания исходного кода в переменную text;

*void showError(char\* err, char\* lex)* – метод вывода ошибки при анализе очередной лексемы;

*int doScan(LEX lex)* – метод анализа очередной лексемы. Возвращает соответствующий идентификатор из таблицы лексем;

Файл *defs.h* содержит список поддерживаемых синтаксисом языка лексем, который представлен в таблице 3.1

Таблица 3.1 – Таблица лексических единиц спроектированного языка

| **Изображение лексемы** | **Тип лексемы** | |
| --- | --- | --- |
| Ключевые слова | | |
| <идентификатор> | t\_id = 1 | |
| var | t\_var = 2 | |
| int | t\_int = 3 | |
| long | t\_long = 4 | |
| double | t\_double = 5 | |
| bool | t\_bool = 6 | |
| neig | t\_neig = 7 | |
| classArray | t\_class\_array = 8 | |
| CellAuto | t\_cell\_auto = 9 | |
| System | t\_system = 10 | |
| const | t\_const = 11 | |
| new | t\_new = 12 | |
| <целая константа> | t\_int = 13 | |
| <целая длинная константа> | t\_long = 14 | |
| <вещественная константа с двойной точностью> | t\_double = 15 | |
| <константа bool> | t\_bool = 16 | |
| for | t\_for = 17 | |
| if | t\_if = 18 | |
| else | t\_else = 19 | |
| delete | t\_delete = 20 | |
| Специальные символы | | |
| { | | t\_op\_brace = 21 |
| } | | t\_cl\_brace = 22 |
| ( | | t\_op\_rnd\_bkt = 23 |
| ) | | t\_cl\_rnd\_bkt = 24 |
| [ | | t\_op\_sc\_bkt = 25 |
| ] | | t\_cl\_sc\_bkt = 26 |
| ; | | t\_semicolon = 27 |
| = | | t\_assignment = 28 |
| # | | t\_scharp = 29 |
| : | | t\_colon = 30 |
| . | | t\_dot = 31 |
| , | | t\_comma = 32 |
| –> | | t\_lambda = 33 |
| \* | | t\_star = 34 |
| + | | t\_plus = 35 |
| - | | t\_minus = 36 |
| / | | t\_div = 37 |
| > | | t\_more = 38 |
| >= | | t\_more\_equal = 39 |
| < | | t\_less = 40 |
| <= | | t\_less\_equal = 41 |
| == | | t\_equal = 42 |
| != | | t\_not\_equal = 43 |
| && | | t\_and = 44 |
| || | | t\_or = 45 |
| $ | | t\_not\_use = 46 |
| @ | | t\_end = 47 |

* + 1. Синтаксический анализатор

Функционал синтаксического анализатора реализован с помощью класса *Parser*. Класс *Parser* содержит следующие поля и методы:

*Scaner\* scaner* – указатель на объект «Лексический анализатор»;

*Semantik \*sem* – указатель на объект «Семантические подпрограммы»;

*Interpret \*intrp* – указатель на объект «Подпрограммы вычислений»

*public void S()* – метод запуска синтаксического разбора;

*public* *void element() –* метод разбора синтаксической конструкции «Элемент программы»;

*public void classArray() –* метод разбора синтаксической конструкции «Класс клеточного массива»;

*public void cellAuto() –* метод разбора синтаксической конструкции «Клеточный автомат»;

*public void data() –* метод разбора синтаксической конструкции «Данные»;

*public void arrayList() –* метод разбора синтакцичекой конструкции «Список массивов»;

*public void type() –* метод разбора синтаксической конструкции «Тип объекта»;

*public void expression() –* метод разбора синтаксической конструкции «Выражение»;

*public void add() –* метод разбора синтаксической конструкции «Сложение»

*public void mul() –* метод разбора синтаксической конструкции «Умножение»;

*public void elExpression() –* метод разбора синтаксической конструкции «Элементарное выражение»;

*public void arrayInit() –* метод разбора синтаксической конструкции «Инициализация массива»;

*public void variable() –* метод разбора синтаксической конструкции «Описание переменной»;

*public void actualParameters() –* метод разбора синтаксической конструкции «Фактические параметры»;

*public void array() –* метод разбора синтаксической конструкции «Массив»;

*public void name() –* метод разбора синтаксической конструкции «Имя»;

*public void nameInner() –* метод разбора синтаксической конструкции «Имя»;

*public void constant() –* метод разбора синтаксической конструкции «Константа».

Каждой синтаксической диаграмме, построенной на основе синтаксиса проектируемого языка соответствует метод разбора в классе *Parser*, который проверяет конструкции на соответствие диаграммам, а также содержит в себе вызовы семантических подпрограмм и подпрограмм интерпретации.

* + 1. Подпрограммы семантического контроля

Подпрограммы семантического контроля представлены классом *Semantik.* Данный класс содержит следующие поля и методы:

*public SemTree \*tree* - указатель на объект «Семантическое дерево»;

*public Scaner \*scan* - указатель на объект «Синтаксический анализатор»;

*public StackTree \*stackTree* - указатель на объект «Стек сохраненных указателей на элементы семантического дерева»;

*public DATA\_TYPE save\_type* – сохраненный тип данных;

*public ElmSemTree \*save\_utype* - сохраненный указатель на пользовательский тип в семантическом дереве;

*public int save\_izm* - сохраненное количество измерений массива;

*public bool save\_const* - сохраненный флаг константы;

*public bool save\_param* - сохраненный флаг параметра;

*public bool save\_external* - сохраненный флаг признака функции внешнего воздействия;

*public bool save\_var* - сохраненный флаг признака var (в параллельной подстановке);

*public ElmSemTree \*fun* - сохраненный указатель на последнюю добавленную функцию в семантическом дереве;

*public bool fspike* – флаг разбора элемента конструкции доступа;

*public bool varadd* - флаг объявления var-переменной;

*public bool fusa* - флаг признака $;

*public Semantic(Scaner \*scanIN)* - конструктор;

*~TSemantic()* - деструктор;

*public void Clear()* - очистка семантических подпрограмм;

*public void ClearSavesFlags()* - очистка всех сохраненных параметров;

*public ElmSemTree\* RetSaveElmOfTree()* - возвращает сохраненный указатель в дереве;

*public void SaveIncIzm()* - увеличивает сохраненное количество измерений массива;

*public void SaveConstChange(bool f)* - изменяет сохраненный флаг константы;

*public void SaveExternalChange(bool f)* - изменяет сохраненный флаг external-функции, а также разрешает неявный доступ к external-переменным, если функция является external-функцией;

*public void ImpAccessChange(bool f)* - изменяет флаг неявного доступа к клеточному массиву;

*public void SaveParamChange(bool f)* - изменяет сохраненный флаг параметра;

*public void SaveVarChange(bool f)* - изменяет сохраненный признак var;

*public void SpikeChange(bool f)* - изменяет флаг разбора элемента конструкции доступа;

*public void OpFunOn()* - выставляет флаг описания функции;

*public bool CeckValueOfCellBaseP(LEX lex, int type)* - проверяет поле "значение" клетки базы/контекста параллельной подстановки (пролог);

*public bool CeckValueOfCellBaseE()* - проверяет поле "значение" клетки базы/контекста параллельной подстановки (эпилог);

*public bool CeckPointerOfCellBase()* - проверяет поле "указатель клетки" клетки базы/контекста параллельной подстановки;

*public bool CeckPointerOfCellRightP()* - проверяет поле "указатель клетки" клетки правой части параллельной подстановки (пролог);

*public bool CeckPointerOfCellRightE()* - Проверяет поле "указатель клетки" клетки правой части параллельной подстановки (эпилог);

*public bool PushToStackAccess(LEX lex)* - заносит идентификатор в стек разбора операций доступа;

*public void ChekTopStackAccessOnSing(int sign)* - сравнивает верхушку стека разбора операций доступа с признаком;

*public void PopToStackAccess()* - выталкивает вершину стека разбора операций доступа;

*public bool ChekTopStackAccess(int type, LEX lex)* - сравнивает верхушку стека разбора операций доступа указанным элементом;

*public bool SaveType(int type, LEX lex)* - запоминание указанного типа;

*public bool AddVar(LEX lex)* - заносит переменную/массив в дерево;

*public bool AddFun(LEX lex)* - заносит функцию в дерево;

*public bool AddCellAuto(LEX lex)* - заносит клеточный автомат в дерево;

*public bool AddContextSub(LEX lex)* - заносит контекстную подстановку в дерево;

*public bool AddSystemSub(LEX lex)* - заносит систему подстановок в дерево;

*public bool AddSub(LEX lex)* - заносит параллельную подстановку в дерево;

*public bool AddClassArray(LEX lex)* - добавляет класс клеточных массивов в семантическое дерево;

*public void AddNul()* - сохраняет в стек вершину дерева и создает нулевого правого потомка;

*public bool CheckVar(String lex, OBJECT\_TYPE obj, DATA\_TYPE type)* - проверка существования идентификатора в дереве;

*public ElmSemTree \*FindIdent(String lex)* - поиск идентификатора в дереве;

*public bool CheckObjectForPrisv(TElmSemTree \*obj)* - проверяет объект на доступность операции присваивания;

*public void CreateSystemObjects()* - добавление в семантическое дерево системных объектов;

*public bool CreateObjectCA()* - создание системного объекта "CA";

*public bool CreateSystemFunctions()* - создание системных функций;

* + 1. Подпрограммы интерпретации

Подпрограммы интерпретации реализованы в классе *Interpret.* Данный класс содержит в себе:

*private bool start –* глобальный флаг признака выполнения клеточного автомата;

*private MainClassArray\* globalArea –* указатель на контейнер для хранения данных клеточного массива;

*public void setGlobalStart(bool f) –* установить признак запуска/остановки работы модели;

*public void runInterpert() –* запуск клеточно-автоматной модели;

*public void RunOnlyParse() –* парсинг и проверка синтаксиса, без интерпретирования;

*public void saveCellAutoState() –* сохранение состояния клеточно-автоматной модели;

*public void printError(string error) –* вывод какой-либо ошибки;

*public MainClassArray\* getGlobalArea() –* получение доступа к контейнеру клеточного массива. Используется в классе LoadConnector для передачи данных для визуализации.

Для соединения модуля нтерпретации и модуля пользовательского интерфейса разработан класс LoadConnector. Его взаимодействие с другими классами представлено на рисунке 3.1. Данный класс содержит в себе следующие поля и методы:

*private Interpret \* intrp –* указатель на класс *Interpter* для получения контроля над процессом интерпретации;

*private PROG\_STATE progState –* переменная для хранения состояния клеточно-автоматной модели. Данное перечисление представлено такими состояниями как:

1. *COMPILE –* состояние клеточно-автоматной модели, при котором происходит анализ кода пользователя на правильность синтаксиса и семантики;
2. START – состояние клеточно-автоматной модели, при котором происходит ее интерпретация и визуализация;
3. STOP – состояние клеточно-автоматной модели, при котором не происходит каких-либо действий;
4. PAUSE – сотояние клеточно-автоматной модели, при котором останавливается процесс выполнения клеточно-автоматной модели;

*public void setProgState(PROG\_STATE) –* установка нового состояния клеточно-автоматной модели;

*public PROG\_STATE getProgState() –* получение текущего состояния клеточно-автоматной модели;

public MainClassArray*\* getGlobalArea() –* получение доступа к данным клеточного массива модели.

* 1. Пользовательский интерфейс

Пользовательский интерфейс представлен классами *MainWindow, GLScene, InputSyntaxScene, CustomGLScene*.

Класс *MainWindow* описывает главное окно программы. Основными методами в данном классе являются методы обработки события по нажатию кнопок в главном окне пользовательского интерфейса:

1. *public void btnCompileClicked*() – обрабатывает событие по нажатию кнопки «Cкомпилировать»;
2. *public void btnStartClicked()* – обрабатывает событие по нажатию кнопки «Запустить»;
3. *public void btnStartStopClicked()* – обрабатывает событие по нажатию кнопки «Пауза/Продолжить»;
4. *public void btnStopClicked()* – обрабатывает событие по нажатию кнопки «Стоп»;
5. *public void btnOpenNew()* – обрабатывает событие по нажатию кнопки «В новом окне»;
6. *public void btnActivateClicked() –* обрабатывает событие по нажатию кнопки «В новом окне».

Класс *GLScene* описывает виджет визуализации, построенный на технологии OpenGL. Он имплементирует класс QGLWidget из библиотеки opengl для платформы Qt. Основной его задачей является отображение состояния данных клеточного массива в дискретный момент времени выполнения клеточно-автоматной модели.

Данный класс содержит в себе следующие поля и методы:

*private LoadConnector \*loadConnector –* указатель на объект класса *LoadConnector* для осуществления взаимодействия с модулем интерпретации;

*private QTimer \*timer –* указатель на объект таймер, который предоставляет интервальные временные события для перерисовки картинки на виджете визуализации;

*protected void initializateGL() –* метод, используемый только один раз при запуске приложения;

*protected void resizeGL() –* метод, перерисовывающий картинку виджета визуализации при изменении размера окна пользовательского интерфеса;

*protected void paintGL() –*  непосредственно перерисовывает изображение по событию сигнала от *QTimer*.

Класс *InputSyntaxScene* описывает набор вкладок с текстовыми полями для ввода синтаксических конструкций описания клеточно-автоматных моделей. Именно с помощью него считывается пользовательский код для парсинга и дальнейшего выполнения. Представлен методами: *loadFromFile() –* для загрузки уже существующих описаний клеточно-автоматных моделей во вкладку, *getTextFromTab(int tabIndex) –* получение текста описания клеточно-автоматной модели.

Класс *CustomGLScene* описывает отдельное окно отображения визуализации процесса выполнения клеточно-автоматной модели. Он функционально схож с классом *GlScene*.

На рисунке 3.2 представлен общий вид основного окна пользовательского интерфейса.

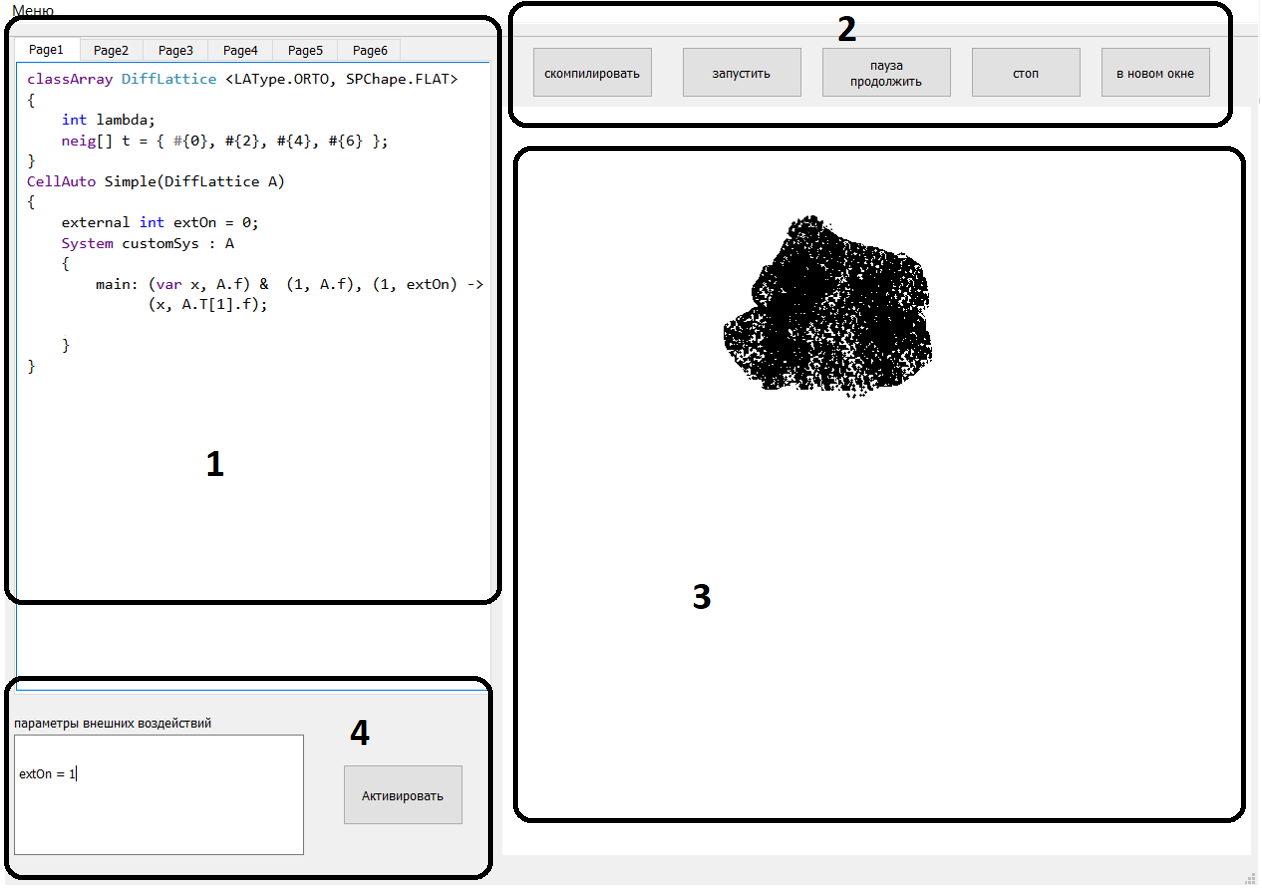


Рисунок 3.2 – Основное окно пользовательского интерфейса

1. Окно для ввода пользователем конструкций описания клеточно-автоматных моделей. Окно предоставляет шесть вкладок, что позволяет единоразово описать до шести клеточно-автоматных моделей;
2. Основной инструментарий управления процессом моделирования. Представлен следующими возможностями:
   1. Кнопка «Скомпилировать». Позволяет пользователю проверить написанные им синтаксические конструкции на правильность ключевых слов, а также семантику без запуска процесса моделирования;
   2. Кнопка «Запустить». Осуществляет запуск процесса интерпретации и моделирования клеточно-автоматной модели;
   3. Кнопка «Пауза/Продолжить». Позволяет пользователю останавливать процесс моделирования с сохранением состояния клеточно-автоматной модели для проведения каких-либо действий (к примеру применения «параметров внешних воздействий» к модели), а затем продолжить его;
   4. Кнопка «Стоп». Осуществляет остановку процесса моделирования без сохранения состояния клеточно-автоматной модели;
   5. Кнопка «В новом окне». Позволяет пользователю открывать виджет визуализации в новом окне с возможностью изменения размеров.
3. Виджет визуализации клеточно-автоматной модели. Отображает состояние данных в клеточном массиве.
4. Инструментарий контроля «внешних воздействий». Представлен полем для описания параметров «внешних воздействий» и их значений. Также в нем можно объявлять новые данные для клеток клеточного массива.

# **Исследования поведения клеточно-автоматных моделей с «внешними воздействиями» в разработанной системе**

Разработанная синтаксичеки ориентированная инструментальная система является учебным стендом для проведения экспериментов с применением «внешних воздействий» при описании клеточно-автоматных моделей. На данной системе проведено несколько экспериментов по клеточно-автоматному моделированию.

Для сравнительного анализа поведения клеточного автомата ТМ-диффузия исследовано его поведение без «внешних воздействий», а также с их применением. Клеточно-автоматная модель ТМ-диффузия разработана учеными Тоффоли Т. и Марголус Н.. Имеет окрестность Марголуса (рисунок 4.1). Данная модель имеет несколько отличительных особенностей. Она обладает свойством, называемое в математике «элегантностью», другими словами сочетанием простоты и эффективности. Также существует строгое математическое доказательство, что она аппроксимирует оператор Лапласа[11].

Клеточный массив разбивают на 2 подмножества, каждое из которых состоит из блоков, содержащих четыре клетки. Функционирование клеточно-автоматной модели происходит в двухтактном синхронном режиме. Каждая итерация делится на два такта. На четных тактах правила перехода применяются к четныи блокам, на нечентных – к нечетным. Правила перехода таковы, что выполняют сдвиг состояний в клетках блока равновероятно по часовой стрелке или против часовой стрелки. Ниже представлен листинг кода описания данной клеточно-автоматной модели в разработанной системе.

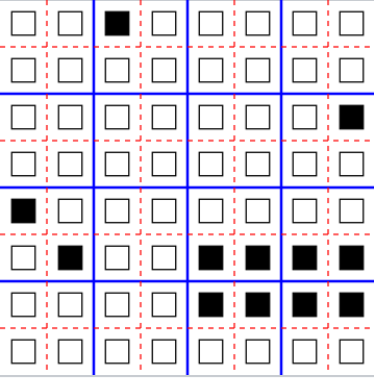


Рисунок 4.1 – Окрестность Марголуса

Листинг обычной клеточно-автоматной модели ТМ-диффузия:

*classArray TArray1 <CA.LAType.HEX, CA.SPChape.FLAT>*

*{*

*int f;*

*neig[] T = { #{0}, #{3}, #{4}, #{5} };*

*}*

*CellAutoTM(TArray1 A, float p)*

*{*

*System : A <A.T>*

*{*

*Q1: (var u1, A.f), (var u2, A.T[1].f), (var u3, A.T[2].f), (var u4, A.T[3].f) &*

*(true, mp), (true, mch) ->*

*(u4, A.f), (u1, A.T[1].f), (u2, A.T[2].f), (u3, A.T[3].f);*

*Q2: (var u1, A.f),(var u2, A.T[1].f), (var u3, A.T[2].f),(var u4, A.T[3].f) &*

*(false, mp),(true, mch) ->*

*(u2, A.f), (u3, A.T[1].f), (u4, A.T[2].f), (u1, A.T[3].f);*

*}*

*context bool mp*

*{*

*if (SystemUtils.randf() < p) return true;*

*else return false;*

*}*

*context bool mch*

*{*

*if (i == j) return true;*

*else return false;*

*}*

*};*

*ТМ(TArray1 A1(1000, 1000) : CA.Mode.BLOCKSYNCH, 0.5);*

На рисунке 4.2 представлен результат моделирования описанной модели.



Рисунок 4.2 – Моделирование КА ТМ-диффузия

В данном эксперименте под «внешним воздействием» понимается добавление нового типа вещества в текущее состояние в процессе моделирования в момент времени, определяемый пользователем. Для осуществдения эксперимента используется разработанный синтаксис описания «внешних воздействий», а также механизм управления ими через «параметры внешних воздействий» с помощью пользовательского интерфейса. Ниже представлен листинг клеточно-автоматной модели ТМ-диффузия «внешним воздействием».

*classArray TArray1 <CA.LAType.HEX, CA.SPChape.FLAT>*

*{*

*int f;*

*neig[] T = { #{0}, #{3}, #{4}, #{5} };*

*neig[] extT = { #{0}, #{1}, #{2}, #{3}, #{4}, #{5}, #{6} };*

*}*

*CellAutoTM(TArray1 A, float p) {*

*external bool extOn = false;*

*System : A <A.T> {*

*Q1: (var u1, A.f), (var u2, A.T[1].f), (var u3, A.T[2].f), (var u4, A.T[3].f) &*

*(true, mp), (true, mch) ->*

*(u4, A.f), (u1, A.T[1].f), (u2, A.T[2].f), (u3, A.T[3].f);*

*Q2: (var u1, A.f),(var u2, A.T[1].f), (var u3, A.T[2].f),(var u4, A.T[3].f) &*

*(false, mp),(true, mch) ->*

*(u2, A.f), (u3, A.T[1].f), (u4, A.T[2].f), (u1, A.T[3].f);*

*external Q3: (0, A.f) & (true, extOn), (true, cnt) -> (2, A.f);*

*}*

*context bool cnt {*

*int inCnt = 0;*

*for(int i = 0; i < A.extT.Size;i++)*

*if(A.f == 2)*

*inCnt++;*

*return inCnt > 2;*

*}*

*context bool mp {*

*if (SystemUtils.randf() < p) return true;*

*else return false;*

*}*

*context bool mch {*

*if (i == j) return true;*

*else return false;*

*}*

*};*

*ТМ(TArray1 A1(1000, 1000) : CA.Mode.BLOCKSYNCH, 0.5);*

На рисунке 4.3 представлен результат эксперимента.

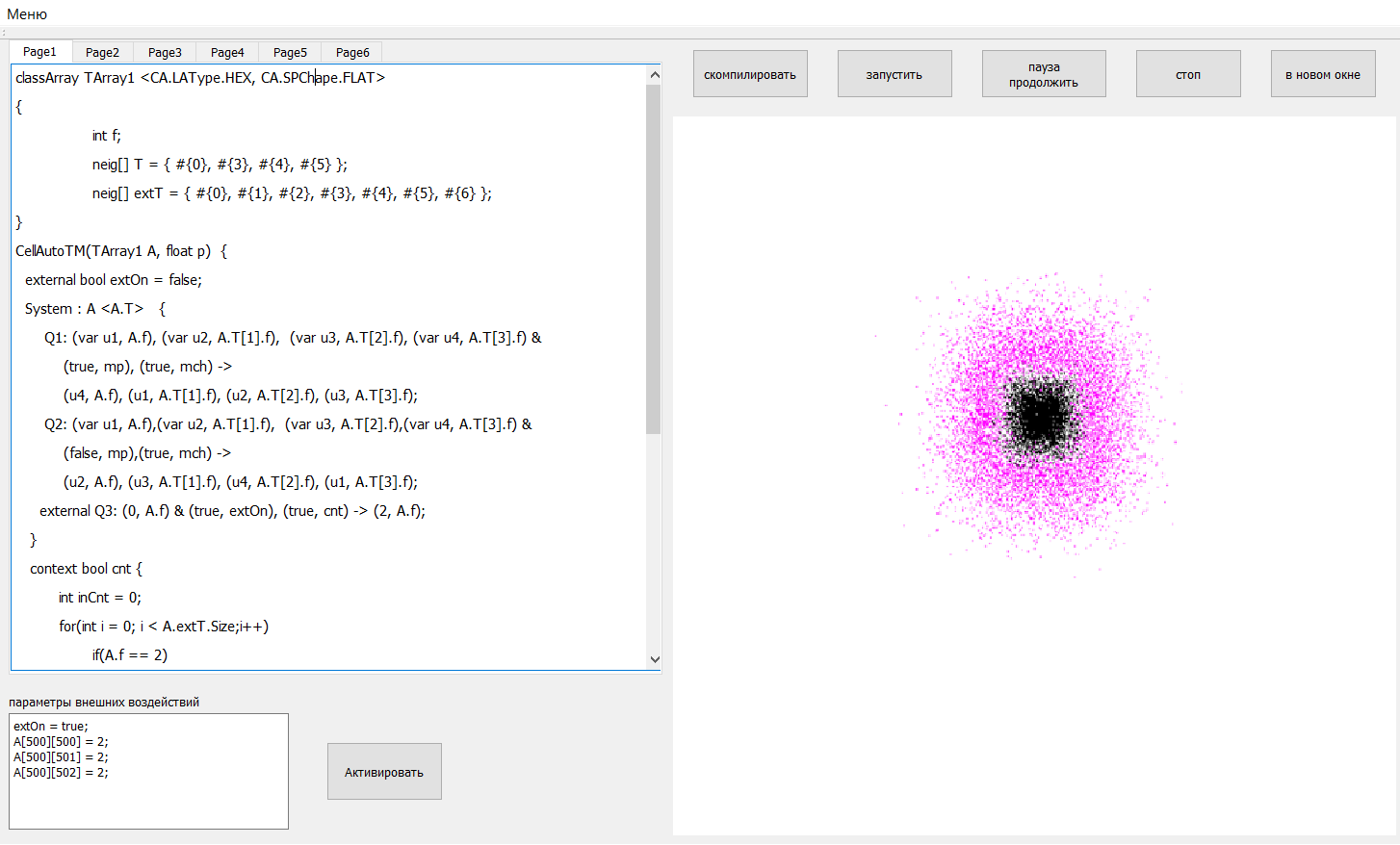


Рисунок 4.3 – КА модель ТМ-диффузия с «внешним воздействием»

Следующим экспериментом является моделирование клеточно-автоматной модели HPP-газ. Известная КА модель, описывающая поведение решетчатых газов. Одна итерация такой модели происходит в два этапа: перемещение частицы в соседние клетки (этап сдвига), соударение частиц в клетках (этап столкновения).

Ниже представлен листинг описания данной КА модели.

*classArray TArrayH<CA.LAType.ORTHO, CA.SPChape.FLAT>*

*{*

*int mas[4];*

*neig[] T = {#{0}, #{1}, #{2}, #{3}, #{4}, #{5}, #{6} };*

*}*

*CellAuto HPP(TArrayH D)*

*{*

*System t1 : D*

*{*

*Q1: ($, D.mas) -> (Fun1(D), D.mas);*

*}*

*System t2 : D*

*{*

*Q2: ($, D.mas) -> (Fun2(D), D.mas);*

*}*

*void Fun1(TArray3 D)*

*{*

*for (int i = 0; i < 4; i++)*

*B.mas[i] = B.T[i+1].mas[i];*

*}*

*void Fun2(TArray2 B)*

*{*

*if (B.mas[0] == 0 && B.mas[1] == 1 &&*

*B.mas[2] == 0 && B.mas[3] == 1)*

*B.mas[0] = 1, B.mas[1] = 0, B.mas[2] = 1, B.mas[3] = 0;*

*}*

*HPP(TArrayH D(700, 700));*

На рисунке 4.4 представлен результат моделирования тривиальной КА модели HPP-газ.

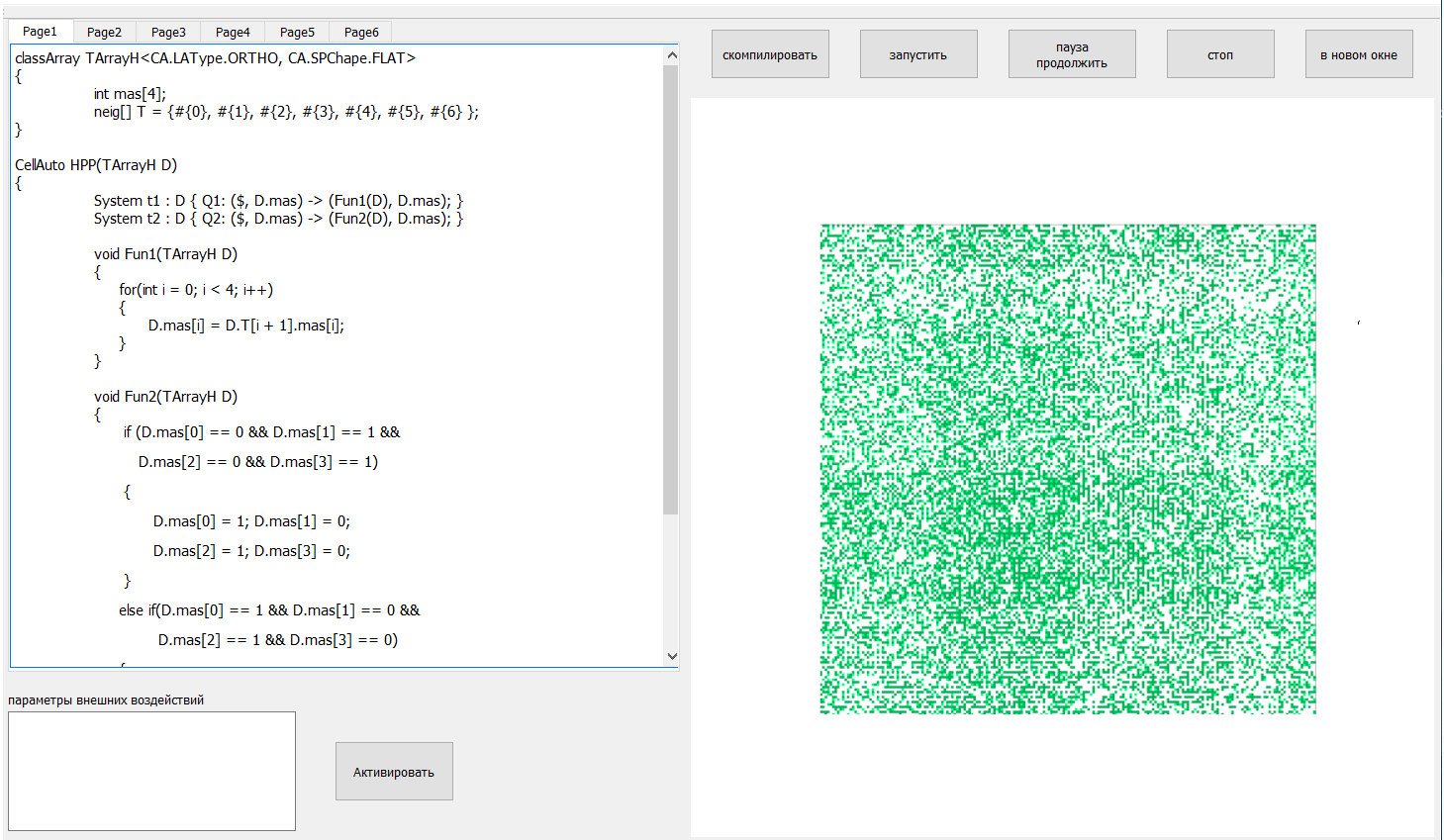


Рисунок 4.4 – КА модель HPP-газ

«Внешним воздействием» в данной модели предполагается добавление газа другого типа. С точки зрения правил перехода в момент добавления нового вещества его молекулы используют свободные клетки КА решетки.

Листинг КА-модели «HPP-газ» с учетом «внешнего воздействия»:

*classArray TArrayH<CA.LAType.ORTHO, CA.SPChape.FLAT>*

*{*

*int mas[4];*

*neig[] T = {#{0}, #{1}, #{2}, #{3}, #{4}, #{5}, #{6} };*

*}*

*CellAuto HPP(TArrayH D)*

*{*

*System t1 : D { Q1: ($, D.mas) -> (Fun1(D), D.mas); }*

*external System t2 : D { Q2: ($, D.mas) -> (Fun2(D), D.mas); }*

*external System tExt : D { Q3: ($, D.mas) -> (Fun3(D), D.mas);}*

*void Fun1(TArrayH D)*

*{*

*for(int i = 0; i < 4; i++)*

*{*

*D.mas[i] = D.T[i + 1].mas[i];*

*}*

*}*

*void Fun2(TArrayH D)*

*{*

*if (D.mas[0] == 0 && (D.mas[1] == 1 || D.mas[1] == 2) &&*

*D.mas[2] == 0 && (D.mas[3] == 1 || D.mas[3] == 2))*

*{*

*if(D.mas[1] == 1)*

*D.mas[0] = 1;*

*else D.mas[0] = 2;*

*D.mas[1] = 0;*

*if(D.mas[3] == 1)*

*D.mas[2] = 1;*

*else D.mas[2] = 2;*

*D.mas[3] = 0;*

*} else if((D.mas[0] == 1 || D.mas[0] == 2) && D.mas[1] == 0 &&*

*(D.mas[2] == 1 || D.mas[2] == 1) && D.mas[3] == 0)*

*{*

*if(D.mas[0] == 1)*

*D.mas[1] = 1;*

*else D.mas[1] = 2;*

*D.mas[0] = 0;*

*if(D.mas[2] == 1)*

*D.mas[3] = 1;*

*else D.mas[3] = 2;*

*D.mas[2] = 0;*

*}*

*}*

*void Fun3(TArrayH D)*

*{*

*if(D.mas[0] == 0 && D.mas[1] == 0 &&*

*D.mas[2] == 0 && D.mas[3] == 0)*

*D.mas[0] = 2;*

*}*

*}*

*HPP(TArrayH D(700, 700));*

На рисунке 4.5 представлен результат моделирования.

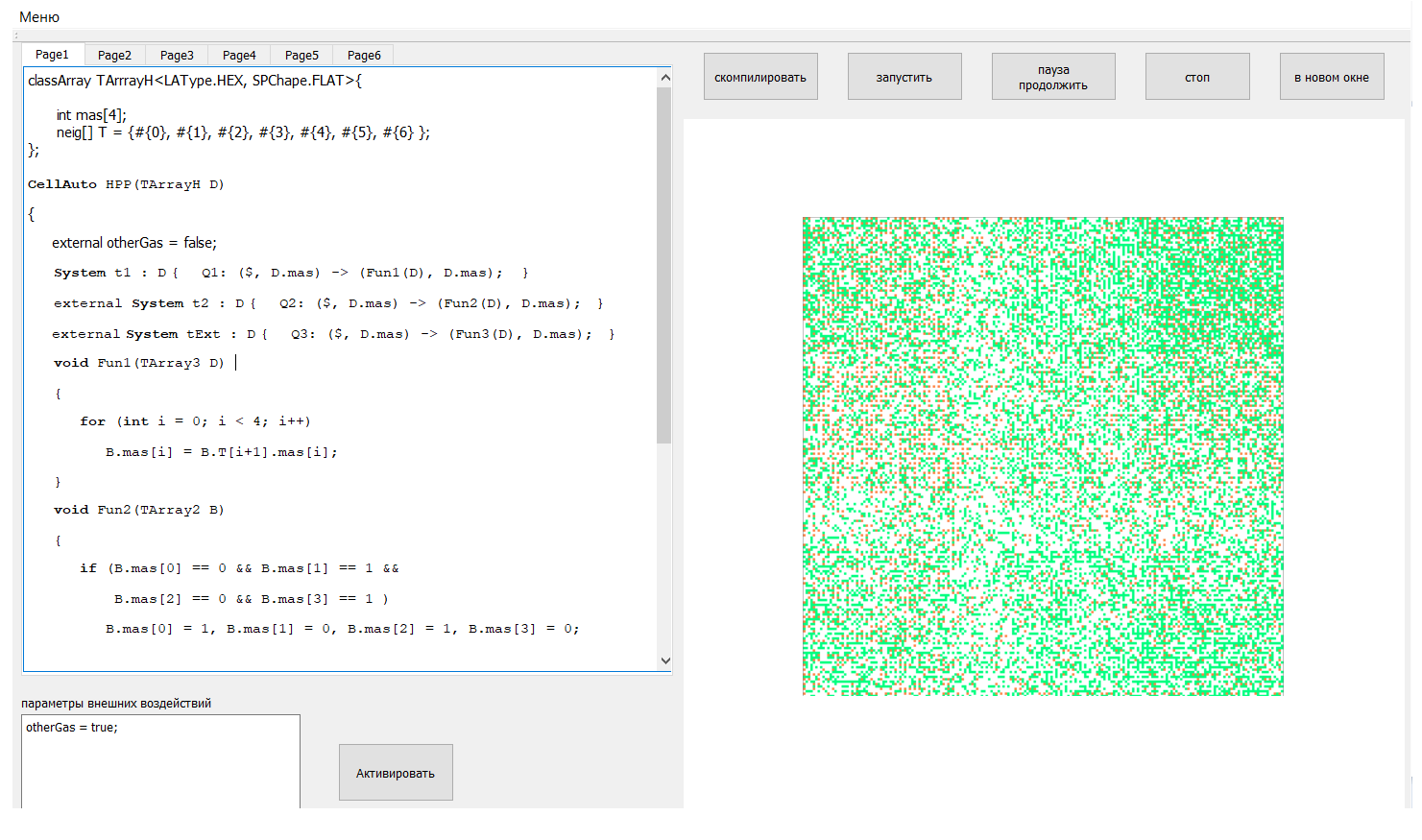


Рисунок 4.5 – КА модель «HPP-газ с внешним воздействием»

# **Заключение**

В ходе выполнения данной практики были выполнены в полном объеме все задачи для достижения поставленной цели практики.

На первом этапе были проанализированы уже существующие синтаксически-ориентированные системы клеточно-автоматного моделирования, а также выделены их основные проблемы.

На следующем этапе был спроектирован собственный язык описания клеточно-автоматных моделей с новыми функциональными возможностями по описанию «внешних воздействий».

На третьем этапе осуществлялась проектирование программной части инструментальной системы и её разработка.

На четвертом этапе были проведены эксперименты по моделированию клеточно-автоматных моделей. Проводился анализ работы тривиальной модели и модели с внедрением «внешних воздействий».

В результате выполнения практики выполнены следующие задачи:

1. Исследование существующих решений для клеточно-автоматного моделирования;
2. Проектирование и разработка собственного синтаксиса описания моделей;
3. Проектирование и разработка среды моделирования с применением разработанного языка.
4. Проведение экспериментов при моделировании клеточно-автоматных моделей с использованием «внешних воздействий»

**Список использованных источников**

1. Toffolli T. Cellular Automata Machine [Текст] / Toffolli T., Margolus N., USA: MIT Press, 1987. – 280c.
2. Achasova S. Parallel Substitution Algorithm. Theory and Application [Текст] / Achasova S., Bandman O., Markova V., Piskunov S., Singapore: World Scientific, 1994. – 296c.
3. Бандман О.Л. Метод построения клеточно-автоматных моделей процессов формирования устойчивых структур [Текст] / Бандман О.Л., ПДМ, 2010, № 4, 91–99.
4. Бандман О.Л. Параллельная реализация клеточно-автоматных алгоритмов моделирования пространственной динамики [Текст] / Бандман О.Л., Сиб. журн. вычисл. математики., РАН. Сиб. отд-ние. – Новосибирск, 2007. – Т. 10, №4. – С.335-348.
5. Бандман О.Л. Клеточно-автоматные модели пространственной динамики [Текст] / Бандман О.Л., Системная информатика, -Новосибирск, 2006, №10, С59.-113.
6. Бандман О.Л. Мелкозернистый параллелизм в вычислительной математике [Текст] / Бандман О.Л., Программирование, -Новосибирск, 2001, №4, 1-18.
7. Бандман О.Л. Методы композиции клеточных автоматов для моделирования пространственной динамики [Текст] / Бандман О.Л., Вестник Томского университета, -Томск, 2002, №9(1), 188-192.
8. Bandman O.L. Cellular Automata composition techniques for spatial dynamics simulation [Текст] / Bandman O.L., Bull. Nov. Comp. Center, Comp. Science., 2008. – V.27. P.1-39.
9. Bandman O.L. Parallel Simulation of Asynchronous Cellular Automata Evolution [Текст] / Bandman O.L., Lect. Notes in Comp. Sci., 2006. – Vol. 4173. P.41-48.
10. Медведев Ю.Г. Расширение клеточно-автоматной модели потока FHP-I до многочастичного варианта FHP-MP [Текст] / Медведев Ю.Г., Новые информационные технологии в исследовании сложных структур, -Томск, 2008. – С73.
11. Малинецкий Г. Г., Степанцов М. Е. Моделирование диффузионных процессов клеточными автоматами с окрестностью Марголуса // Журн. Вычислит. Матем. И математич. Физ. 1998. – №6. – 1017-1021 с.
12. Cellular Automata [Электронный ресурс] / Режим доступа: http://www.cell-auto.com
13. Крючкова Е.Н. Теория языков программирования и методы трансляций [Текст] / Крючкова Е.Н. - Алт. гос. техн. унив. Им. И.И.Ползунова -Барнаул: АлтГТУ им. И.И. Ползунова, 2003. – 240 с..
14. Конспект лекций «Методы построения компиляторов» [Электронный ресурс] / Режим доступа: <http://it.mmcs.rsu.ru/wiki/Конспект_лекций_> «Методы\_построения\_компиляторов»
15. Элементы теории перевода [Электронный ресурс] / Режим доступа: http://citforum.ru/programming/theory/serebryakov/5.shtml
16. Павловская Т.А. С/C++ Программирование на языках высокого уровня [Текст] / Павловская Т.А. - СПб.: Питер, 2004. – 461с.
17. H. Gutowitz., Cellular automata: theory and experiment // Mit Press. Cambridge, 1991
18. В. К. Ванаг, “Исследование пространственно распределенных динамических систем методами вероятностного клеточного автомата”, УФН, 169:5 (1999), 481–505; Phys. Usp., 42:5 (1999), 413–434
19. Wolfram S. A New Kind of Science. N. Y.: Wolfram Media, 2002. 1197 с.
20. Culik K., Yu S. Undecidability of CA classification schemes // Complex Systems. 1988. Р. 177-190.
21. Li W., Packard N. The structure of the elementary cellular automata rule space // Complex Systems, 1990. Р. 281-297.
22. G. Braga [et al] Pattern growth in elementary cellular automata // Theoret. Comput. 1995. Р. 1-26.
23. Fates N., Morvan M., Remila E. Experimental study of elementary cellular automata dynamics using the density parameter // Discrete Mathematics Theoretical Computer Science. 2003. Р. 155-166.
24. Введение в математическое моделирование транспортных потоков / А.В. Гасников[и др.]М.: МФТИ, 2010.
25. Степанцов М.Е. Применение клеточных автоматов для математического моделирования динамических процессов: Автореф. дис. …канд. тех. наук. –М., 1999. – 18 с.
26. Малинецкий Г. Г., Степанцов М. Е. Применение моделей класса решеточных газов для решения задач газодинамики // Известия Высших учебных заведений. Прикладная нелинейная динамика, 1996, том 4, N 4,5, с.59-04.
27. Жуков А.Е., Клеточные автоматы в криптографии //Вопросы кибербезопасности. – 2017. – №4(22). – C. 47-66.
28. Разработка генераторов псевдослучайных двоичных последовательностей на основе клеточных автоматов Сухинин Б.М. [Электронный ресурс] / Науки и образование. Режим доступа: <http://technomag.edu.ru/doc/159714.html>. свободный. – Загл. с экрана.
29. Долгушин Д.Ю. Многофакторное моделирование транспортных потоков на основе клеточных автоматов: Автореф. дис. …канд. тех. наук. –М., 1999. – 18 с
30. Тоффоли Т., Марголус Н., Машины клеточных автоматов. М.: Мир, 1991. – 280 с.
31. Legendi T. Cellprocessors in Computer Architecture // Comp. Linguist. and Comp. Languages. 1976. Vol. 11. N 2. P. 147—167.
32. Cannataro M. et al. A parallel cellular automata environment on multicomputers for computational science // Parallel Computing. 1995. Vol. 21. P. 803—823.
33. Наумов Л. Метод введения обобщенных координат и инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экпериментов с использованием клеточных автоматов //Дисс. на соиск. уч. ст. канд. техн. наук. СПбГУ ИТМО. 2007. 283 с.
34. Матюшкин И. В. Перспективы развития современных средств проектирования клеточных автоматов // Информационные технологии. – 2011. – №4. С. 8 – 12.
35. Остапкевич М. Б., Пискунов С.В. Система имитационного моделирования алгоритмов с мелкозернистым параллелизмом WinALT // Вестник НГУ. Сер. Информационные технологии. 2012. Том 10. Выпуск 3. – С. 34 – 45.
36. Остапкевич М. Б. Параллельная реализация мелкозернистых алгоритмов в системе WinALT // Вестник ЮУрГУ. Сер. Вычислительная математика и Информатика. 2012. Том 2. Выпуск 1. –С. 80 – 89.
37. Баранов Е.Ю., Мирозян А.С., Петров А.В. Программа моделирования алгоритма параллельных подстановок и пример его реализации на ПЛИС [Электронный ресурс] // Конструкторское бюро «Информационные технологии», 2015. URL: <http://kbit.rsatu.ru/skbit/images/articles/plis/1.pdf> (дата обращения: 01.03.2018).