МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

АГЕНТНОЕ МОДЕЛИРОВАНИЕ ИГРЫ «ЖИЗНЬ»

Курсовая работа

Легушева Дмитрия Александровича

студента 2 курса, специальность
1-31 03 09 Компьютерная математика и системный анализ

Научный руководитель:
кандидат физ.-мат. наук,
доцент О. А. Лаврова

Минск, 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Моделирование является неотъемлемой стороной человеческой деятельности – от живописи до математического моделирования сложных систем и имеет многовековую историю. В целом моделирование – это общенаучный метод изучения свойств объектов и процессов по их моделям, используемый в целях познания, исследования, проектирования, принятия решений. В настоящее время методы компьютерного моделирования прочно вошли в практику решения широкого круга теоретических проблем и прикладных технических задач в различных сферах практической деятельности.

Компьютерное моделирование становится сегодня обязательным этапом в принятии ответственных решений во всех областях деятельности человека в связи с усложнением систем, в которых человек должен действовать и которыми он должен управлять. Сущность компьютерного моделирования состоит в построении модели, которая представляет собой некоторый программный комплекс, алгоритмически описывающий развитие процесса или поведение объекта.

В данной курсовой работе рассказывается об одном из типов имитационного моделирования — агентном моделировании с акцентом на моделировании клеточных автоматов. Агентное моделирование описывает поведение целой систем посредством описания поведения ее локальных элементов.

Курсовая работа носит программно-исследовательский характер. При этом целью работы является исследование принципов агентного моделирования на примере реализации игры «Жизнь». Ведь само агентное моделирование является относительно новым методом, получившим широкое практическое распространение только после 2000 года. Поэтому сейчас стоит задача не только разобраться с данным видом, а также реализовать собственную программу на основе данного вида.

ГЛАВА 1
ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Введение в агентное моделирование

Моделирование – метод исследования объекта познания (оригинала) путем его замещения другим объектом – моделью. Моделирование применяется, когда проведение экспериментов над реальной системой невозможно или нецелесообразно, например, из-за высокой стоимости или длительности проведения эксперимента в реальном масштабе времени. или для некоторой оптимизации продукта до его реализации. Моделирование включает в себя отображение проблемы из реального мира в мир моделей, анализ модели, нахождения решения, и отображение решения обратно в реальный мир [1].

Имитационное моделирование (ситуационное моделирование) – метод, позволяющий создавать модели, который описывают процессы так, как они проходили бы в действительности [2]. Чаще всего имитационная модель является компьютерной программой, которая позволяет получать подробную статистику о различных аспектах функционирования системы в зависимости от входных данных путем постановки над ней экспериментов. Целью данного моделирования в конечном счете является принятие обоснованных, целесообразных управленческих решений. Знание принципов и возможностей имитационного моделирования, умение создавать и применять модели являются необходимыми требованиями к инженеру, менеджеру, бизнес-аналитику [3].

Агентное моделирование – метод имитационного моделирования, исследующий поведение децентрализованных агентов и то, как такое поведение определяет поведение всей системы в целом [4]. Под агентом в агентном моделировании понимается элемент модели, который может иметь поведение, память (историю), контакты и т.д. и может моделировать людей, компании, проекты, автомобили, города, животных, корабли, товары и т.д. Агентное моделирование зародилось в 1990-х гг. В стенах Университета Карнеги-Меллон. Сегодня это, пожалуй, наиболее передовой метод имитационного моделирования, который используется учеными и исследователями в области экономики и управления, позволяющий смоделировать обстановку практически неограниченной сложности. Агентное моделирование позво-

ляет сделать историческую науку экспериментальной и вводить в нее сослагательное наклонение, которое порой может дать богатую пищу для размышлений современному поколению[5].

Агентное моделирование основывается на двух основных типа построения агентских моделей:

• на основе программного кода (пример может служить программа NetLogo).

• на основе карты состояний («стейтчарта») и переходов между состояниями (пример такого руководства служит программа AnyLogic).

Агентное моделирование является инструментом, при помощи которого возможно успешное моделирование сложных адаптивных систем. Оно позволяет моделировать неагрегированные элементы системы и базируется на идее моделирования процессов «снизу вверх»: в основе модели лежит набор основных элементов, из взаимодействия которых рождается обобщенное поведение системы.

Возможности агентного моделирования:

- Оптимизация сети поставщиков и планирование перевозок;
- Планирование развития производства;
- Прогнозирование спроса на продукцию и объемы продаж;
- Оптимизация численности персонала;
- Прогнозирование развития социально-экономических систем (городов, регионов);
- Моделирование миграционных процессов;
- Имитация и оптимизация пешеходного движения;
- Моделирование транспортных систем;
- Прогнозирование экологического состояния окружающей среды и т.д.

1.2 Программное обеспечение для имитационного моделирования – AnyLogic

AnyLogic – программное обеспечение для имитационного моделирования, разработанное российской компанией The AnyLogic Company (бывшая «Экс Джей Технолоджис», англ. XJ Technologies) [5]. AnyLogic используется для разработки имитационных исполняемых моделей и последующего их прогона для анализа. Разработка модели выполняется в графическом редакторе AnyLogic с использованием многочисленных средств поддержки, упрощающих работу. Построенная модель затем компилируется встроенным компилятором AnyLogic и запускается на выполнение. В процессе выполнения модели пользователь может наблюдать ее поведение, изменять параметры модели, выводить результаты моделирования в различных формах и выполнять разного рода компьютерные эксперименты с моделью.

AnyLogic был разработан на основе новых идей в области информационных технологий, теории параллельных взаимодействующих процессов и теории гибридных систем. Благодаря этим идеям чрезвычайно упрощается построение сложных имитационных моделей, имеется возможность использования одного инструмента при изучении различных стилей моделирования. Программный инструмент AnyLogic основан на объектно-ориентированной концепции. Другой базовой концепцией является представление модели как набора взаимодействующих, параллельно функционирующих активностей. Активный объект в AnyLogic – это объект со своим собственным функционированием, взаимодействующий с окружением. Он может включать в себя любое количество экземпляров других активных объектов. Графическая среда моделирования поддерживает проектирование, разработку, документирование модели, выполнение компьютерных экспериментов, оптимизацию параметров относительно некоторого критерия [3].

AnyLogic является надстройкой над языком Java – одним из самых мощных и в то же время самых простых современных объектно-ориентированных языков. Все объекты, определенные пользователем при разработке модели с помощью графического редактора, компилируются в конструкции языка Java, а затем происходит компиляция всей собранной программы на Java, задающей модель, в исполняемый код. Хотя программирование сведено к минимуму, разработчику модели необходимо иметь некоторое представление об этом языке.

В начале 1990-х в компьютерной науке наблюдался большой интерес к построению математически трактуемого описания взаимодействия параллельных процессов. Что сказалось на подходах к анализу корректности параллельных и распределённых программ. Группа учёных из Санкт-Петербургского Политехнического университета разработала программное обеспечение для анализа корректности системы; новый инструмент назвали COVERS (Параллельная Верификация и Моделирование). Анализируемая система процессов задавалась графически, с помощью описания её структуры и поведения отдельных параллельных компонентов, которые могли взаимодействовать с окружением – с другими процессами и средой. Инструмент использовался в исследовательских проектах компании Хьюлетт-Паккард.

В 1998 г. успех этого исследования вдохновил лабораторию организовать коммерческую компанию с миссией создания нового программного обеспечения для имитационного моделирования. Акцент при разработке ставился на прикладные методы: моделирование стохастических систем, оптимизацию и визуализацию модели. Новое программное обеспечение, выпущенное в 2000 г., было основано на последних преимуществах информационных технологий: объектно-ориентированный подход, элементы стандарта UML, языка программирования Java, современного GUI, и других [10].

Продукт получил название AnyLogic, потому что он поддерживал все три известных метода моделирования:

- Системная динамика;
- Дискретно-событийное (процессное) моделирование;
- Агентное моделирование.

А также любую комбинацию этих подходов в пределах одной модели. Первой версии был присвоен индекс 4 – Anylogic 4.0, так как нумерация продолжила историю версий предыдущей разработки — COVERS 3.0.

1.3 Игра «Жизнь»

Клеточный автомат – дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Включает регулярную решетку ячеек, каждая из которых может находиться в одном из конечного множества состояний, таких как 1 и 0.

Решетка может быть любой размерности. Для каждой ячейки определено множество ячеек, называемых окрестностью. К примеру, окрестность может быть определена как все ячейки на расстоянии не более 2 от текущей (окрестность фон Неймана ранга 2). Для работы клеточного автомата требуется задание начального состояния всех ячеек и правил перехода ячеек из одного состояния в другое. На каждой итерации, используя правила перехода и состояния соседних ячеек, определяется новое состояние каждой ячейки. Обычно правила перехода одинаковы для всех ячеек и применяются сразу ко всей решётке [6].

Джон Конвей заинтересовался проблемой, предложенной в 1940-х годах известным математиком Джоном фон Нейманом, который пытался создать гипотетическую машину, которая может воспроизводить сама себя. Джону фон Нейману удалось создать математическую модель такой машины с очень сложными правилами. Конвей попытался упростить идеи, предложенные Нейманом, и, в конце концов, ему удалось создать правила, которые стали правилами игры «Жизнь».

На рисунке 1.1 представлена иллюстрация примера игры «Жизнь».

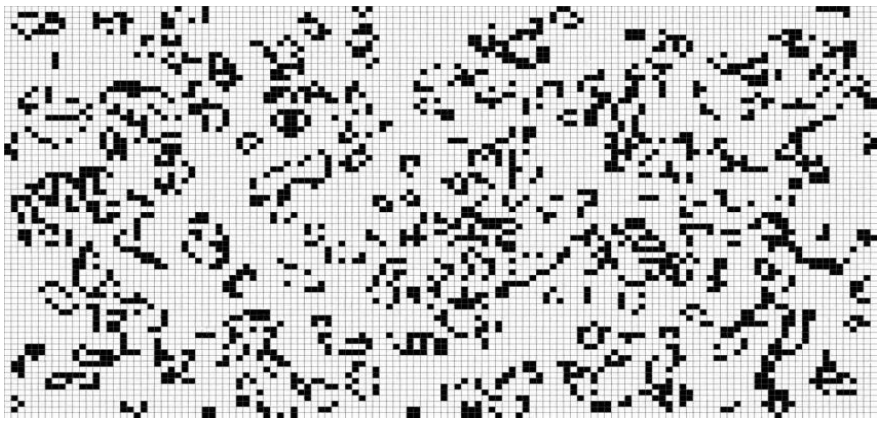


Рисунок 1.1 Игра «Жизнь»

Впервые описание этой игры было опубликовано в октябрьском (1970 год) выпуске журнала Scientific American, в рубрике «Математические игры» Мартина Гарднера.

В эту игру играют поодиночке. Возникающие в процессе игры ситуации очень похожи на реальные процессы, происходящие при зарождении, развитии и гибели колоний живых организмов. По этой причине «Жизнь» можно отнести к категории так называемых «моделирующих игр» – игр, которые в той или иной степени имитируют процессы, происходящие в реальной жизни. Уже довольно давно никто не играет в эту игру без использования компьютера, хотя это вполне возможно.

Основная идея игры состоит в том, чтобы, начав с какого-нибудь простого расположения фишек (организмов), расставленных по различным клеткам доски, проследить за эволюцией исходной позиции под действием «генетических законов» Конуэя, которые управляют рождением, гибелью и выживанием фишек. Конуэй тщательно подбирал свои правила и долго проверял их «на практике», добиваясь, чтобы поведение популяции было достаточно интересным, а главное, непредсказуемым.

Каждая клетка на бесконечном во все стороны поле имеет ровно восемь соседей. Рождаются и погибают клетки по следующим правилам:

- Если фишка имеет четырех или более соседей, то она умирает от перенаселенности (с этой клетки снимается фишка);
- Если фишка не имеет соседей или имеет ровно одного соседа, то она умирает от нехватки общения;
- Если клетка без фишки имеет ровно трех соседей, то в ней происходит рождение (на клетку кладется фишка);
- Если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить;
- Если не выполнено ни одно из перечисленных выше условий, состояние клетки не изменяется.

ГЛАВА 2
АГЕНТНОЕ МОДЕЛИРОВАНИЕ ИГРЫ «ЖИЗНЬ»

2.1 Принципы агентного моделирования в AnyLogic

AnyLogic поддерживает метод агентного моделирования и позволяет эффективно комбинировать этот метод с другими известными подходами. Существуют несколько «шаблонов» задания агентных систем, которые упрощают моделирование и включены в AnyLogic:

- Стандартная архитектура;
- Агентная синхронизация («Шаги»);
- Состояние (непрерывное или дискретное или ГИС), подвижности и анимация;
- Агентные связи (сети) и их взаимодействие друг с другом;
- Динамическое создание и уничтожение агентов.

Также в агентном моделировании используется пространства. Даже в моделях, где местоположение и движение агентов не так важны, пространство используется для визуализации агентов. AnyLogic поддерживает три вида пространства:

- Непрерывное трехмерное пространство;
- Дискретное пространство (сетка из ячеек);
- Пространство ГИС (Географическая Информационная система).

Остановимся более подробно на дискретном пространстве. Двумерное дискретное пространство представляет собой прямоугольный массив ячеек, полностью или частично занятых агентами. В одной ячейке может находиться не больше одного агента. Поддержка этого типа пространства в AnyLogic включает в себя возможности по распределению агентов по ячейкам, их перемещение в соседние или любые другие ячейки, определение того, какие агенты являются соседями (согласно выбранной модели соседства), нахождению свободных ячеек и т.д.

На рисунке 2.2 представлены размерности и типы соседства в дискретном пространстве.

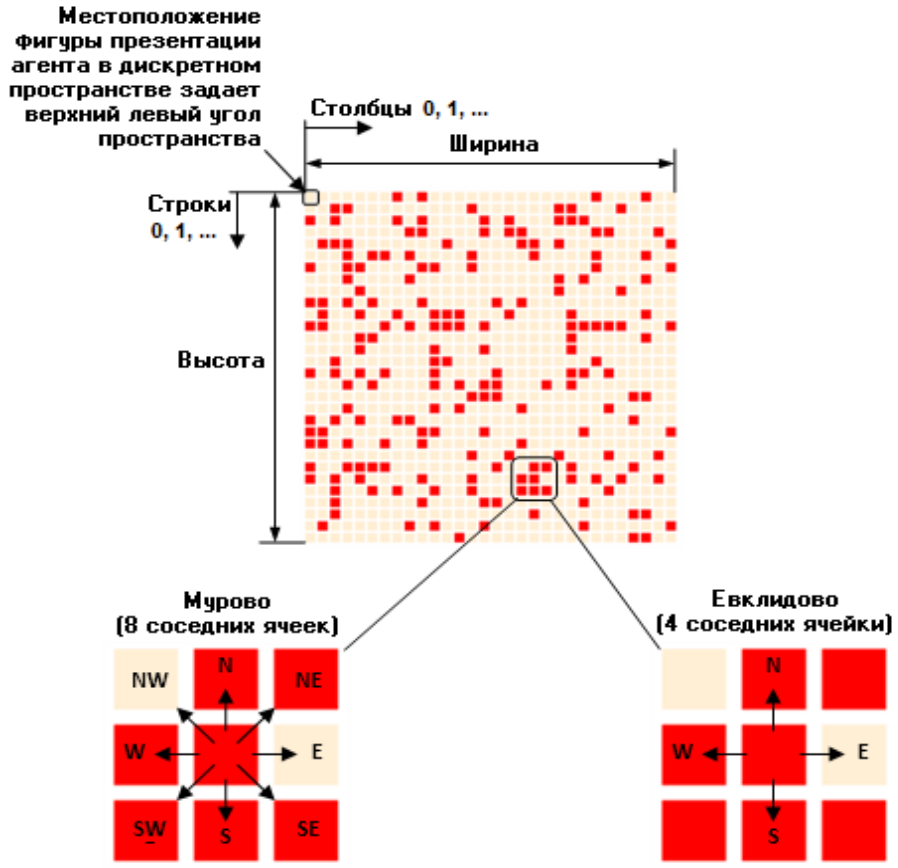


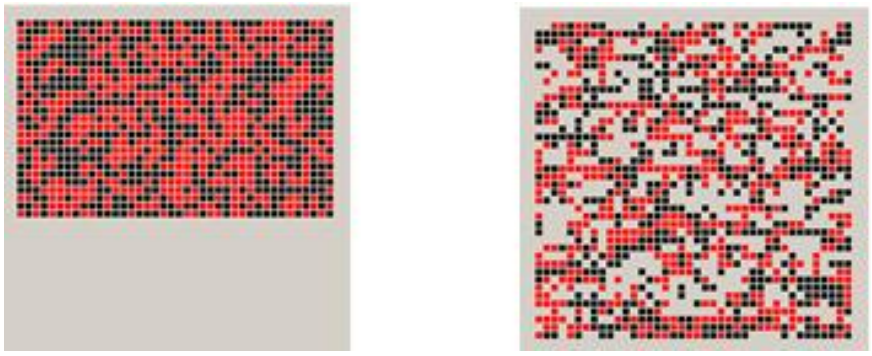
Рисунок 2.2 Размерности и типы соседства в дискретном пространстве

Также предлагаются функции, которые предоставляют информацию о размерностях пространства и размерах ячеек:

- double spaceHeight() - возвращает высоту пространства;
- double spaceWidth() - возвращает ширину пространства;
- double spaceCellHeight() - возвращает высоту ячейки;
- double spaceCellWidth() - возвращает ширину ячейки;
- int spaceColumns() - возвращает количество столбцов в пространстве;
- int spaceRows() - возвращает количество строк в пространстве.

Местоположение агента в дискретном пространстве определяется двумя координатами (строка, столбец) типа int. AnyLogic поддерживает два типа расположения агентов в дискретном пространстве:

- Случайное (агенты случайно распределены по ячейкам; в каждой ячейке может быть максимум один агент);
- Упорядоченное (агенты упорядоченно распределяются по ячейкам строка за строкой (слева направо и сверху вниз)).

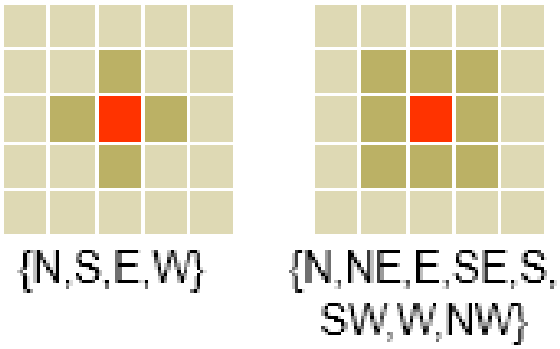


(*а*) (*б*)

Рисунок 2.2 Пример упорядоченного (*а*) и случайного (*б*) расположения агентов.

Рассмотрим также типы соседства у агентов. Тип соседства предназначен для того, чтобы определить какие ячейки будут считаться соседними. На рисунке 2.2 представлен пример упорядоченного и случайного расположения агентов. В AnyLogic предусмотрен специальный метод Agent[] getNeighbors(), возвращающий массив агентов. Есть две возможных модели соседства:

- Модель Мура (Moore) – соседними считаются 8 ячеек, располагающиеся к северу, югу, востоку, западу, северо-востоку, северо-западу, юго-востоку и юго-западу от данной (NORTH, SOUTH, EAST, WEST, NORTHEAST, NORTHWEST, SOUTHEAST, SOUTHWEST);
- Модель Евклида (Euclidian) – соседними считаются 4 ячейки, располагающиеся к северу, югу, востоку и западу от данной (NORTH, SOUTH, EAST и WEST).



(*а*) (*б*)

Рисунок 2.3 Иллюстрация евклидова (*а*) и мурова (*б*) типа соседства

AnyLogic также предоставляет возможность помещения агентов в пространство с некоторой функциональностью, отличной от имеющейся:

- Тороическое пространтсво (для этого необходимо переопределить метод);
- Дополнительная информация для каждой ячейки (реализовывается при помощи добавление специального для этой информации массива в агенте верхнего уровня (встроенного класса)).

2.2 Реализация игры «Жизнь» в AnyLogic

Для агентного моделирования игры «Жизнь» средствами AnyLogic было использовано дискретное пространство, упорядоченное расположение агентов и муров тип соседства. В процессе работы задавалось популяция с 2500 агентами и, следовательно, размерность пространства составила 500×500 (рисунок 2.4). Шаг между эпохами принимался равным 1 секунду (рисунок 2.5). Изменение цвет ячейки соответствовала изменению её состояния с живой на мертвую или, наоборот, и задавалось при помощи тернарной условной операции, введенной в специальную форму свойств ячеек «Fill color». В нашем случае живой ячейке соответствовал черный цвет, мертвой – белый (рисунок 2.6). Главное условие перехода между эпохами задавалось также в специальной форме onStep свойств всех ячеек, которая соответствует методу onStep(), соответствующего класса Cell в Java программе. Пробегая, всех агентов мы проверяем их на условия, соответствующие правилам игры «Жизнь» и в случае выполнения одного из этих условий состояние (живая или мертвая) клетки изменяется или остается прежней. В данном случае мы использовали специальную наследуемую функцию getNeighboors() из встроенного в AnyLogic класса Agent (рисунок 2.7)

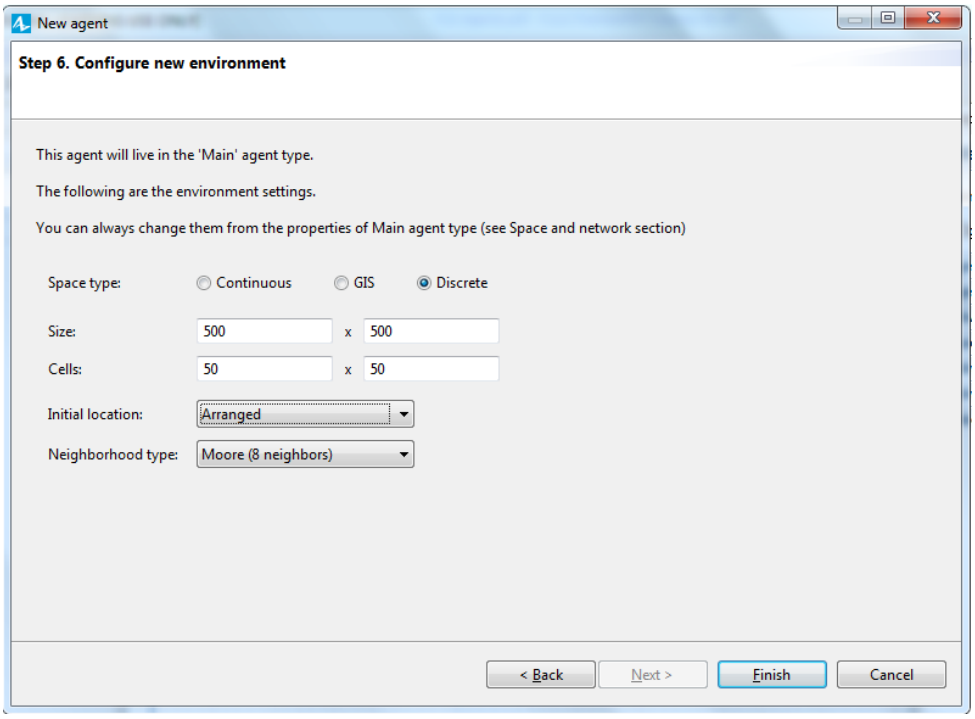


Рисунок 2.4 Задание типов пространства, расположения, соседства

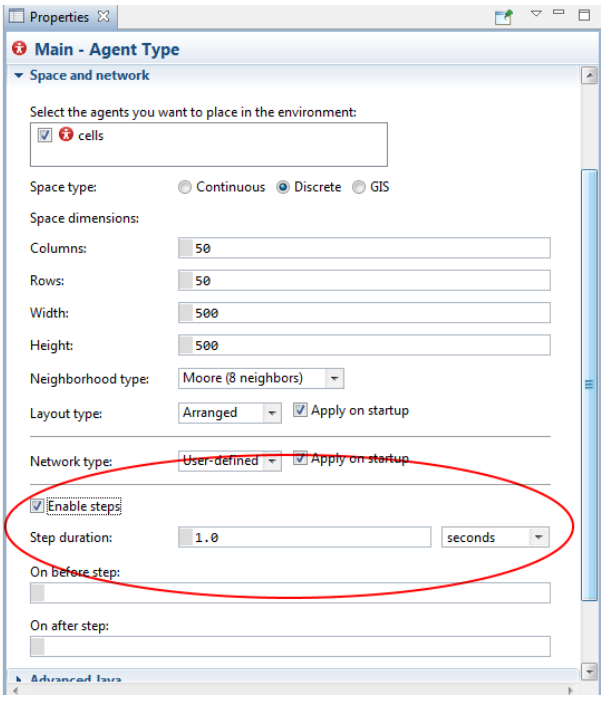


Рисунок 2.5 Задание времени перехода между эпохами

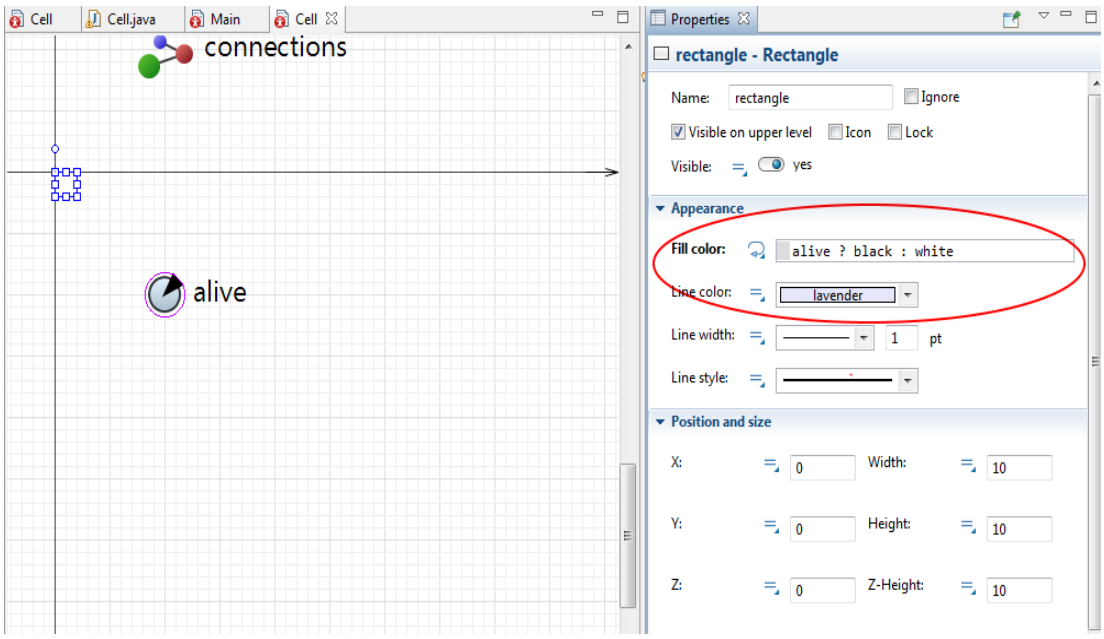


Рисунок 2.6 Задание изменения цвета (состояния) ячейки

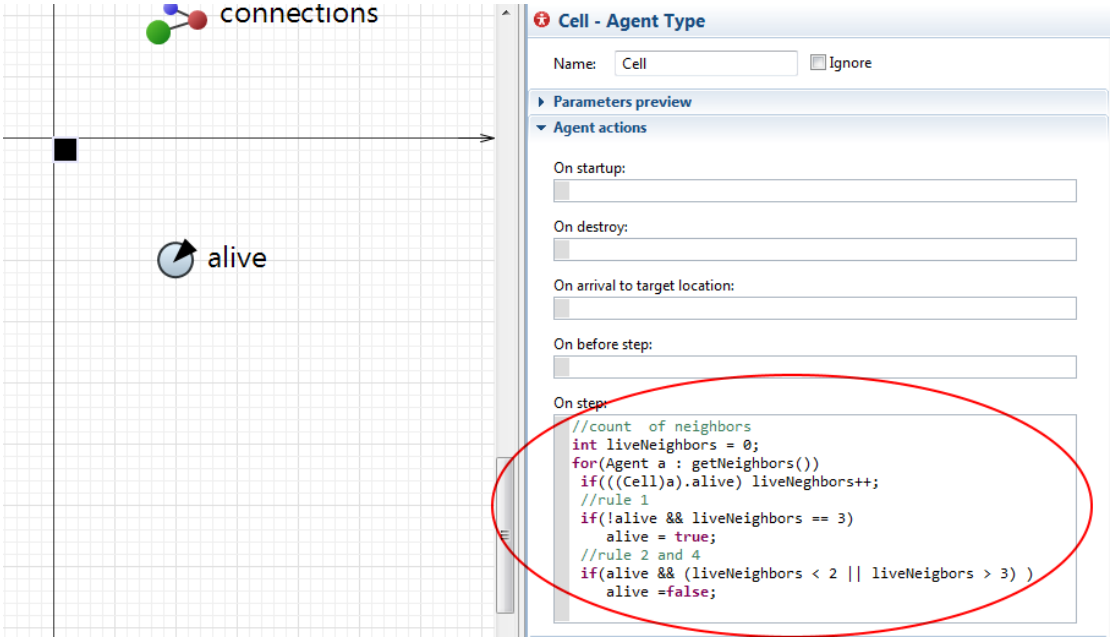


Рисунок 2.7 Задание главного условия перехода между эпохами

Для того, чтобы более тщательно проанализировать один из способов реализации агентного моделировании в AnyLogic, была составлена диаграмма классов, на основе которой становится более ясно строение отдельных объектов. Классом-ро-

дителем в исследуемой систему является класс Agent. В нем задается один из основных методов getNeighbors(), который возвращает текущих соседей для данного агента в зависимости от того, какой тип соседства выбран. Также этот класс содержит методы, предоставляющие информацию о размерах пространства и отдельной ячейки (данные методы представлены выше). Наследуемыми от данного класса Agent является классы Cell и Main. Cell содержит информацию о состоянии ячейки в текущей эпохе (живая или мертвая), которая представлена как отдельное поле alive типа Boolean. Значение true данного поля соответствует состоянию живая ячейка, а соответственно false – мертвая ячейка. Также данные класс содержит метод onStep(), который на каждом шаге вычисляет состояние текущей ячейки в зависимости от количества соседей. Таким образом в данном методе используется наследуемый метод getNeighboors() класса Agent. Main предназначен в большей степени для того, чтобы управлять процессами происходящими в модели. Одним из основных методов данного класса является doStart(), который дает начало работы вычисления для данного пространства. Этот метод запускает деятельность каждой ячейки в отдельности путем прохода по циклу по всем существующим. После окончания работы модели в класс Main предусмотрена функция onDestroy(), очищающая отработанную модель от всех агентов. Метод getPopulation() возвращает список всех текущих агентов, принадлежащих данному пространству (модели). Диаграмма классов представлена на рисунке 2.8.

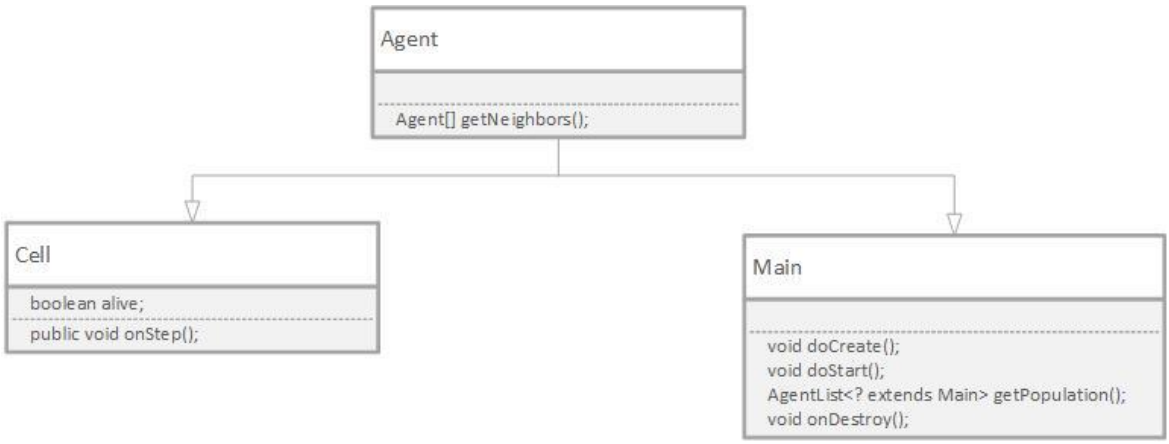


Рисунок 2.8 Диаграмма классов реализации агентного подхода моделирования в AnyLogic.

В результате всех взаимодействий в системе получилась динамически изменяющееся анимация игры Жизнь, соответствующее прямоугольному пространству. Отображение результата представлено на рисунке 2.9.

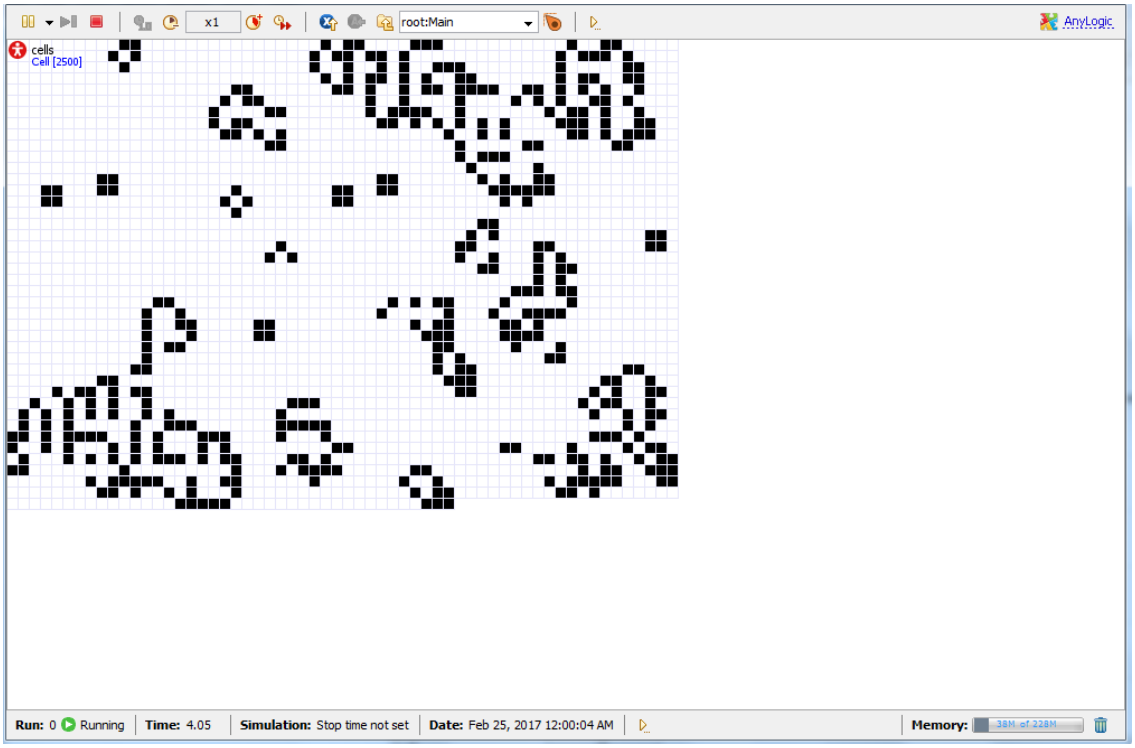


Рисунок 2.9 Отображение результата

2.3 Реализация игры «Жизнь» на Python.

При моделировании игры жизнь на языке программирования Python было предложено задействовать свою структуру в формирования агентного подхода в контексте объектно-ориентированных и графических возможностей языка. В нашем случае была выбрана библиотека процессно-ориентированной дискретно-событийной системы моделирования SimPy как среды, в которой будут происходить все события пространства игры «Жизнь». Библиотека matplotlib была выбрана для визуализации данных посредством двухмерной графики.

SimPy выпущен как open source проект под лицензией MIT в декабре 2002 года. Процессы в SimPy – это просто Python генераторы, которые используются для моделирования активных компонентов, например, таких как покупатели, транспортные средства или агенты. Моделирование может выполняться в режиме “as fast as possible”, в режиме реального времени (wall clock time) или в режиме ручного выполнения событий [8]. В нашем же случае SimPy предоставляет возможность

дискретизировать пространства, и ввести определенные единицы времени (эпохи), переходя через которые ячейки меняли свое состояние в соответствии с правилами игры «Жизнь».

Matplotlib – библиотека на языке программирования Python для визуализации данных двумерной (2D) графики (3D графика также поддерживается). Matplotlib является гибким, легко конфигурируемым пакетом, который вместе с NumPy, SciPy и IPython предоставляет возможности, подобные MATLAB. Пакет поддерживает многие виды графиков и диаграмм [9]. Для визуализации игры жизнь мы используем модуль matplotlib.pyplot., предоставляющий возможность визуализации в специализированном окне Figure (подобно MATLAB).

При реализации игры «Жизнь» на языке Python был введен класс Cell, который объединял в себе функции класса Agent и Cell в AnyLogic:

- ij_Neighbors(self) – возвращает коорината соседей у текущей клетки;
- count_neighbors(self,field) — возвращает количетсво живых соседей у текущей клетки;
- run(self, field) – изменяет состояние текущей ячейки исходя из количетсва соседей.

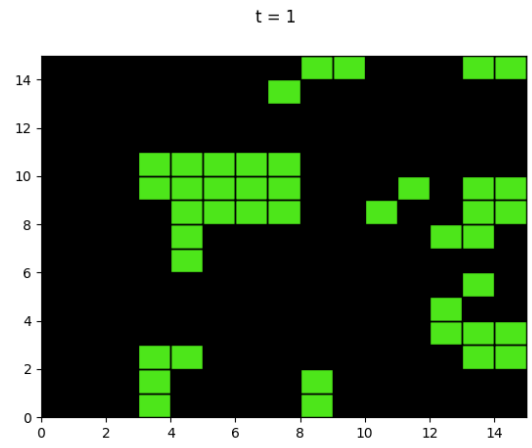
Таким образом функция onStep() в AnyLogic в нашем случае разбивается на две count_neighbors и run.

Также была введена специальная граничная функция next_epoch(), которая проверяет и выполняет необходимые операции над каждой ячейкой (переводит или не переводит в иное состояние). Конец выполнения данной функции сигнализирует среде (Environment) о переходе к следующей эпохе.

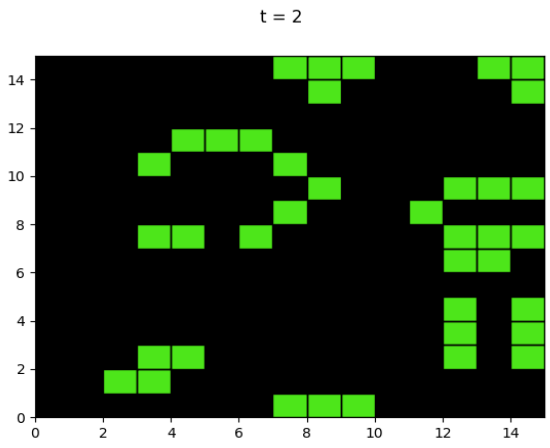
При визуализации ячеек живых или мертвых клеток использовались две функции:

- do_square(xy) – инициализирует вершины квадрата необходимыя для визуализации ячейки;
- plotField(field) – инициализирует область визуализации и строит нужные ячейки на плоскости при помощи функции do_square.

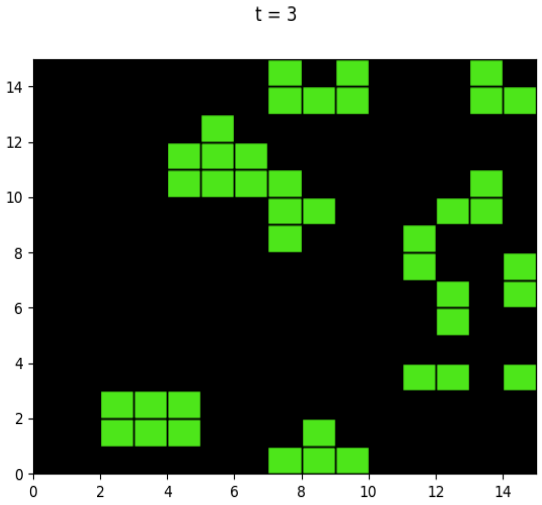
Также при реализации игры «Жизнь» на языке Python было реализовано тороическое пространство, то есть граничные противоположные ячейки связывались друг с другом отношениями соседства. Результат проиллюстрирован на рисунках 2.10



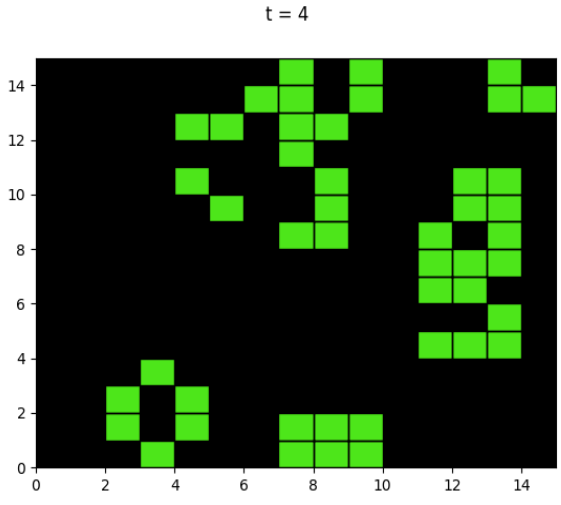
(*а*)



(*б*)



(*в*)



(*г*)

Рисунок 2.10 Результат реализация игры «Жизнь»: 1-ая эпоха (*а*); 2-ая эпоха (*б*); 3-я эпоха (*в*); 4-ая эпоха (*г*)

ЗАКЛЮЧЕНИЕ

На основании освоения полученной темы можно сделать вывод, что агентное моделирование на сегодняшний день является интересным, привлекательным и наиболее перспективным методом в моделировании имитационных процессов.

В первой главе курсового проекта приводятся сведения о различных понятиях, таких как имитационное моделирование, агентное моделлирование, агент и другие. Также рассказывается про программу для имитационного моделирования AnyLogic и про клеточный автомат на примере игры «Жизнь». Данная глава состоит преимущественно из теоретических основ разрабатываемой темы.

Во второй главе содержится самостоятельный анализ темы, где описываются основные этапы реализации проекта, производится обработка и анализ результатов опытно-эксперементальной работы. В данной главе приводятся сведения про принципы агентного моделирования, а также подробно излагается реализация игры «Жизнь» в AnyLogic и на Python.

В результате курсового проекта и полученных знаний были достигнуты цели и задачи, которые были поставлены.

В перспективе хотелось бы придумать, отыскать и реализовать некоторые расширения для правил игры, с которыми можно моделировать не только изменение численности популяции, но и, так называемый, естественный отбор внутри неё. При этом мы сможем наделить клетку возможностью примитивно думать и выживать в текущих условиях и, на основе этого, выделить некоторые образовавшиеся стаи, примерно совпадающих по типу поиска наиболее привлекательного положения для жизни. Это поможет наиболее ближе и точно подойти к моделированию процессов живого мира, его исследованию и изучению

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Anylogic – многоподходное имитационное моделирование [Электронный ресурс]. – Режим доступа: https://www.anylogic.ru/use-of-simulation. Дата доступа 28.02.2017.

2.Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: ние. Дата доступа 25.01.2017.

3.Киселева, М. В. К44 Имитационное моделирование систем в среде AnyLogic : учеб.-метод. пособие / М. В. Киселёва. Екатеринбург: УГТУ - УПИ, 2009. 88 с.

4.Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: . Дата доступа 25.03.2017.

5. Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/AnyLogic. Дата доступа 25.03.2017.

6. Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клеточный_автомат. Дата доступа 25.03.2017.

7.Каталевский, Л.Ю. Основы имитационного моделирования и системного анализа в управлении: учебное пособие; 2-е изд., перераб. и доп. / Д.Ю. Каталевский. — М.: Издательский дом «Дело» РАНХиГС, 2015. — 496 с.

8.Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/SimPy. Дата доступа 25.03.2017.

9. Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Matplotlib. Дата доступа 25.03.2017.

10. Сделано у нас[Электронный ресурс] / Имитационное моделирование нового поколения. – Режим доступа: https://sdelanounas.ru/blogs/13328/. Дата доступа 12.03.2017.

11.http://help.anylogic.ru/index.jsp?topic=/com.xj.anylogic.help/html/agentbased/Discrete%20space.html

12. Справка AnyLogic[Электронный ресурс]. – Режим доступа: http://help.anylogic.ru/index.jsp?topic=/com.xj.anylogic.help/html/agentbased/Discrete%20space.html. Дата доступа 12.04.2017.

ПРИЛОЖЕНИЕ А.

Код программы

Ниже приведен код алгоритма в Anylogic:

```
//count  of neighbors
int liveNeighbors = 0;
for(Agent a : getNeighbors())
 if(((Cell)a).alive) liveNeighbors++;
 //rule 1
 if(!alive && liveNeighbors == 3)
     alive = true;
 //rule 2 and 4
 if(alive && (liveNeighbors < 2 || liveNeighbors > 3) )
                alive = false;
```

Ниже приведен код алгоритма на Python:

```
from __future__ import print_function
from itertools import product
from copy import deepcopy
from random import randint
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches
# gelp library draw results
from matplotlib import pyplot as plt

#method for vizualization
def do_square(xy):
    verts = []
    codes = []
    for i in range(len(xy)):
        verts.extend([(xy[i][0],xy[i][1]),      # left, bottom
            (xy[i][0],xy[i][1] + 1),            # left, top
            (xy[i][0] + 1,xy[i][1] + 1),        # right, top
            (xy[i][0] + 1 ,xy[i][1]) ,          # right, bottom
            (xy[i][0],xy[i][1])])

        codes.extend([Path.MOVETO,
                Path.LINETO,
                Path.LINETO,
                Path.LINETO,
                Path.CLOSEPOLY,])
    return [verts,codes]
```

```
# help function for drawing result
def plot_field(field,env):
    xy = []
    fig = plt.figure()
    for i in field.keys():
        if field[i].state:
            xy.append([field[i].x,field[i].y])
    plot = fig.add_subplot(111)
    plot.set_facecolor('black')
    plot.set_xlim([0, 15])
    plot.set_ylim([0, 15])
    [verts, codes] = do_square(xy)
    path = Path(verts, codes)
    patch = patches.PathPatch(path, face-
color=(0.3,0.9,0.1),edgecolor='black',lw = 1)
    plot.add_patch(patch)
    fig.suptitle('t = {0}'.format(env.now))
    fig.savefig('t{0}.png'.format(env.now))
    plt.close(fig)

# simpy environment
import simpy

env = simpy.Environment()

class Cell(object):
    '''Cell of the space with 2 condition: state = True (cell is alive)
        and state = False (cell is death)'''

    def __init__(self, x, y, state = False):
            self.x = x
            self.y = y
            self.state = state

    def count_neighbors(self, field):
        ''' quantitty cells-neighboors '''
        length = len([1 for i in set(self.ij_Neighnoors()).intersec-
tion(set(field.keys())) if field[i].state])
        return length

    def run(self, field):
        countN = self.count_neighbors(field)
        if countN == 2:
            pass
        elif countN == 3:
            self.state = True
```

```
        else:
            self.state = False
    def ij_Neighnoors(self):
        i, j = self.x, self.y
        return [(i % 16, (j + 1) % 16), ((i - 1) % 16, (j + 1) % 16), ((i -
1) % 16, j % 16), ((i - 1) % 16, (j - 1) % 16),
                (i % 16, (j - 1) % 16), ((i + 1) % 16, (j - 1) % 16), ((i +
1) % 16, j % 16), ((i + 1) % 16, (j + 1) % 16)]

# my Field
field = {}

# initilalization all cells in field
for x, y in product(range(0, 15), range(0, 15)):
    field.update({(x, y): Cell(x, y)})
# certain alive cells
#init1 = Cell(0, 0, True)
#init2 = Cell(1, 0, True)
#init3 = Cell(2, 0, True)
#init4 = Cell(1, 2, True)
#init5 = Cell(2, 1, True)
#field.update({(0, 0): init1, (1, 0): init2, (2, 0): init3, (1, 2): init4,
(2, 1): init5})

# random alive cells
for i in range(50):
    init = Cell(randint(0,15), randint(0,15), True)
    field.update({(init.x,init.y): init})

# method for moving to next epoch
def next_epoch():
    while True:
        _field = deepcopy(field)
        for key in field.keys():
            __field = deepcopy(_field)
            field[key].run(__field)
        yield env.timeout(1)
        plot_field(field, env)

# simPy functions
env.process(next_epoch())
env.run(until = 10)
```