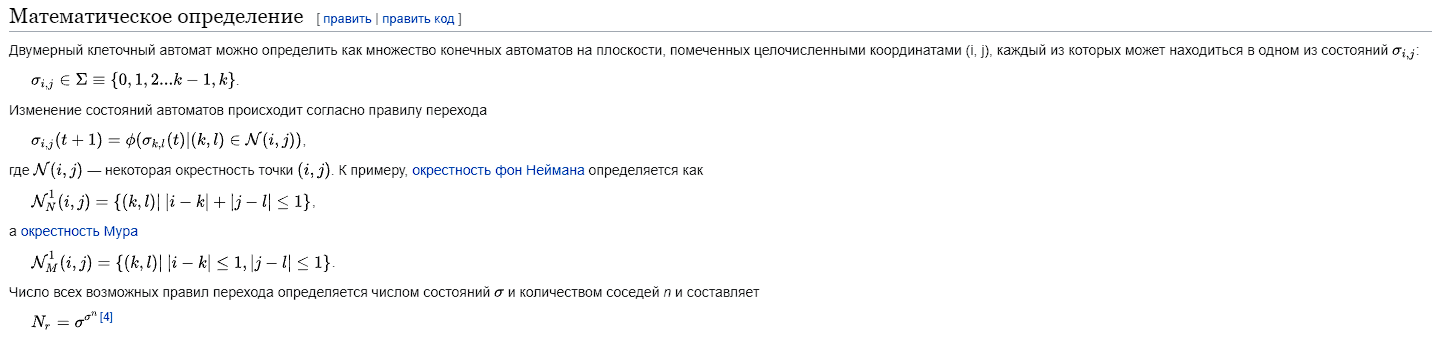
Клеточный автомат.

Клеточный автомат — дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Основой является пространство из прилегающих друг к другу клеток (ячеек), образующих решётку. Каждая клетка может находиться в одном из конечного множества состояний (например, 1 и 0). Решётка может быть любой размерности, бесконечной или конечной.



Игра «Жизнь» (англ. Game of Life) — клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году. Это игра без игроков, в которой человек создаёт начальное состояние, а потом лишь наблюдает за её развитием. В игре можно создать процессы с полнотой по Тьюрингу, что позволяет реализовать любую машину Тьюринга.

Влияние на развитие наук

Хотя игра состоит всего из двух простых правил, тем не менее она более сорока лет привлекает внимание учёных. Игра «Жизнь» и её модификации повлияли (в ряде случаев взаимно) на многие разделы таких точных наук, как математика, информатика и физика. Это, в частности:

* Теория автоматов,
* Теория алгоритмов,
* Теория игр и математическое программирование,
* Алгебра и теория чисел,
* Теория вероятностей и математическая статистика,
* Комбинаторика и теория графов,
* Фрактальная геометрия,
* Вычислительная математика,
* Теория принятия решений,
* Математическое моделирование.

Варианты, модификации и обобщения

Существуют модификации игры «Жизнь» / «Эволюция» по:

* размерности — на плоскости, в объёме;
* цветности — однотоновая, чёрно-белая (шахматная), полноцветная;
* направлению алгоритма — прямой, обратный;
* константам эволюции — классические (B3/S23), изменённые;
* размерам игрового поля — ограниченное, неограниченное, полуограниченное;
* активности поля — активное, пассивное;
* количеству игроков — zero-game, один, два;
* активности игры — пассивная, активная;
* геометрии поля — прямоугольная, шестиугольная.

К настоящему времени более-менее сложилась следующая классификация фигур:

* Устойчивые фигуры: фигуры, которые остаются неизменными:
* Блоки
* Ульи и пасеки
* Пруды
* Караваи
* Лодки
* Корабли
* Устойчивый пожиратель «рыболовный крючок»
* Пекарня (симметрия двойных караваев)
* Длинные баржи, лодки и корабли
* Лодочный и корабельный банты
* Каноэ
* Авианосец
* Интеграл
* Манго
* Змея
* Долгожители: фигуры, которые долго меняются, прежде чем стабилизироваться;
* Периодические фигуры: фигуры, у которых состояние повторяется через некоторое число поколений, большее 1:
* Периодические мигалки
* Жаба
* Часы
* Маяк
* Bipole, Tripole, Quadropole,Pentapole
* Феникс
* Лист
* Стрекоза
* Стрела
* Волк
* Бормочущий ров
* Очки
* Пульсар
* Звезда
* Двигающиеся фигуры: фигуры, у которых состояние повторяется, но с некоторым смещением:
* Классические планеры или глайдеры
* Корабли (лёгкие, средние и тяжёлые)
* Паровозы
* Ружья: фигуры с повторяющимися состояниями, дополнительно создающие движущиеся фигуры:
* Ружьё Госпера
* 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;
* Фигуры, которые при столкновении с некоторыми фигурами дублируются;

Райским садом (садом Эдема) называется такое расположение клеток, у которого не может быть предыдущего поколения. Практически для любой игры, состояние клеток в которой определяется несколькими соседями на предыдущем шаге, можно доказать существование садов Эдема, но построить конкретную фигуру гораздо сложнее.

Структура проекта.

**Глобальные переменные и константы**

1. **canvas**: HTML элемент <canvas> для рисования.
2. **ctx**: Контекст рисования на холсте, получаемый из canvas.getContext('2d').
3. **cellSize**: Размер клетки в пикселях.
4. **fillRadius**: Радиус заполнения клеток при взаимодействии мышью.
5. **isRunning**: Флаг, указывающий, запущена ли симуляция.
6. **grid**: Двумерный массив, представляющий текущее состояние клеток на игровом поле.
7. **rows**: Количество строк на игровом поле.
8. **cols**: Количество столбцов на игровом поле.
9. **isMouseDown**: Флаг, указывающий, нажата ли кнопка мыши.
10. **mouseButton**: Переменная для хранения значения нажатой кнопки мыши (левая или правая).
11. **radius**: Радиус соседства для правил жизни клеток.
12. **cellStates**: Количество состояний клетки (живая или мёртвая).
13. **maxNeighbors**: Максимальное количество соседей.
14. **survivalRange**: Диапазон количества соседей, при котором клетка остаётся живой.
15. **birthRange**: Диапазон количества соседей, при котором клетка рождается.
16. **neighborhood**: Тип соседства (Моор или фон Нейман).
17. **initialSettings**: Объект, содержащий начальные настройки симуляции.
18. **patterns**: Объект, содержащий предустановленные шаблоны клеток (например, глайдер).

**Функции**

1. **init()**
   * **Описание**: Инициализирует игровое поле, создавая сетку клеток и рисуя её. А также создаёт двумерный массив заданных размеров, заполненный нулями.
   * **Параметры**: Нет.
2. **drawGrid()**
   * **Описание**: Рисует сетку клеток на холсте на основе текущего состояния grid.
   * **Параметры**: Нет.
3. **step()**
   * **Описание**: Выполняет один шаг симуляции, обновляя состояние grid в соответствии с правилами жизни.
   * **Параметры**: Нет.
4. **countAliveNeighbors(row, col)**
   * **Описание**: Считает количество живых соседей для клетки в заданной позиции.
   * **Параметры**:
     + row: Номер строки клетки.
     + col: Номер столбца клетки.
   * **Возвращает**: Количество живых соседей.
5. **toggleRunning()**
   * **Описание**: Переключает состояние симуляции (запущена или остановлена).
   * **Параметры**: Нет.
6. **run()**
   * **Описание**: Запускает симуляцию, выполняя шаги с задержкой, соответствующей значению ползунка скорости.
   * **Параметры**: Нет.
7. **clearGrid()**
   * **Описание**: Очищает сетку, заполняя её нулями, и перерисовывает её.
   * **Параметры**: Нет.
8. **resetSettings()**
   * **Описание**: Сбрасывает настройки симуляции к начальному состоянию и перерисовывает сетку.
   * **Параметры**: Нет.
9. **randomFill()**
   * **Описание**: Заполняет сетку случайным образом, устанавливая каждую клетку в живое или мёртвое состояние.
   * **Параметры**: Нет.
10. **getCellCoordinates(event)**
    * **Описание**: Преобразует координаты мыши в координаты клетки на сетке.
    * **Параметры**:
      + event: Событие мыши.
    * **Возвращает**: Объект с координатами клетки { row, col }.
11. **toggleCellState(row, col, state)**
    * **Описание**: Изменяет состояние клеток в радиусе заполнения вокруг заданной клетки.
    * **Параметры**:
      + row: Номер строки центральной клетки.
      + col: Номер столбца центральной клетки.
      + state: Состояние (1 - живая, 0 - мёртвая).
12. **placePattern(pattern, startRow, startCol)**
    * **Описание**: Размещает заданный шаблон клеток на сетке, начиная с указанной позиции.
    * **Параметры**:
      + pattern: Название шаблона.
      + startRow: Начальная строка для размещения шаблона.
      + startCol: Начальный столбец для размещения шаблона.
13. **parsePattern()**
    * **Описание**: Парсит строку шаблона, обновляет параметры симуляции и перерисовывает сетку.
    * **Параметры**: Нет.
14. **updatePatternControls()**
    * **Описание**: Обновляет состояние элементов управления шаблонами в зависимости от выбранного режима заполнения (круг, квадрат или шаблон).
    * **Параметры**: Нет.

### Константы, обозначающие HTML-объекты

1. **canvas**
   * HTML элемент: <canvas id="gameCanvas">
2. **ctx**
   * Контекст рисования для canvas.
3. **startStopBtn**
   * HTML элемент: <button id="startStopBtn">
4. **stepBtn**
   * HTML элемент: <button id="stepBtn">
5. **clearBtn**
   * HTML элемент: <button id="clearBtn">
6. **resetBtn**
   * HTML элемент: <button id="resetBtn">
7. **randomBtn**
   * HTML элемент: <button id="randomBtn">
8. **cellSizeInput**
   * HTML элемент: <input id="cellSize" type="number">
9. **wrapCheckbox**
   * HTML элемент: <input id="wrapCheckbox" type="checkbox">
10. **patternInput**
    * HTML элемент: <input id="pattern" type="text">
11. **parsePatternBtn**
    * HTML элемент: <button id="parsePatternBtn">
12. **resultElement**
    * HTML элемент: <p id="result">
13. **speedSlider**
    * HTML элемент: <input id="speedSlider" type="range">
14. **speedValue**
    * HTML элемент: <span id="speedValue">
15. **interactionCheckbox**
    * HTML элемент: <input id="interactionCheckbox" type="checkbox">
16. **fillRadiusSlider**
    * HTML элемент: <input id="fillRadiusSlider" type="range">
17. **fillRadiusValue**
    * HTML элемент: <span id="fillRadiusValue">
18. **fillShapeRadios**
    * HTML элементы: <input name="fillShape" type="radio"> (несколько элементов, например, для выбора круга или квадрата)

### Обработчики событий и привязанные функции

1. **canvas**
   * **mousedown**: Привязана функция для взаимодействия мышью с игровым полем.
     + Привязанная функция: Анонимная функция, которая определяет тип взаимодействия (добавление или удаление клеток) и вызывает toggleCellState.
   * **mousemove**: Привязана функция для взаимодействия мышью с игровым полем при движении.
     + Привязанная функция: Анонимная функция, которая добавляет или удаляет клетки в зависимости от нажатой кнопки мыши и вызывает toggleCellState.
   * **mouseup**: Привязана функция для завершения взаимодействия мышью.
     + Привязанная функция: Анонимная функция, которая сбрасывает флаг isMouseDown.
   * **contextmenu**: Привязана функция для предотвращения контекстного меню при нажатии правой кнопки мыши.
     + Привязанная функция: Анонимная функция, которая вызывает event.preventDefault().
2. **startStopBtn**
   * **click**: Привязана функция для запуска и остановки симуляции.
     + Привязанная функция: toggleRunning
3. **stepBtn**
   * **click**: Привязана функция для выполнения одного шага симуляции.
     + Привязанная функция: step
4. **clearBtn**
   * **click**: Привязана функция для очистки игрового поля.
     + Привязанная функция: clearGrid
5. **resetBtn**
   * **click**: Привязана функция для сброса настроек к начальным.
     + Привязанная функция: resetSettings
6. **randomBtn**
   * **click**: Привязана функция для случайного заполнения игрового поля.
     + Привязанная функция: randomFill
7. **cellSizeInput**
   * **change**: Привязана функция для обновления размера клетки и переинициализации игрового поля.
     + Привязанная функция: Анонимная функция, которая обновляет cellSize и вызывает init.
8. **parsePatternBtn**
   * **click**: Привязана функция для парсинга введённого шаблона и обновления параметров симуляции.
     + Привязанная функция: parsePattern
9. **speedSlider**
   * **input**: Привязана функция для обновления значения скорости симуляции.
     + Привязанная функция: Анонимная функция, которая обновляет текстовое значение speedValue.
10. **fillRadiusSlider**
    * **input**: Привязана функция для обновления значения радиуса заполнения.
      + Привязанная функция: Анонимная функция, которая обновляет fillRadius и текстовое значение fillRadiusValue.
11. **fillShapeRadios**
    * **change**: Привязана функция для обновления состояния элементов управления шаблонами в зависимости от выбранного режима заполнения (круг, квадрат или шаблон).
      + Привязанная функция: updatePatternControls

По диаграмме:

Здесь представлена подробная блок-схема с пояснениями к представленному коду:

1. \*\*Инициализация\*\*: Этот подграф обрабатывает начальную настройку приложения, включая инициализацию холста, переменных, предопределенных шаблонов и настроек игры.

2. \*\*Основные функции\*\*: Этот подграф содержит основные функции реализации Game of Life, такие как инициализация сетки, рисование сетки, вычисление следующего поколения, подсчет соседей, переключение состояния выполнения, очистка сетки, сброс настроек, случайное заполнение сетки, получение координат ячеек, переключение состояний ячеек, размещение паттернов и разбор входных данных паттерна.

3. \*\*Обработчики событий\*\*: Этот подграф управляет различными слушателями событий, прикрепленными к холсту, кнопкам и элементам ввода. Он обрабатывает события опускания, перемещения и подъема мыши на холсте, а также события нажатия на кнопки старт/стоп, шаг, очистка, сброс и случайный выбор, и события ввода на кнопки размера ячейки, разбора узора, ползунка скорости, ползунка радиуса заливки и радиокнопки формы заливки.

На блок-схеме подробно описана структура и функциональность кода, а каждый подграф представляет логическую единицу или функцию в приложении. Пояснения, прикрепленные к различным элементам диаграммы, помогают прояснить назначение и поведение каждого компонента, облегчая пользователю понимание общей логики и потока реализации Game of Life.