**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт вычислительной математики и информационных технологий

Кафедра анализа данных и технологий программирования

Направление подготовки: 09.03.03 – Прикладная информатика

 Профиль: Прикладная информатика

**КУРСОВАЯ РАБОТА**

**Программа для визуализации клеточного автомата по правилам “Игра жизнь**

Студент 1 курса

Группы 09-253

«\_\_\_» \_\_\_\_\_\_\_ 2023 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тазетдинов А.М

Научный руководитель

заместитель директора по воспитательной и социальной работе,

«\_\_\_» \_\_\_\_\_\_\_ 2023 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бандеров В.В

Казань-2023

**СОДЕРЖАНИЕ**

[Введение](#_Введение) 3

[ГЛАВА 1. Теоритическая часть](#_1._Теория_клеточных)  4

[1.1 Определение клеточных автоматов](#_1.1_Определение_клеточных) 4

[1.2 Виды клеточных автоматов](#_1.2_Виды_клеточных) 5

[1.3 Примеры клеточных автоматов](#_1.3_Примеры_клеточных)

[1.4 Применение клеточных автоматов](#_1.3_Применение_клеточных)  6

[Глава 2. Практическая часть](#_2._Разработка_программы)

[2.1 Постановка задачи](#_2.1_Постановка_задачи)

[2.2 Использованные технологии](#_2.2_Использованные_технологии)

[2.3 Моделирование системы](#_2.3_Моделирование_системы)

[2.4 Примеры работы программы](#_2.4_Пример_работы)

[ЗАКЛЮЧЕНИЕ](#_Заключение)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Список_использованных_источников)

ПРИЛОЖЕНИЯ

# Введение

В современном мире, где вычислительная мощность растет с каждым годом, мы обнаруживаем все больше и больше областей, где компьютеры и алгоритмы играют важную роль. Одной из таких областей является исследование и моделирование сложных систем, включая живые организмы и природные процессы. В этом контексте клеточные автоматы становятся одним из увлекательных объектов изучения. Благодаря несложной математической модели они позволяют моделировать различные биологические процессы, такие как развитие тканей, распространение заболеваний, эволюция и тд.

Целью данного проекта является разработка программы для визуализации клеточного автомата по правилам “Игра Жизнь”. Конкретные правила были выбраны как самые популярные для клеточных автоматов. Этот позволит пользователям исследовать и наблюдать интересные эволюционные паттерны, возникающие в клеточном автомате по правилам "Игра Жизнь".

Актуальность проекта. Данный проект актуален по нескольким причинам:

1. Обучение и популяризация: Визуализация клеточных автоматов является мощным инструментом для обучения и популяризации научных концепций. С помощью визуализации можно наглядно объяснить сложные идеи, показать эмерджентное поведение системы и продемонстрировать, как изменение локальных правил влияет на глобальное поведение.
2. Проектирование и оптимизация систем: Клеточные автоматы могут использоваться для проектирования и оптимизации различных систем. Например, в инженерии они могут применяться для моделирования течения жидкости или поведения материалов. Визуализация позволяет наблюдать за эволюцией системы, выявлять проблемные области и оптимизировать параметры.
3. Творчество и искусство: Визуализация клеточных автоматов имеет большой потенциал в области творчества и искусства. Они могут использоваться для создания уникальных и привлекательных визуальных эффектов, интерактивных инсталляций и генеративного искусства.

# 1. **Теория клеточных автоматов**

# 1.1 **Определение клеточных автоматов**

Клеточный автомат (англ. Cellular automaton)[1] - это математическая модель, которая состоит из сетки ячеек, каждая из которых может находиться в одном из нескольких состояний. Клеточные автоматы развиваются в дискретных временных шагах в соответствии с заранее определенными правилами, которые определяют, как изменяются состояния клеток на основе состояний их соседей.

Основные компоненты клеточного автомата включают:

1. Сетка клеток: Клетки образуют сетку, которая может быть одномерной, двумерной или даже высокомерной, в зависимости от конкретной модели клеточного автомата
2. Состояния клеток: Каждая клетка может находиться в определенном состоянии. Состояния могут быть дискретными (например, "включено" или "выключено") или непрерывными (например, числовые значения).
3. Правила: Правила определяют, как изменяются состояния клеток на основе состояний их соседей. Правила могут быть простыми или сложными и могут зависеть от текущего состояния клетки и состояний её соседей.

# 1.2 Виды клеточных автоматов

Клеточные автоматы могут быть классифицированы по различным критериям. Вот некоторые распространенные классификации клеточных автоматов:

1. **По размерности**
   * Одномерные клеточные автоматы: Каждая ячейка имеет две соседние ячейки.
   * Двумерные клеточные автоматы: Каждая ячейка имеет восемь соседних ячеек.
   * Трехмерные клеточные автоматы: Каждая ячейка имеет 26 соседних ячеек
2. **По состояниям ячеек**
   * Двоичные клеточные автоматы: Каждая ячейка может принимать одно из двух состояний (например, 0 или 1).
   * Множественные состояния клеточных автоматов: Каждая ячейка может принимать одно из нескольких состояний
3. **По правилам перехода**
   * Элементарные клеточные автоматы: Используют простые правила перехода, которые зависят только от состояний самой ячейки и ее соседей.
   * Комплексные клеточные автоматы: Используют более сложные правила перехода, которые могут зависеть от множества соседей и их состояний.
4. **По типу сетки**
   * Регулярные клеточные автоматы: Используют регулярные сетки, где каждая ячейка имеет одинаковое количество соседей
   * Нерегулярные клеточные автоматы: Используют нерегулярные сетки, где количество соседей может различаться для каждой ячейки
5. **По типам поведения**
   * **Статические клеточные автоматы**: В таких автоматах состояния ячеек остаются неизменными или изменяются только в пределах некоторого ограниченного цикла. Такие автоматы не проявляют сложных динамических свойств и могут быть использованы для представления статических паттернов
   * **Периодические клеточные автоматы**: В этих автоматах состояния ячеек изменяются циклически, образуя периодические структуры или паттерны. Паттерны могут повторяться после определенного числа шагов и продолжать свое поведение в том же цикле.
   * **Хаотические клеточные автоматы**: В таких автоматах состояния ячеек изменяются хаотически и предсказуемо, поскольку даже небольшие изменения в начальных условиях могут привести к значительным изменениям в дальнейшем развитии. Хаотические клеточные автоматы могут проявлять сложные и непредсказуемые динамические свойства.
   * **Осциллирующие клеточные автоматы**: В таких автоматах состояния ячеек изменяются периодически, но не образуют постоянных паттернов или периодических структур. Состояния колеблются между несколькими различными состояниями
   * **Клеточные автоматы с самоорганизацией**: В этих автоматах состояния ячеек могут изменяться таким образом, чтобы образовывать сложные, самоорганизующиеся структуры. Они способны к эмерджентному поведению, где сложные свойства системы возникают из простых правил взаимодействия между ячейками.

# 1.3 Примеры клеточных автоматов

**Элементарные клеточные автоматы**

Простейшим нетривиальным клеточным автоматом является одномерный клеточный автомат с двумя возможными состояниями. В таком автомате соседями клетки являются смежные с ней клетки. Такие автоматы называются элементарными. На каждом шаге времени, три клетки (центральная и её соседи) определяют одну из 8 возможных комбинаций состояний. Затем, на основе анализа текущей тройки, принимается решение о том, каким будет состояние центральной клетки на следующем шаге (белой или чёрной). Всего существует 256 возможных правил для таких автоматов, которые кодируются согласно коду Вольфрама - стандартному соглашению, предложенному Вольфрамом. Его обычно записывают в виде таблицы, где под текущим состоянием указано состояние центральной клетки после шага эволюции.

Интересными правилами являются правила с номерами 30 (Рисунок 1.1) и 110 (Рисунок 1.2) . На двух изображениях ниже представлены эволюции автоматов, соответствующих этим правилам. Начальное условие для каждого автомата - одна центральная клетка чёрная, а остальные клетки белые. По оси Y откладывается дискретное время, а по оси X - состояния клеток автомата.

Правило 30:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Текущее состояние** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| **Новое состояние центральной клетки** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

Правило 110

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Текущее состояние** | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| **Новое состояние центральной клетки** | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

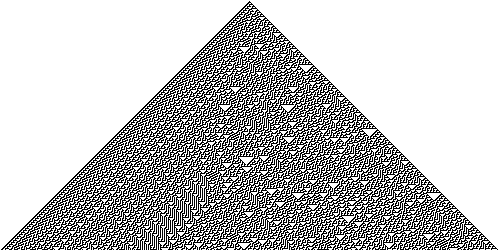


Рисунок 1.1 - эволюция правила 30

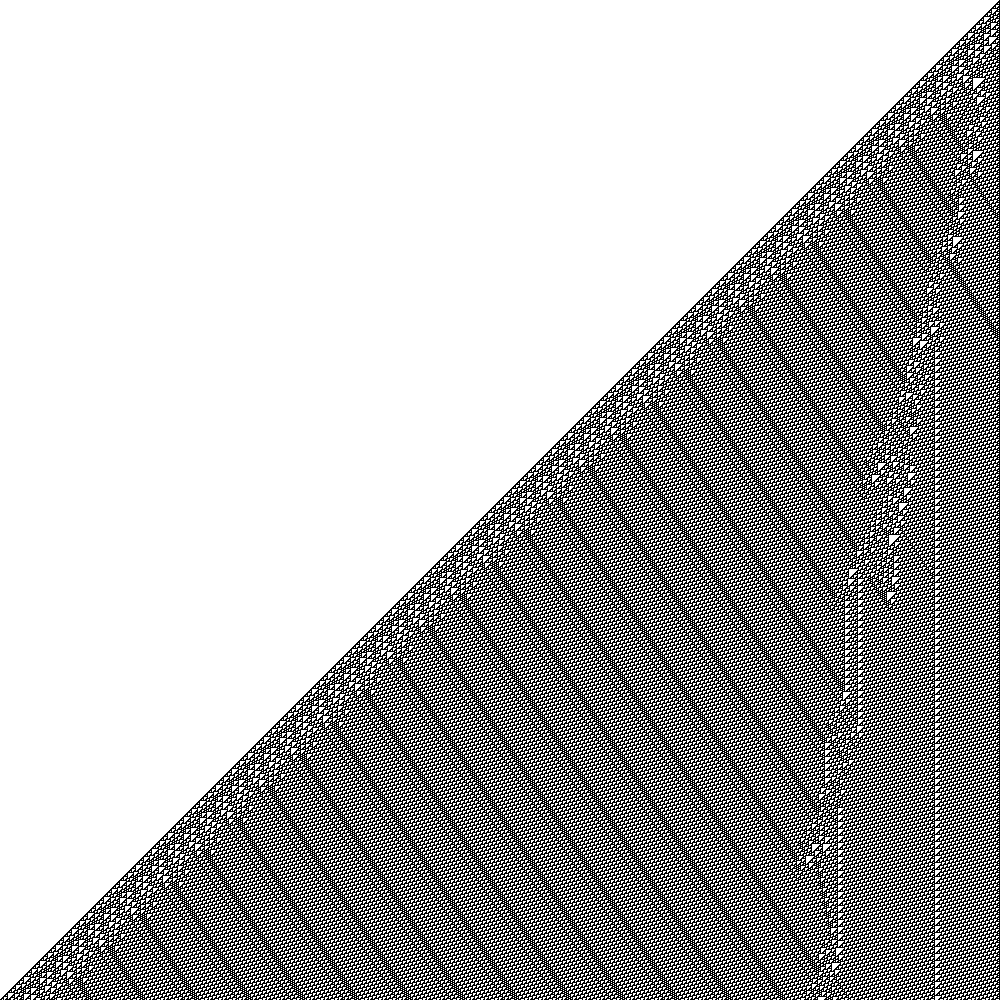


Рисунок 1.2 – Эволюция правила 110

**Клеточные автоматы на двумерной решётке**: Этот вид клеточных автоматов работает на двумерной решётке клеток. Каждая клетка может принимать одно из нескольких состояний, и обновление состояний происходит параллельно в дискретные моменты времени. Классическим примером двумерных клеточных автоматов является "Игра жизнь"[2] Джона Конвея. Она обладает следующими правилами:

1. Живая клетка, которая имеет меньше двух живых соседей, умирает от одиночества.
2. Живая клетка, которая имеет два или три живых соседа, продолжает жить и на следующем шаге.
3. Живая клетка, которая имеет больше трёх живых соседей, умирает от перенаселения.
4. Мёртвая клетка, которая имеет ровно три живых соседей, оживает.

Таким образом, состояние каждой клетки на следующем шаге определяется по состоянию её соседей на текущем шаге. Игра продолжается, пока не происходят следующие события: все клетки умирают, не остаётся новых живых клеток, или популяция входит в циклическое состояние, в котором конфигурации повторяются. Благодаря такому набору правил в этом клеточном автомате могут формироваться различные фигуры, для которых даже разработали свою классификацию:

* **Устойчивые фигуры**: фигуры, которые остаются неизменными
* **Долгожители**: фигуры, которые долго меняются, прежде чем стабилизироваться
* **Периодические фигуры**: фигуры, у которых состояние повторяется через некоторое число поколений, большее 1;
* **Двигающиеся фигуры**: фигуры, у которых состояние повторяется, но с некоторым смещением. Одной из первых таких фигур является Планер (Рисунок 1.3)
* **Ружья**: фигуры с повторяющимися состояниями, дополнительно создающие движущиеся фигуры. Первым “ружьем” можно считать планерное ружье Госпера (Рисунок 1.4)
* **Паровозы**: двигающиеся фигуры с повторяющимися состояниями, которые оставляют за собой другие фигуры в качестве следов;
* **Пожиратели**: устойчивые фигуры, которые могут пережить столкновения с некоторыми двигающимися фигурами, уничтожив их;
* **Отражатели**: устойчивые или периодические фигуры, способные при столкновении с ними движущихся фигур поменять их направление;
* **Размножители**: конфигурации, количество живых клеток в которых растёт как квадрат количества шагов;

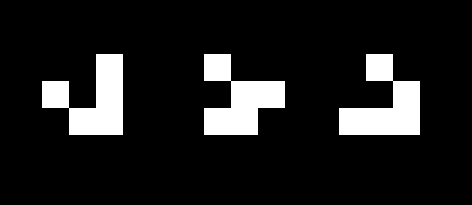


Рисунок 1.3 – все возможные состояния планера

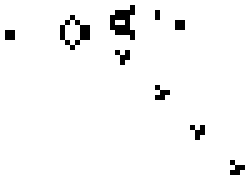


Рисунок 1.4 – одно из состояний планерного ружья Госпера

**Многозначные клеточные автоматы**: В отличие от элементарных клеточных автоматов, в которых клетки могут принимать только два состояния, многозначные клеточные автоматы позволяют клеткам иметь большее число возможных состояний. Это расширяет возможности моделирования и позволяет решать более сложные задачи.

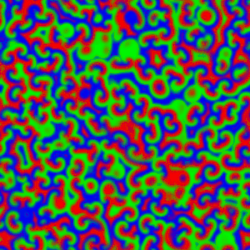


Рисунок 1.5 – Применение многозначного клеточного автомата для моделирования реакции Белоусова — Жаботинского в тонком слое жидкости в чашке Петри.

# 1.4 Применение клеточных автоматов

Клеточные автоматы применяются для:

1. **Моделирования природных явлений**: клеточные автоматы могут использоваться для моделирования различных природных процессов, таких как распространение пожаров, диффузия веществ в жидкостях, распространение эпидемий и т.д. Они позволяют исследовать взаимодействия между элементами системы и изучать эмерджентное поведение.
2. **Искусственная жизнь**: клеточные автоматы широко используются для создания моделей искусственной жизни, где клетки могут имитировать поведение живых организмов. Это позволяет исследовать эволюцию, развитие, поведение и динамику популяций в виртуальных экосистемах.
3. **Криптография**: клеточные автоматы могут быть использованы в криптографии для создания сложных шифровальных алгоритмов. Они обеспечивают высокую степень случайности и нелинейности, что делает их полезными для защиты информации.
4. **Обработка изображений и видео**: клеточные автоматы могут применяться для обработки изображений и видео, включая сегментацию объектов, фильтрацию шума, сжатие данных и другие операции обработки.
5. **Исследование сложных систем**: клеточные автоматы позволяют исследовать сложные системы с простыми правилами взаимодействия. Они могут быть использованы для изучения физических систем, экономических моделей, социальных сетей, эволюционной биологии и других дисциплин.
6. **Искусство и генерация контента**: клеточные автоматы используются в искусстве и генерации контента для создания уникальных и интересных паттернов, текстур, анимаций и музыки. Они могут служить инструментом для творческого выражения и создания новых форм искусства.

# 2. Практическая часть

# 2.1 Постановка задачи

Необходимо разработать программу, которая позволит пользователям создавать собственные игровые сценарии, определять начальные расстановки клеток и просматривать процесс эволюции. Программа должна обладать следующими основными функциями:

1. Создание и редактирование сценариев
   * Возможность создания нового сценария и сохранения его для последующего использования
   * Возможность редактирования существующих сценариев, изменение расстановки клеток и правил эволюции
   * Предоставление пользователю набора инструментов для добавления, удаления и перемещения клеток внутри игрового поля
2. Визуализация эволюции
   * Отображение игрового поля с текущим состоянием клеток
   * Анимация процесса эволюции, показывающая изменение состояния клеток по времени
   * Возможность управления скоростью анимации (пауза, ускорение, замедление)
3. Управление эволюцией
   * Реализация правил "Игры Жизнь" для определения следующего состояния клеток на основе текущего состояния и количества соседей
   * Возможность запуска/остановки эволюции в произвольный момент времени
4. Взаимодействие с пользователем
   * Интуитивный и простой в использовании интерфейс, позволяющий пользователям легко создавать и изменять игровые сценарии, а также управлять эволюцией
   * Возможность сохранения и загрузки сценариев для последующего использования

# 2.2 Использованные технологии

В качестве основы для проекта была выбрана технология Windows Presentation Foundation. Windows Presentation Foundation (WPF)[3] - это технология, разработанная компанией Microsoft для создания графических пользовательских интерфейсов в приложениях под управлением операционных систем Windows. Она обеспечивает богатый набор инструментов и функциональность для создания современных и привлекательных пользовательских интерфейсов.

Применение WPF в данном проекте обеспечит возможность создания интерактивной и интуитивно понятной программы для визуализации клеточного автомата по правилам "Игра Жизнь". С использованием WPF можно будет создать графический интерфейс, который позволит пользователям взаимодействовать с игровым полем и управлять его состоянием.

# 2.3 Моделирование системы

Прежде всего, необходимо было определить и описать классы для работы с полем игры, его визуализацией и интерфейсом программы. Я остановился на следующем наборе классов.

**GameLifeArea**

Класс, содержащий информацию о текущем состоянии поля и методы работы с ним. Методы:

* public CellType CellAt(int x, int y) – возвращает состояние клетки по координате x и y
* public void SetCell(CellType cell, int x, int y) – устанавливает состояние клетки по координатам
* public bool IsAlive(int x, int y) – определяет, жива ли клетка по заданным координатам
* public int NeighborsCount(int x, int y) – возвращает количество соседей для клетки по заданным координатам
* public void SetWidth(int width) – задает ширину поля
* public void SetHeight(int height) – задает высоту поля
* public void UpdateArea() – функция для очистки поля от промежуточных состояний клеток после применения алгоритма
* public void Clear() – очистка поля от живых клеток
* public void FillRandom() – заполнение поля случайным количеством живых клеток
* private void updateNeighbors(int x, int y) – обновление данных об соседях клетки в кеше
* private (int, int) checkCoords(int x, int y) – нормализация координат
* private void initArea(int width, int height) – инициализация массивов для поля и соседей размеру поля

**GameLife**

В этом классе содержится реализация правил игры жизнь. Содержит в себе экземпляр класса GameLifeArea. Методы:

* public void Update() – Эволюция текущего состояния поля согласно правилам игры жизнь

**GameLifeUi**

Класс, являющийся кастомным виджетом[4] для отрисовки текущего состояния поля. Принял решение использовать именно такой способ отрисовки, так как он оказался самым быстрым. Имеет в себе функционал камеры, то есть позволяет наблюдать за конкретным участком поля в приближении:

* public void Draw() – отрисовка поля исходя из размеров одной клетки и текущего положения камеры относительно начала поля. Живые клетки рисуются белым цветом
* private void GameLifeUi\_MouseDown(object sender, MouseButtonEventArgs e) – обработчик события нажатия мыши. В этом методе рассчитываются координаты нажатой клетки и передаются в главный класс программы
* private void GameLifeUi\_MouseMove(object sender, MouseEventArgs e) – обработчик события движения мыши. Нужен для реализации функционала движения камеры.
* private void GameLifeUi\_PreviewMouseWheel(object sender, MouseWheelEventArgs e) – обработчик события прокрутки колесика мыши. Нужен для реализации приближения и отдаления камеры.

**MainWindow**

Класс главного окна программы. В нем находиться описание логики всех элементов окна(кнопок, крутилок и тд). Инициализирует классы GameLife и GameLifeUi. Методы:

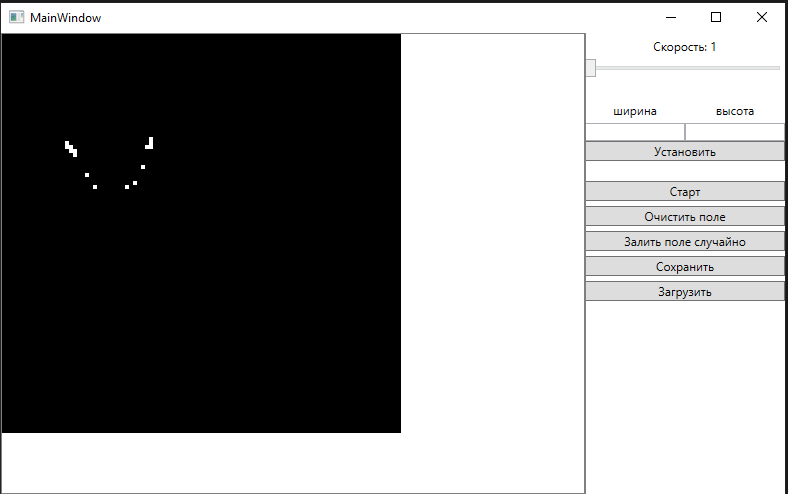
* private void GameCellClicked(MouseEventArgs e, int x, int y) – обработчик события нажатия на клетку. Применяет обновление только если в данный момент не запущена симуляция.
* private void setSpeed\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) – обработчик события изменения состояния слайдера скорости. Изменяет скорость симуляции от 10 миллисекунд до 1 секунды с шагом 10мс.
* private void updateSize\_Click(object sender, RoutedEventArgs e) – обработчик события нажатия на кнопку изменения размера. Изменяет размер игрового поля согласно данным из соответсвующих полей ввода.
* private void btnStart\_Click(object sender, RoutedEventArgs e) – обработчик события нажатия на кнопку начала/окончания процесса симуляции.
* private void Window\_SizeChanged(object sender, SizeChangedEventArgs e) – обработчик события изменения размеров окна. Перерисовывает поле чтобы оно соответсвовало размерам окна.
* private void timer\_Tick(object sender, EventArgs e) – обработчик события таймера. Данный таймер является основным циклом симуляции игры. При получении данного события обновляется и перерисовывается игровое поле.
* private void btnClear\_Click(object sender, RoutedEventArgs e) – обработчик нажатия на кнопку очистки поля. Запускает очистку только если симуляция в данный момент не запущена.
* private void Window\_KeyDown(object sender, KeyEventArgs e) – обработчик события нажатия на клавишу клавиатуры. При нажатии на пробел останавливает или запускает симуляцию. При нажатии на delete очищает поле.
* private void startStop() – Запускает или останавливает симуляцию.
* private void btnFillRandom\_Click(object sender, RoutedEventArgs e) – Если симуляция не запущена, заполняет поле случайным количеством живых клеток(GameArea.FillRandom)
* private void btnSave\_Click(object sender, RoutedEventArgs e) – Обработчик кнопки сохранения состоняния поля в файл. Если симуляция не запущена, открывает диалоговое окно с выбором имя файла, после чего записывает иформацию о поле в данный файл.
* private void btnLoad\_Click(object sender, RoutedEventArgs e) – Обработчик события нажатия на кнопку загрузки состояния поля из файла. Если симуляция не запущена, открывает диалоговое с выбором файла, считывает и валидирует его содержимое.

# 2.4 Пример работы программы

После запуска программы можно увидеть ее интерфейс. Слева находиться виджет для визуализации поля. Изначальный размер поля 100 на 100 клеток. Справа находятся основные элементы управления.



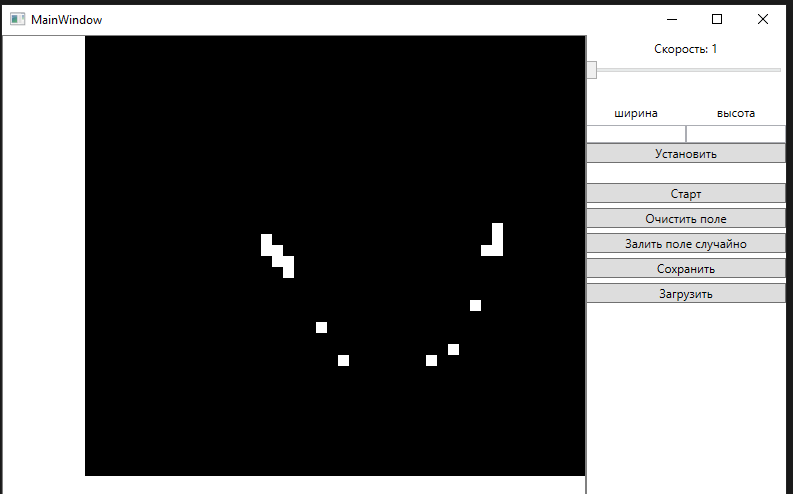
При нажатии и передвижении по полю левой кнопкой мыши клетки под мышью становятся живыми: 

При нажатии на живые клетки правой кнопкой мыши можно снова сделать их мертвыми: 

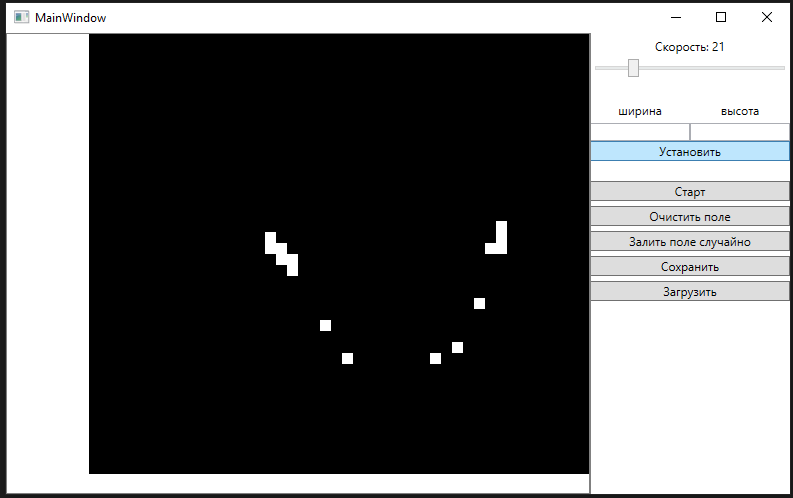
При нажатии и удержании средней кнопки мыши можно перемещать камеру в пределах виджета:



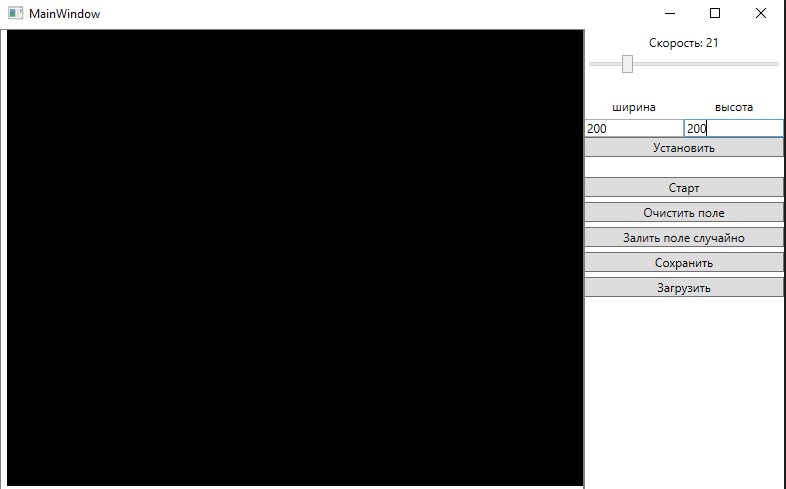
Прокруткой колесика мыши можно увеличивать размер клеток(приближать и удалять камеру):



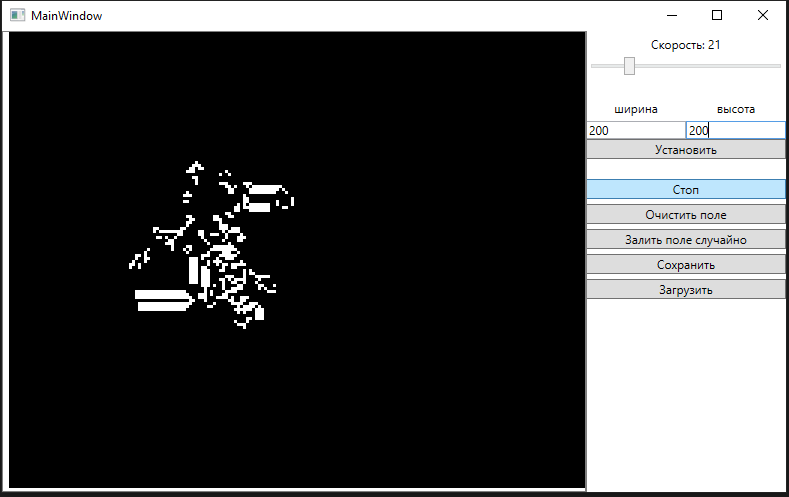
Прокруткой слайдера справа можно изменить скорость симуляции. Чем меньше значение слайдера, тем быстрее будет происходить симуляция:



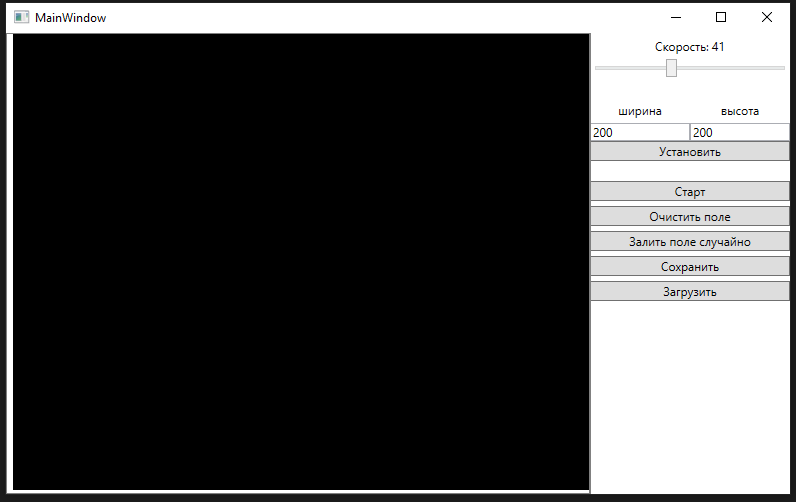
Нажатием кнопки установить можно изменить размер поля согласно полям ввода “ширина” и “высота”. При этом все данные с предыдущего поля будут стерты:



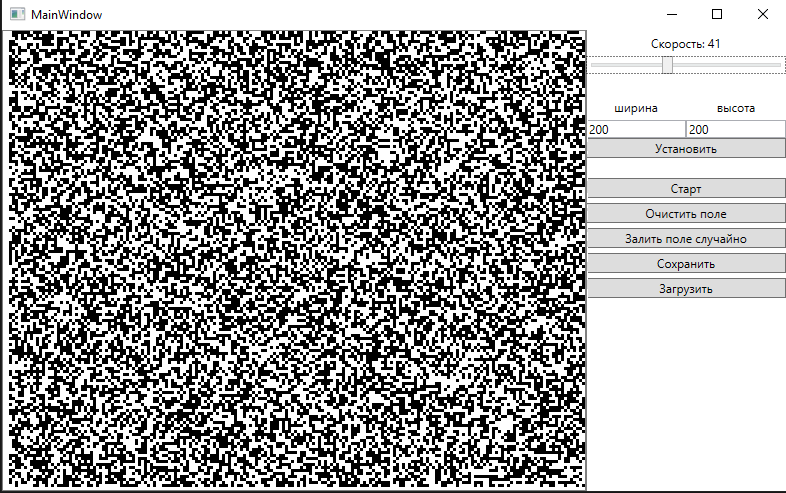
Нажатием кнопки старт можно запустить процесс симуляции. При этом ее текст изменится на “Стоп”. Этой же кнопкой можно будет остановить симуляцию:



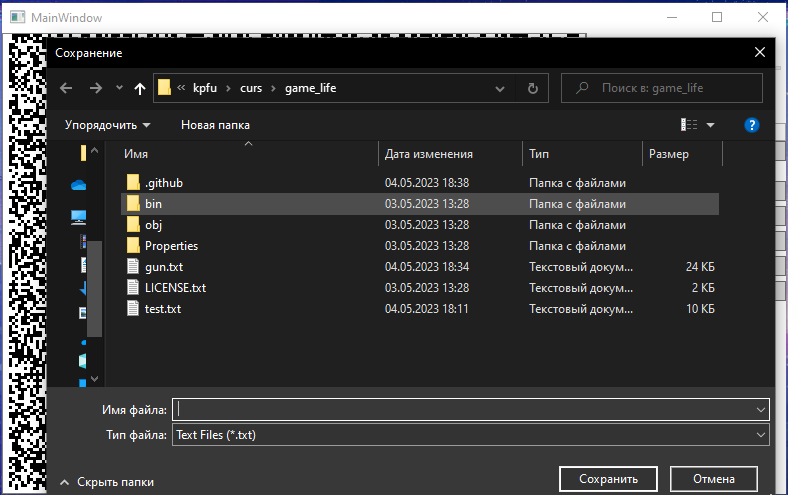
Нажатие кнопки “Очистить поле” очищает поле от живых клеток:



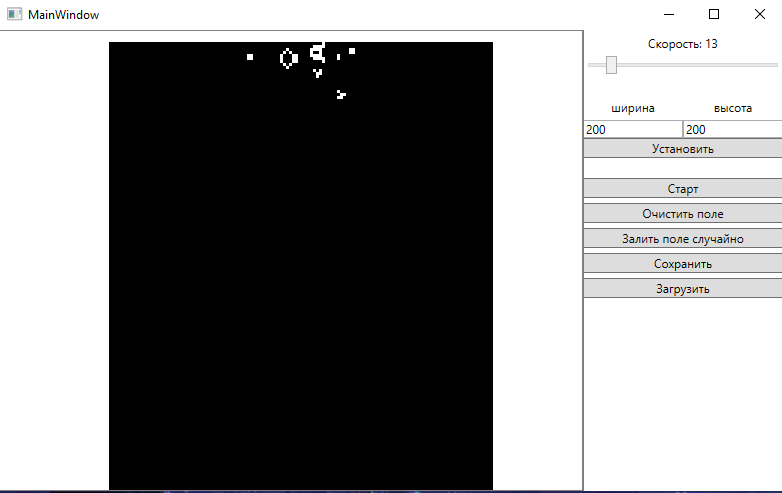
Нажатие кнопки “Залить поле случайно” заполняет поле в случайном порядке случайным количеством живых клеток:



Нажатием на кнопку сохранить можно записать текущее состояние поля в файл. При этом открывается всплывающее окно с выбором места и названия файла для сохранения



Нажатием на кнопку “Загрузить” можно загрузить состояние поля из файла.



# Заключение

В заключение данной курсовой работы можно сделать вывод о важности и актуальности разработки программ для визуализации клеточных автоматов. Использование клеточных автоматов в различных областях науки и техники, таких как физика, биология, компьютерная графика и моделирование, позволяет исследовать сложные системы и принимать обоснованные решения.

В результате выполнения данного проекта была разработана программа, способная визуализировать клеточные автоматы, предоставляя исследователям и разработчикам удобный и интуитивно понятный инструмент для анализа и визуализации динамических процессов. Программа предоставляет широкий набор функций и возможностей, позволяющих визуализировать их состояние в режиме реального времени, cохранять и анализировать результаты моделирования.

При разработке программы был использован современный фреймворк Microsoft WPF, что обеспечивает удобство использования и эффективную работу с программой. Были учтены основные принципы модульности, расширяемости и удобного взаимодействия с пользователем.

Дальнейшее развитие данной программы может включать в себя расширение функциональности, добавление новых типов клеточных автоматов, улучшение графического интерфейса и оптимизацию производительности. Программа может быть полезна для учебных целей, научных исследований, а также для создания визуальных эффектов и игр.

# Список использованных источников

# 1) Клеточный автомат // Wikipedia [Электронный ресурс] URL: <https://ru.wikipedia.org/wiki/Клеточный_автомат> Дата обращения 05.05.2023

2)Игра жизнь // Wikipedia [Электронный ресурс] URL: [https://ru.wikipedia.org/wiki/Игра\_«Жизнь»](https://ru.wikipedia.org/wiki/Игра_) Дата обращения 05.05.2023

3)Библиотека фигур клеточного автомата “Игра Жизнь” // Life Lexicon [Электронный ресурс] <https://conwaylife.com/ref/lexicon/lex.htm> Дата обращения 05.05.2023

4)Натан, А. WPF 4. Подробное руководство: учебное пособие / А. Натан – Санкт-Петербург – Москва : Символ-Плюс, 2011. – С. 45 – 48.

# 5) Using DrawingVisual Objects // Microsoft docs [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/using-drawingvisual-objects?view=netframeworkdesktop-4.8>

Дата обращения 10.05.2023

# Приложения

Приложение 1. MainWindow.xaml:

<Window x:Class="game\_life.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:game\_life"

mc:Ignorable="d"

Title="MainWindow" Height="500" Width="800"

SizeChanged="Window\_SizeChanged"

KeyDown="Window\_KeyDown">

<Grid Name="mainGrid">

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"></ColumnDefinition>

<ColumnDefinition Width="200"></ColumnDefinition>

</Grid.ColumnDefinitions>

<StackPanel Name="controlUi" Grid.Row="0" Grid.Column="1">

<Label Name="speedStatus" HorizontalAlignment="Center">Скорость: 1</Label>

<Slider Name="setSpeed" Minimum="1" Maximum="100" IsSnapToTickEnabled="True" ValueChanged="setSpeed\_ValueChanged" SmallChange="1" Value="1"/>

<Grid Margin="0,20,0,0">

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Label Grid.Row="0" Grid.Column="0" HorizontalAlignment="Center">ширина</Label>

<TextBox Name="areaWidth" Grid.Row="1" Grid.Column="0"></TextBox>

<Label Grid.Row="0" Grid.Column="1" HorizontalAlignment="Center">высота</Label>

<TextBox Name="areaHeight" Grid.Row="1" Grid.Column="1"></TextBox>

<Button Name="updateSize" Click="updateSize\_Click" Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2" Focusable="False">Установить</Button>

</Grid>

<Button Name="btnStart" Click="btnStart\_Click" Margin="0,20,0,0" Focusable="False">Старт</Button>

<Button Name="btnClear" Content="Очистить поле" Click="btnClear\_Click" Margin="0,5,0,0" Focusable="False"/>

<Button Name="btnFillRandom" Content="Залить поле случайно" Click="btnFillRandom\_Click" Margin="0,5,0,0" Focusable="False"/>

<Button Name="btnSave" Content="Сохранить" Click="btnSave\_Click" Margin="0,5,0,0" Focusable="False"/>

<Button Name="btnLoad" Content="Загрузить" Click="btnLoad\_Click" Margin="0,5,0,0" Focusable="False" />

</StackPanel>

</Grid>

</Window>

Приложение 2. MainWindow.xaml.cs:

using System;

using System.Windows;

using System.Windows.Media;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Threading;

using System.Diagnostics;

using System.Collections.Generic;

using Microsoft.Win32;

using System.IO;

using System.Linq;

namespace game\_life

{

public partial class MainWindow : Window

{

private GameLife game;

private GameLifeUi gameWidget;

private bool isRunning = false;

private DispatcherTimer timer;

public MainWindow()

{

InitializeComponent();

game = new GameLife(100, 100);

gameWidget = new GameLifeUi(game.Area, Brushes.White, Brushes.Black);

mainGrid.Children.Add(gameWidget);

Grid.SetColumn(gameWidget, 0);

Grid.SetRow(gameWidget, 0);

gameWidget.CellUpdate = GameCellClicked;

timer = new DispatcherTimer(DispatcherPriority.Render);

timer.Tick += new EventHandler(timer\_Tick);

timer.Interval = new TimeSpan(1000000);

}

private void GameCellClicked(MouseEventArgs e, int x, int y)

{

if (isRunning)

return;

if (e.LeftButton == MouseButtonState.Pressed)

{

game.Area.SetCell(CellType.Alive, x, y);

gameWidget.Draw();

}

else if (e.RightButton == MouseButtonState.Pressed)

{

game.Area.SetCell(CellType.Dead, x, y);

gameWidget.Draw();

}

}

private void setSpeed\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

speedStatus.Content = $"Скорость: {e.NewValue}";

if (timer != null) {

timer.Interval = new TimeSpan(100000 \* (int)e.NewValue);

}

}

private void updateSize\_Click(object sender, RoutedEventArgs e)

{

try {

var width = Int32.Parse(areaWidth.Text);

var height = Int32.Parse(areaHeight.Text);

if (width > 0)

{

game.Area.Width = width;

}

if (height > 0)

{

game.Area.Height = height;

}

gameWidget.Draw();

}

catch(Exception err){

Trace.WriteLine(err);

return;

}

}

private void btnStart\_Click(object sender, RoutedEventArgs e)

{

startStop();

}

private void Window\_SizeChanged(object sender, SizeChangedEventArgs e)

{

gameWidget.Draw();

}

private void timer\_Tick(object sender, EventArgs e)

{

game.Update();

gameWidget.Draw();

if (game.IsStable)

{

btnStart.Content = "Старт";

timer.Stop();

isRunning = false;

}

}

private void btnClear\_Click(object sender, RoutedEventArgs e)

{

if (!isRunning)

{

game.Area.Clear();

gameWidget.Draw();

}

}

private void Window\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.Key)

{

case Key.Space:

startStop();

break;

case Key.Delete:

game.Area.Clear();

gameWidget.Draw();

break;

}

}

private void startStop()

{

if (!isRunning)

{

btnStart.Content = "Стоп";

timer.Start();

isRunning = true;

}

else

{

btnStart.Content = "Старт";

timer.Stop();

isRunning = false;

}

}

private void btnFillRandom\_Click(object sender, RoutedEventArgs e)

{

if (!isRunning)

{

game.Area.FillRandom();

gameWidget.Draw();

}

}

private void btnSave\_Click(object sender, RoutedEventArgs e)

{

if (isRunning)

return;

SaveFileDialog dialog = new SaveFileDialog();

dialog.Filter = "Text Files (\*.txt)|\*.txt";

if (dialog.ShowDialog() == true)

{

using (StreamWriter writer = new StreamWriter(dialog.FileName))

{

for (int y = 0; y < game.Area.Height; y++)

{

for (int x = 0; x < game.Area.Width; x++)

{

writer.Write(game.Area.IsAlive(x, y) ? 1 : 0);

}

writer.Write('\n');

}

}

}

}

private void btnLoad\_Click(object sender, RoutedEventArgs e)

{

if (isRunning)

{

return;

}

OpenFileDialog dialog = new OpenFileDialog();

dialog.Filter = "Text Files (\*.txt)|\*.txt";

if (dialog.ShowDialog() == true)

{

var lines = File.ReadAllLines(dialog.FileName);

HashSet<int> nums = new HashSet<int>(lines.Select(x => x.Length));

if (lines.Length == 0 || nums.Count != 1)

{

MessageBox.Show("Неверный формат файла");

return;

}

game.Area.SetHeight(lines.Length);

game.Area.SetWidth(lines[0].Length);

for(int y = 0; y < lines.Length; y++)

{

for (int x = 0; x < lines[0].Length; x++)

{

game.Area.SetCell(lines[y][x] - '0' == 0? CellType.Dead : CellType.Alive, x, y);

}

}

gameWidget.Draw();

}

}}}

Приложение 3. App.config

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />

</startup>

</configuration>

Приложение 4. App.xaml

<Application x:Class="game\_life.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:game\_life"

StartupUri="MainWindow.xaml">

<Application.Resources>

</Application.Resources>

</Application>

Приложение 5. GameLife.cs

using System;

using System.Collections.Generic;

using System.Linq;

namespace game\_life

{

public enum CellType

{

Dead,

Alive,

ToAlive,

ToDead,

}

public class GameLifeArea

{

private List<List<CellType>> area;

private List<List<int>> neighbors;

private int aliveCount = 0;

public int AliveCount { get => aliveCount; }

public List<CellType> this[int key]

{

get => area[key];

}

public int Width

{

get => area[0].Count;

set { SetWidth(value); }

}

public int Height

{

get => area.Count;

set { SetHeight(value); }

}

public GameLifeArea(int width, int height)

{

initArea(width, height);

}

public CellType CellAt(int x, int y)

{

(x, y) = checkCoords(x, y);

return area[y][x];

}

public void SetCell(CellType cell, int x, int y)

{

var prevCell = area[y][x];

area[y][x] = cell;

if (cell == CellType.Alive)

{

aliveCount += 1;

if (prevCell != cell)

updateNeighbors(x, y);

}

else if (cell == CellType.Dead)

{

aliveCount -= 1;

if (prevCell != cell)

updateNeighbors(x, y);

}

}

public bool IsAlive(int x, int y)

{

return CellAt(x, y) == CellType.Alive || CellAt(x, y) == CellType.ToDead;

}

public int NeighborsCount(int x, int y)

{

return neighbors[y][x];

}

public void SetWidth(int width)

{

if (width < 0)

{

throw new ArgumentOutOfRangeException("width");

}

initArea(width, Height);

}

public void SetHeight(int height)

{

if (height < 0)

{

throw new ArgumentOutOfRangeException("height");

}

initArea(Width, height);

}

public void UpdateArea()

{

for (int x = 0; x < Width; x++)

{

for (int y = 0; y < Height; y++)

{

switch (area[y][x])

{

case CellType.ToAlive:

{

SetCell(CellType.Alive, x, y);

break;

}

case CellType.ToDead:

{

SetCell(CellType.Dead, x, y);

break;

}

}

}

}

}

public void Clear()

{

for (int y = 0; y < Height; y++)

{

for (int x = 0; x < Width; x++)

{

area[y][x] = CellType.Dead;

neighbors[y][x] = 0;

}

}

}

public void FillRandom()

{

var rand = new Random();

for (int y = 0; y < Height; y++)

{

for (int x = 0; x < Width; x++)

{

var num = rand.Next(0, 2);

SetCell(num == 0 ? CellType.Dead : CellType.Alive, x, y);

}

}

}

private void updateNeighbors(int x, int y)

{

var delta = IsAlive(x, y) ? 1 : -1;

updateNeighbor(x - 1, y - 1, delta);

updateNeighbor(x, y - 1, delta);

updateNeighbor(x + 1, y - 1, delta);

updateNeighbor(x - 1, y, delta);

updateNeighbor(x + 1, y, delta);

updateNeighbor(x - 1, y + 1, delta);

updateNeighbor(x, y + 1, delta);

updateNeighbor(x + 1, y + 1, delta);

}

private void updateNeighbor(int x, int y, int delta)

{

(x, y) = checkCoords(x, y);

neighbors[y][x] += delta;

}

private (int, int) checkCoords(int x, int y)

{

if (x < 0)

{

x = Width + x;

}

else if (x >= Width)

{

x -= Width;

}

if (y < 0)

{

y = Height + y;

}

else if (y >= Height)

{

y -= Height;

}

return (x, y);

}

private void initArea(int width, int height)

{

area = new List<List<CellType>>(height);

neighbors = new List<List<int>>(height);

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

{

area.Add(new List<CellType>(Enumerable.Range(0, width).Select(x => CellType.Dead)));

neighbors.Add(new List<int>(Enumerable.Range(0, width).Select(x => 0)));

}

}

}

class GameLife

{

private GameLifeArea area;

public GameLifeArea Area { get => area; }

private bool isStable = true;

public bool IsStable { get => isStable; }

public GameLife(int areaWidth, int areaHeight)

{

area = new GameLifeArea(areaWidth, areaHeight);

}

public void Update()

{

int updated = 0;

for (int x = 0; x < area.Width; x ++)

{

for (int y = 0; y < area.Height; y ++)

{

var neighbors = area.NeighborsCount(x, y);

if (area.IsAlive(x, y)) {

if (neighbors < 2 || neighbors > 3)

{

updated++;

area.SetCell(CellType.ToDead, x, y);

}

}

if (!area.IsAlive(x, y) && neighbors == 3)

{

updated++;

area.SetCell(CellType.ToAlive, x, y);

}

}

}

area.UpdateArea();

isStable = area.AliveCount == 0 || updated == 0;

}

}

}

Приложение 6. GameLifeVisual.cs

using System;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

namespace game\_life

{

public class GameLifeUi : FrameworkElement

{

private readonly VisualCollection \_children;

private DrawingVisual canvas;

private GameLifeArea area;

private Brush aliveBrush;

private Brush deadBrush;

private Pen penBorder = new Pen(Brushes.Gray, 2);

private int cellSize = -1;

private int currX = 0;

private int currY = 0;

private int lastMouseX = -1;

private int lastMouseY = -1;

public Action<MouseEventArgs, int, int> CellUpdate;

public GameLifeUi(GameLifeArea area, Brush aliveBrush, Brush deadBrush)

{

canvas = new DrawingVisual();

\_children = new VisualCollection(this){canvas};

this.area = area;

this.aliveBrush = aliveBrush;

this.deadBrush = deadBrush;

MouseMove += GameLifeUi\_MouseMove;

MouseDown += GameLifeUi\_MouseDown;

PreviewMouseWheel += GameLifeUi\_PreviewMouseWheel;

}

public void Draw()

{

var rect = new Rect();

if (cellSize < 0)

cellSize = Math.Max((int)Math.Min(ActualWidth, ActualHeight) / Math.Max(area.Width, area.Height), 1);

rect.Width = cellSize;

rect.Height = cellSize;

var areaStartX = Math.Min(Math.Max((currX > 0 ? currX : 0) / cellSize - 1, 0), area.Width);

var areaStartY = Math.Min(Math.Max((currY > 0 ? currY: 0) / cellSize - 1, 0), area.Height);

var marginX = currX - areaStartX \* cellSize;

var marginY = currY - areaStartY \* cellSize;

var areaEndX = Math.Min(areaStartX + (int)ActualWidth / cellSize + 1, area.Width);

var areaEndY = Math.Min(areaStartY + (int)ActualHeight / cellSize + 1, area.Height);

var ctx = canvas.RenderOpen();

var deadRect = new Rect(-marginX, -marginY, (areaEndX - areaStartX) \* cellSize, (areaEndY - areaStartY) \* cellSize);

if (deadRect.X + deadRect.Width > ActualWidth)

{

deadRect.Width = Math.Max(0, ActualWidth - deadRect.X);

}

ctx.DrawRectangle(deadBrush, null, deadRect);

for (int x = areaStartX; x < areaEndX; x++)

{

rect.X = (x - areaStartX) \* cellSize - marginX;

if (rect.X > ActualWidth)

{

continue;

}

for (int y = areaStartY; y < areaEndY; y++)

{

if (area[y][x] == CellType.Alive)

{

rect.Y = (y - areaStartY) \* cellSize - marginY;

if (rect.Y > ActualHeight)

{

break;

}

ctx.DrawRectangle(aliveBrush , null, rect);

}

}

}

ctx.DrawRectangle(null, penBorder, new Rect(0, 0, ActualWidth, ActualHeight));

ctx.Close();

}

private void GameLifeUi\_MouseDown(object sender, MouseButtonEventArgs e)

{

var p = e.GetPosition(this);

switch (e.ChangedButton)

{

case MouseButton.Left:

case MouseButton.Right:

CellUpdate(e, (currX + (int)p.X) / cellSize, (currY + (int)p.Y) / cellSize);

break;

case MouseButton.Middle:

lastMouseX = (int)p.X;

lastMouseY = (int)p.Y;

break;

}

}

private void GameLifeUi\_MouseMove(object sender, MouseEventArgs e)

{

if (e.MiddleButton == MouseButtonState.Pressed)

{

var p = e.GetPosition(this);

currX = currX + (int)(lastMouseX - p.X);

currY = currY + (int)(lastMouseY - p.Y);

lastMouseX =(int)p.X;

lastMouseY =(int)p.Y;

Draw();

} else if (e.LeftButton == MouseButtonState.Pressed || e.RightButton == MouseButtonState.Pressed)

{

var p = e.GetPosition(this);

CellUpdate(e, (currX + (int)p.X) / cellSize, (currY + (int)p.Y) / cellSize);

}

}

private void GameLifeUi\_PreviewMouseWheel(object sender, MouseWheelEventArgs e)

{

cellSize += e.Delta / 120;

if (cellSize < 1)

{

cellSize = 1;

} else if (cellSize > (int)Math.Min(ActualWidth, ActualHeight)) {

cellSize = (int)Math.Max(ActualWidth, ActualHeight);

}

Draw();

}

protected override int VisualChildrenCount => \_children.Count;

protected override Visual GetVisualChild(int index)

{

if (index < 0 || index >= \_children.Count)

{

throw new ArgumentOutOfRangeException();

}

return \_children[index];

}}}