UNICHECK

Ім'я користувача:
Бариніна Людмила Іванівна

ID перевірки:
1008185193

Дата перевірки:
04.06.2021 21:01:01 +03

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
04.06.2021 21:01:21 +03

ID користувача:
91039

Назва документа: **МОЛЧАНОВА_Варвара_e45854_main_part**

Кількість сторінок: **62** Кількість слів: **10302** Кількість символів: **74641** Розмір файлу: **5.58 MB** ID файлу: **1008262467**

3.58%
Схожість

Найбільша схожість: **1.05%** з джерелом з Бібліотеки (ID файлу: **1008240347**)

0.86% Джерела з Інтернету	13	Сторінка 64
3.07% Джерела з Бібліотеки	135	Сторінка 64

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2

ВСТУП

Одним з основних застосувань клітинних автоматів є комп'ютерне моделювання різних динамічних процесів. Навіть якщо КА не були винайдені для цієї мети, вони мають особливості, які роблять їх надзвичайно придатними для цього завдання. Основна перевага використання КА для динамічного моделювання обумовлена їх абсолютно дискретним характером, що дозволяє точно моделювати процеси на комп'ютері. Також не останньою причиною широкої застосовності моделювання КА є той факт, що алгоритми, а не диференційні рівняння, є найкращими інструментами схематизації динамічних процесів для складних та організованих систем [22]. Оскільки прості алгоритми можуть бути природно реалізовані на КА, останні дуже корисні для реалізації простих моделей та моделювання в багатьох галузях: біології, економіці, екології, нейронних мережах, моделях трафіку, інформатиці та, особливо, в криптографії. Ще однією великою перевагою КА в комп'ютерному моделюванні є те, що за своєю суттю вони добре придатні для реалізації на паралельних машинах.

Але розробка програм взагалі та паралельних програм зокрема вимагає чималих навичок. Якщо ж ви хочете ознайомитися з роботою клітинних автоматів чи змоделювати за їх допомогою якесь явище, але не хочете чи не маєте можливості розробити для цього власну програму, вам потрібна система з можливістю створювати різноманітні КА зі зрозумілим інтерфейсом, що не вимагатиме від вас додаткових знань, окрім безпосередньо знань в області КА. Ця умова й стала головною вимогою до розроблюваної в цьому проєкті системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Історична довідка

У звичайних моделях обчислень, таких як машина Тюрінга, розрізняють зафіксовану структурну частину комп'ютера та змінні дані, якими комп'ютер оперує. Комп'ютер не може оперувати своєю власною «матеріальною частиною», він не може розширювати, модифікувати себе, чи створювати інші комп'ютери. Над знищенням цієї різниці між даними та структурою працював наприкінці сорокових років Джон фон Нейман, ім'я якого зв'язують з архітектурою сучасних комп'ютерів. Він думав про створення машини, яка була б здатна імітувати мозок людини і завдяки цьому розв'язувати дуже складні задачі, однак підвищення ефективності роботи комп'ютерів не було його головною метою. Він вважав, що машина з такою ж складністю, як мозок, повинна мати механізми самоконтролю та самовідтворення. Це привело його до ідеї автомату, здатному будувати самого себе з доступного матеріалу. Але враховуючи, що побудувати він хотів машину для рішення прикладних задач, самоконструювання повинно бути лише можливим типами її активності, головною ж властивістю мала стати здатність до універсальних обчислень – тобто виконання будь-якого комп'ютерного алгоритму.

Слідуючи цим правилам та ідеям Станіслава Улама [13], фон Нейман розглянув це питання в рамках повністю дискретного простору, що складається з клітин. Кожна клітина характеризується внутрішнім станом, який зазвичай складається з кінцевої кількості інформаційних бітів. Фон Нейман припустив, що ця система клітин розвивається з дискретними часовими кроками, як прості автомати, які знають лише простий рецепт для обчислення свого нового внутрішнього стану. Правило, що визначає еволюцію цієї системи, однакове для всіх клітин і є функцією від станів

сусідніх клітин. Подібно до будь-якої біологічній системі, діяльність клітин відбувається одночасно. Ці повністю дискретні динамічні системи (клітинний простір), винайдені фон Нейманом, тепер називають клітинними автоматами. Правило універсальних обчислень виконується завдяки тому, що за допомогою правил клітинних автоматів можна змоделювати будь-який логічний вентиль, а на їх основі – побудувати обчислювальну машину [2].

Сплеск активності у дослідженнях КА відбувся у 70-х роках з представлення «гри життя» Джона Конвея [14]. Його мотивацією було знайти просте правило, що веде до складної поведінки. Гра була створена як проста екологічна модель, яка містить клітини, що можуть мати лише два стани: бути живою або мертвою, та які живуть та вмирають відповідно до кількох простих правил:

- Жива клітина, що має менше ніж два живих сусіда, вмирає через «малонаселеність»
- Жива клітина, що має два або три живих сусіда, продовжує жити у «наступному поколінні»
- Жива клітина, що має більше ніж три живих сусіда, вмирає через «перенаселеність»
- Мертва клітина, що має рівно три живих сусіда, стає живою «у наслідку розмноження»

На рисунку 1.1 показано три конфігурації автомата гри життя, розділені 10 ітераціями. Виявилося, що автомат гри життя має несподівано багату поведінку. Складні структури виходять із примітивного «супу» і еволюціонують таким чином, щоб розвинути деякі навички. Наприклад, можуть утворюватися об'єкти, що називаються глайдерами. Глайдери відповідають певному розташуванню сусідніх комірок, що має властивість пересуватися по простору вздовж прямих траєкторій. Набагато більше таких структур було виявлено у величезному обсязі літератури, присвяченої грі життя [4.21,4.22]. Що стосується правила фон Неймана, гра життя – це

клітинні автомати, здатні до обчислювальної універсальності.

Рисунок 1.1 – Автомат «Гра життя»

Наступною хвилею досліджень стало застосування КА до фізики, зокрема, демонстрація того, як основні риси фізики можна охопити в обчислювальній формі. Інтерес до цієї теми був спричинений здебільшого Томмазо Тоффолі [15] та Едвардом Фредкіним. Стівен Вольфрам відповідав за охоплення широкого інтересу фізичної спільноти низкою статей у 80-ті роки [16], тоді як інші застосовували КА до різноманітних проблем в інших областях [17]. Важливим технологічним розвитком у цей час було впровадження спеціального обладнання у вигляді клітинних автоматів [2] та масово паралельних комп'ютерів. Ці дослідження поклали початок розробці ґратчастих газів [18], які самі по собі стали окремим напрямком досліджень [19].

1.2. Визначення клітинного автомату

Існує декілька способів визначення КА. Науковці зазвичай визначають його як динамічну систему, в якій простір, час та стани системи є дискретними та мають наступні властивості:

- Простір представлений сіткою в кінцевій кількості вимірів (зазвичай — в одному чи двох)

- Кожна клітина в КА може перебувати в одному стані з їх кінцевої множини
- Система КА розвивається протягом кількох часових етапів. Стан усіх клітин оновлюється синхронно на кожному етапі
- Стан клітин оновлюється у відповідності до набору правил, які ставлять у відповідність поточному стану клітини та її сусідів новий стан цієї клітини [2].

Природно описати КА в їх початковому контексті: теорії (автоматизованих) обчислень. Теоретичні обчислювальні пристрої можна згрупувати в класи еквівалентності, відомі як моделі обчислень, а елементи цих класів називаються автомати. Мабуть, найпростіший тип автомата, який можна розглянути, – це скінченний автомат або кінцевий автомат. Кінцеві автомати мають входи, виходи, кінцевий обсяг стану (або пам'яті) та зворотний зв'язок з виходу на вхід; крім того, стан змінюється якимось добре регламентованим способом, наприклад, у відповідь на годинник (рис. 1.2).

Рисунок 1.2 – Кінцевий автомат [3]

Вихід залежить від поточного стану, а наступний стан залежить від входів, а також поточного стану. Тоді клітинний автомат – це звичайний масив однакових скінченних автоматів, вхід яких береться з виходів сусідніх автоматів. КА мають перевагу над іншими моделями обчислень тим, що вони паралельні, як і фізичний світ. У цьому сенсі вони кращі за послідовні

моделі для опису процесів, які мають незалежну, але одночасну діяльність, що відбувається у фізичному просторі [3].

Оскільки модель КА завжди виражається як алгоритм і незмінно реалізується як комп'ютерна програма і запускається на комп'ютері, корисно дати обчислювальне визначення КА. Тоді КА – це комп'ютерна програма, в якій виконуються наступні впорядковані обчислення:

- Створюється матриця з певними значеннями елементів.
- Визначається функція або набір функцій, які можна використовувати для зміни значення елемента матриці на основі значень елемента та сусідніх елементів.
- Функція застосовується (неодноразово) до матриці, щоразу змінюючи значення всіх елементів матриці одночасно [2].

В усіх визначеннях ключовими є поняття правил, за якими змінюються значення клітин, та сусідства – набору клітин, теперішній стан яких впливає на майбутній стан обраної клітини. Ці поняття дуже тісно зв'язані та можуть визначатися одне через інше. При побудові КА можна або спочатку визначити сусідство клітини відповідно до явища, яке моделюється, та складати правила, що використовують це сусідство; або спочатку скласти набір правил та прийняти за сусідство клітини всі клітини, що в цих правилах використовуються.

1.2.1. Види сусідств клітини

Набір клітин, який використовується в якості сусідства, залежить від властивостей системи, що моделюється, але для двовимірних КА є два найпоширеніших універсальних набори [1]:

1. Набір з п'яти клітин, що складаються з клітини X та чотирьох найближчих місць, що лежать на північ (вгорі), схід (праворуч), південь (знизу) та захід (ліворуч) від X, відомі як сусідство фон Неймана (рис. 1.3) для X.

Рисунок 1.3 – Сусідство фон Неймана

2. Набір з дев'яти клітин, що складаються з клітини X, чотирьох найближчих сусідів у його сусідстві фон Неймана та чотирьох найближчих клітин, що лежать на північний схід, південний схід, південний захід та північний захід від X, відомий як сусідство Мура (рис. 1.4) для X.

Рисунок 1.4 – Сусідство Мура

Сусідство Марголуса, яке часто використовується при симуляції фізичних явищ, буде розглянуто у розділі 1.3 разом з блоковими КА, до яких воно застосовується.

1.2.2. Правила переходу між станами КА

Правила КА є функцією скінченної множини, тому канонічною формою їх запису є довідкова таблиця, тобто запис відповідності кожному набору вхідних значень (теперішніх станів клітини та її сусідів) вихідного (наступному стану клітини). Якщо взяти за приклад правило «Parity», запропоноване Едвардом Фредкіним з МТИ на початку ери клітинних автоматів [12], яке базується на сусідстві фон Неймана, то його табличний запис буде виглядати так, як показано у таблиці 1.1. Якщо почати симуляцію

КА, побудованого на цьому правилі, з квадрату 32x32, після 50 та 100 кроків будуть отримані результати, зображені на рисунку 1.5.

Рисунок 1.5 – Правило «Parity» після 50 та 100 кроків [2]

Таблиця 1.1 – Табличний запис правила «Parity»

EWSNX	Next X	EWSNX	Next X	EWSNX	Next X	EWSNX	Next X
0	0	1000	1	10000	1	11000	0
1	1	1001	0	10001	0	11001	1
10	1	1010	0	10010	0	11010	1
11	0	1011	1	10011	1	11011	0
100	1	1100	0	10100	0	11100	1
101	0	1101	1	10101	1	11101	0
110	0	1110	1	10110	1	11110	0
111	1	1111	0	10111	0	11111	1

У випадку класичних одновимірних клітинних автоматів (сусідством вважаються три клітини над поточною, клітини може мати два стани), запис можна зробити ще зручнішим: поставити у відповідність кожному з варіантів станів сусідства певний розряд двійкового числа. Тоді вихідні значення клітин будуть кодувати значення цих розрядів, а правило можна записати у вигляді єдиного числа. На рисунках 1.6 та 1.7 можна побачити

одні з найвідоміших «правило 30» ($00011110_2 = 30_{10}$) та «правило 126» ($01111110_2 = 126_{10}$) для одновимірного клітинного автомата та їх запис.

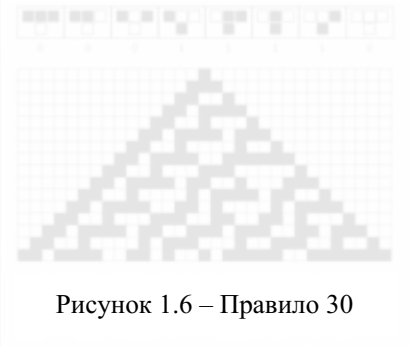

Рисунок 1.6 – Правило 30

Рисунок 1.7 – Правило 126

Такий запис правил у вигляді єдиного числа називають «кодом Вольфрама». Також саме Вольфрам розробив найвідомішу класифікацію КА, яку буде розглянуто в наступному параграфі.

1.3. Класифікація клітинних автоматів

1.3.1 Класифікація Вольфрама

Стівен Вольфрам у своїй книзі «A New Kind of Science» та декількох роботах 1980-х років визначив чотири класи, на які можна розділити клітинні автомати та деякі інші прості обчислювальні моделі, виходячи з їх поведінки. У той час як більш ранні дослідження намагалися визначити тип закономірностей для конкретних правил, класифікація Вольфрама стала

першою змогою класифікувати самі правила. У порядку зростання складності, він відокремив наступні класи [5]:

- Клас 1: Поведінка дуже проста, майже усі початкові конфігурації ведуть до однакового універсального кінцевого стану.

Рисунок 1.8 – Правило 250

- Клас 2: Існує декілька різних можливих кінцевих станів, але усі вони складаються з визначеного набору простих структур, які або залишаються незмінними назавжди, або повторюються кожні декілька кроків.

Рисунок 1.9 – Правило 108

- Клас 3: Поведінка більш складна, і багато в чому здається випадковою, але трикутники та інші дрібномасштабні структури майже завжди з'являються на якомусь з рівнів. До даного класу належать розглянуті вище правила 30 та 126.

- Клас 4: Включає суміш порядку та випадковості: створюються локалізовані структури, які самі по собі досить прості, але ці структури рухаються і взаємодіють одна з одною дуже складними способами. Найкращим прикладом цього класу є «Гра життя» Конвея.

Далі розглянемо декілька класів КА, що відрізняються один від одного структурою, а не складністю.

1.3.2. Блокові клітинні автомати

Це окремий клас клітинних автоматів, правила конструювання якого суттєво відрізняються від розглянутих вище стандартних, але який є дуже корисним для моделювання фізичних властивостей. Блоковим є автомат, у якому [2]:

1. Масив клітин розбитий на множину кінцевих, окремих однорідних частин – блоків.
2. Блоки не перетинаються та обмін інформацією між блоками відсутній.
3. Є правила для блоків, які розглядають та оновлюють вміст усього блоку (а не окремої клітини як у звичайному КА). Одне й те саме правило застосовується до всіх блоків.
4. Розбиття змінюється від кроку до кроку таким чином, щоб відбувалося перекриття блоків, що використовуються на сусідніх кроках. Це правило є дуже суттєвим, адже при використанні одного й того самого розбиття на всіх кроках КА був би розбитий на сукупність незалежних підсистем, та не зміг функціонувати як одне ціле.

Звичайно, для такої специфічної структури звичайні сусідства та правила незастосовні. Найпростішим та найчастіше використовуваним у блокових автоматах є сусідство Марголуса, в якому чітка розбита на двох-

клітинні блоки (або квадрати 2x2 у двовимірних автоматах, або куби 2x2x2 у тривимірних і т.д.), які здвигаються на одну клітину вздовж кожного виміру після кожного кроку часу [6]. Для двовимірного КА це виглядає наступним чином (правила перетворення застосовуються спочатку на блоки, обмежені синіми лініями, на наступному кроці – на блоки, обмежені червоними лініями, і так далі):

Рисунок 1.10 – Сусідство Марголуса для блокового КА

Таблиця правил переходу у наступний стан для блокового автомата містить вихідний стан для кожного з можливих варіантів вхідного складається усього з 16 елементів, якщо правила для парної та непарної фази збігаються (рис. 1.11). В іншому випадку вона буде складатися з 32 елементів.

Рисунок 1.11 – Приклад запису правила для блокового КА

Корисність блокових автоматів для моделювання фізичних явищ зумовлена тим, що для нього легко створити правила таким чином, щоб він підкорявся важливій властивості реверсивності, характерної наприклад для ґратчастих газів.

1.3.3. Реверсивні (оборотні) клітинні автомати

Це клітинні автомати, в яких у кожної конфігурації є єдиний можливий попередник, тому попередній стан кожної клітини перед оновленням можна однозначно визначити за оновленими станами сусідніх клітин. Реверсивний клітинний автомат з такою ж динамікою, яка йде у зворотному напрямку часу, завжди можливо описати за допомогою іншого правила (однак, можливо, з використанням значно більшого сусідства).

Існує декілька методів задання реверсивних КА, найвідоміші з них це:

- Блокові КА, які розбивають клітини на блоки та застосовують оборотні функції окремо до кожного блоку.
- КА другого порядку, у якому правило оновлення клітин залежить не від одного, а від двох попередніх кроків автомату.

Якщо КА не заданий за допомогою одного з цих способів, питання, чи є даний автомат оборотним можливо вирішити лише для одновимірного автомату. Також, було доведено, що будь-який реверсивний клітинний автомат можна симулювати за допомогою блокового клітинного автомату.

Оборотні клітинні автомати часто використовують для моделювання таких фізичних явищ, як динаміка газів і рідин, оскільки вони підкоряються законам термодинаміки.

1.3.4. Тоталістичні клітинні автомати

КА, в яких стан клітин виражається числом, та стан клітини у момент часу залежить від суми (або середнього арифметичного) значень клітин сусідства на попередньому кроці. Подібно до елементарних (тобто одновимірних) клітинних автоматів, еволюція одновимірного тоталістичного

клітинного автомату може бути повністю описана правилом, що визначає стан, який буде мати дана клітина в наступному поколінні на основі суми значень трьох клітин, що складаються з клітини трохи вище він у сітці, той, що ліворуч, і той праворуч. «Гра життя» Конвея є найвідомішим прикладом тоталістичних клітинних автоматів.

1.4. Огляд існуючих систем моделювання КА

1.4.1 Wolfram Mathematica

Wolfram Mathematica (зазвичай називається Mathematica) – це програмна система із вбудованими бібліотеками для декількох областей технічних обчислень, що дозволяє виконувати символічні обчислення, маніпулювання матрицями, побудову графіків функцій та різних типів даних, реалізацію алгоритмів, створення користувацьких інтерфейсів та взаємодію з програмами, написані іншими мовами програмування. Усе це відбувається за допомогою багатопарадигмової мови Wolfram Language, розробленою, як і сама Mathematica, дослідницькою групою Wolfram Research на чолі зі Стівеном Вольфрамом.

Моделювання клітинних автоматів є лише однією з задач, що можна вирішити за допомогою Wolfram Mathematica. Елементарні одновимірні КА можна побудувати за допомогою вбудованої функції *CellularAutomaton[rule, init, steps]* (рис. 1.12).

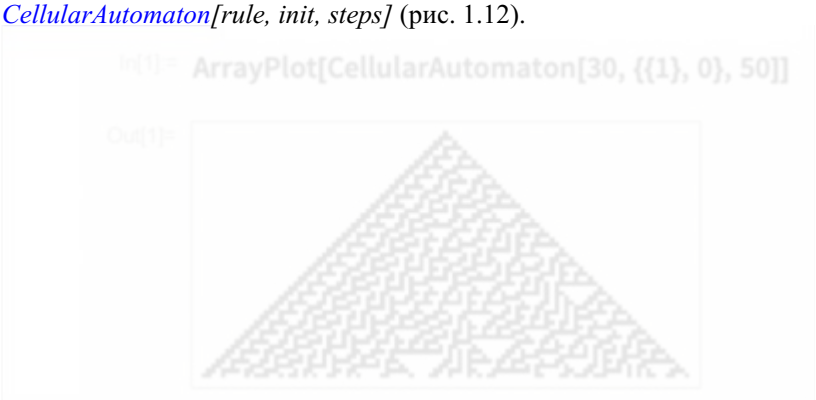

Рисунок 1.12 – Моделювання одновимірного КА у Wolfram Mathematica

Але для моделювання більш складної систем потрібно вивчити Wolfram Language на достатньо глибокому рівні. нижче показаний фрагмент програми, яка створює модель «Гри життя» Конвея:

```
LiveConfigs = Join[
   Map [Join [ {O} #]&, Permutations[{1, 1, 1, 0, 0, 0, 0, 0} 1],
   Map [Join [ {1} #]&, Permutations[{1, 1, 1, 0, 0, 0, 0, 0} 1],
   Map [Join [ {1} #]&, Permutations[{1, 1, 0, 0, 0, 0, 0, 0}]]1;
DieConfigs = Complement[Flatten[
      Map[Permutations, Map[Join[Tab1e[1, {#}1, Tab1e[O, {(9- #)}1]&,
Range[O, 9]]], 1] Liveconfigs];
   Map[Length, {Liveconfigs, DieConfigs}] {140, 372}
   Apply[(update[##] 1)&, LiveConfigs, 1];
   Apply[(update[##] 0)&, DieConfigs, 1];
```

1.4.2 CAM

CAM-6 є машиною клітинних автоматів, призначеної для того, щоб служити лабораторією експериментатора, засобом повідомлення результатів і середовищем для інтерактивної демонстрації в режимі реального часу. Машина була спочатку розроблена в Лабораторії інформатики Массачусетського технологічного інституту (MTI). В даний час вона проводиться фірмою SYSTEMS CONCEPTS (Сан-Франциско, CA), яка її поширює з ясно сформульованої мети: після задоволення внутрішніх потреб MTI зробити подальшу продукцію виробничого конвеєра доступною широким науковим колам по настільки низькою, наскільки це можливо, ціною.

Фізично CAM-6 складається з модуля, який вставляється в один роз'єм IBM-PC (XT, AT або сумісних з ними моделей), і керуючого програмного забезпечення, що працює в середовищі PC-DOS2. У той час як цей легко доступний головний комп'ютер забезпечує розміщення, екранування, електроживлення, дискову пам'ять, монітор і стандартну операційну середу, вся справжня робота з моделювання клітинних автоматів з дуже високою

швидкістю відбувається самим модулем з швидкодією, яке можна порівняти (для цього приватного додатки) з швидкодією CRAY-1. Керуючий програмне забезпечення для CAM-6 написано на FORTH і працює на IBM-PC з пам'яттю 256 К. Це програмне забезпечення доповнено низкою готових додатків і демонстраційних прикладів і включає повний анотований список джерел. Сама система FORTH, що отримана з моделі F83 Лакса і Перрі, є загальнодоступною і супроводжується повною вхідною інформацією, забезпеченою примітками [2].

CAM-6 дає можливість моделювати автомати з багатьма станами клітин за допомогою площин бітів. Якщо уявити, що кожна клітина розділена на два біти, що знаходяться один над іншим, то вона зможе мати 4 стани: 00, 01, 10, 11. Тоді увесь масив клітин можна уявити як декілька масивів (площин) бітів, що знаходяться один над іншим. Користувач CAM-6 має доступ до чотирьох таких площин, отже клітини може мати до 2^4=16 станів.

1.4.3 CelLab

Розробка CelLab була почата Руді Рукером та Джоном Уокером у 1988 та 1989 роках, коли вони працювали в дослідницькій лабораторії Autodesk. CelLab дозволяє досліджувати клітинні автомати на власному персональному комп'ютері за допомогою симулятора WebCA, що працює у веб-браузері, використовуючи засоби JavaScript та HTML5, щоб забезпечити високоефективну емуляцію клітинних автоматів. Система пропонує широкий спектр готових до використання правил, що імітують такі різноманітні процеси, як тепловий потік, дифузія газів, відпал металу, поведінка трубчастих червів на дні океану, хімічні реакції та екосистеми штучного життя [10]. Вигляд симуляції потоків газу у цій системі показаний на рисунку 1.13. А якщо серед запропонованих готових моделей не знайшлося потрібної, користувач може визначити власні правила, написавши короткі програми на JavaScript або Java. Наприклад, опис «Гри

життя» для симуляції в даній системі на мові Java буде виглядати так, як зображено на рисунку 1.14.

Рисунок 1.13 – Симуляція потоків газу в системі CelLab

Рисунок 1.14 – Запис правил «Гри життя» в системі CelLab

У таблиці 1.2 наведено порівняння розглянутих вище систем моделювання КА та нашої системи.

Таблиця 1.2 – Порівняння систем моделювання КА

Властивість \ Назва системи	Wolfram Mathematica	CAM-6	CelLab	Наша система
Графічний інтерфейс	-	-	+	+
Безплатна	-	-	+	+
Універсальна мова опису правил	+	+	+	-
Графічне завдання правил	-	-	-	+
Доступність онлайн	-	-	+	+

Можна побачити, що при плануванні розроблюваної системи ми концентрувалися на зручності її використання будь-яким користувачем, адже додатками з графічним інтерфейсом користуватися простіше, ніж побудованими на складних мовах обчислення або програмування. Також один з пунктів порівняння, а саме «графічне завдання правил» може викликати питання, адже в розділі 1.2.2 було сказано, що класичний та найзручніший для сприйняття машиною спосіб запису правил – за допомогою таблиці. Але те, що зручно для обчислювальних машин нечасто буває зручним для людей. У наступному розділі описано, які особливості клітинних автоматів змусили нас відмовитися від цього способу запису.

1.5 Проблеми моделювання КА

1.5.1 Необхідність великої кількості одночасних обчислень

Гнучкість підходу клітинних автоматів до синтезу систем досягається не безкоштовно. Замість невеликого числа змінних, взаємодія яких може

бути задано довільним чином, клітинний автомат використовує багато змінних (одна на клітку), але вимагає, щоб вони взаємодіяли лише локально й одноманітно. Щоб синтезувати структури значної складності, необхідно використовувати велику кількість клітин, а для того щоб ці структури взаємодіяли один з одним і істотно еволюціонували, необхідно дозволити автомату працювати протягом великої кількості кроків. Для елементарних наукових проблем задовільна експериментальна робота може зажадати обчислення мільярдів подій (подією є оновлення однієї клітини); для більш складних додатків може бути бажаним значення в тисячу або мільйон разів більше (тобто 10^{12}-10^{15} подій); в дійсності межі встановлюються тим, скільки ми можемо виконати, а не тим, скільки хочемо [2]. Отже структуру клітинного автомату майже неможливо змоделювати без використання систем з високим рівнем паралелізму.

1.5.2 Неефективність використання табличного запису правил

Якщо повернутися до правила «Parity», розглянутого в розділі 1.2, можна помітити, що його суть дуже легко виразити словами: «Стан клітини відповідає парності її сусідства». Але для того щоб записати його в канонічному табличному виді, треба записати наступний стан клітини для кожного з варіантів станів сусідства, яких для сусідства фон Неймана існує 2^5=32. А якщо спробувати записати в канонічному вигляді правила «Гри життя» Конвея, яка використовує сусідство Мура, нам би довелось записати результат для 2^9=512 варіантів розташування сусідів клітини. Якщо робити це власноруч, це мало того що займе надто багато часу, але й майже напевно призведе до великої кількості прикрих помилок.

Але найбільша проблема з табличними правилами виникає навіть не при їх запису, а ще при їх конструюванні, адже розмір простору правил надзвичайно великий. Для сусідства фон Неймана, так як є 32 позиції та 2 варіанта заповнення кожної з них, кількість ймовірних правил буде

дорівнювати приблизно 4 мільярди, а для сусідства Мура це число виростає до 2^{512}, що дорівнює квадрату приблизної кількості елементарних частинок у Всесвіті. Якщо не застосовувати ніякого систематичного підходу, а, наприклад, задавати правила випадково, ми навіть за все людське життя навряд зможемо побудувати цікавий світ.

Обидві ці проблеми приводять до думки, що для використання усіх можливостей КА, при створенні системи їх моделювання треба розробити зручний для користувача спосіб задання правил, які потім, перед початком обчислення станів клітин, будуть конвертуватися в таблицю або інший, зручний вже для обчислювальної машини, формат. Описані вище існуючі системи моделювання КА використовують для цього власну мову, адже більшість з них були створенні для розв'язання значно більш широкого спектру обчислювальних задач. І хоча ці мови загалом дуже потужні, вивчати їх у повному обсягу для вирішення однієї задачі не завжди ефективно.

Висновки до розділу 1

У першому розділі ми дослідили історію виникнення клітинних автоматів, їх структуру та класифікацію. Також був проведений аналіз таких існуючих систем моделювання КА як Wolfram Mathematica та CAM-6. З цього аналізу та розглянутих проблем, пов'язаних з моделюванням КА можна зробити наступні висновки щодо вимог до розроблюваної системи:

- Система має вміти моделювати роботу клітинних автоматів з усіма розглянутими видами сусідств: фон Неймана, Мура та Марголуса (тобто мати можливість переключатися у режим моделювання блокових КА). Користувач повинен мати змогу використовувати для створення моделей найвідоміші типи правил, як-от: класичні, тоталістичні, блокові.
- Обчислювальна частина системи повинна мати високу швидкодію та значний рівень паралелізму, щоб мати змогу впродовж довгого часу швидко обчислювати стани автомату навіть при великій кількості клітин.
- Система має надавати користувачу можливість власноруч створювати набори правил переходу для КА у зручний та інтуїтивно зрозумілий спосіб. Так як створення мови опису правил – складна задача для розробника, а її вивчення – для користувача, для створення правил КА в нашій системі було вирішено розробити графічний інтерфейс для задання правил різних заздалегідь означених типів (класичні, тоталістичні, блокові).

З огляду на необхідність створення графічного інтерфейсу для системи та бажання легко її розповсюджувати, найкращим засобом її реалізації є веб-додаток, технології розробки яких будуть розглянуті у наступному розділі.

РОЗДІЛ ДРУГИЙ. ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ТА ДЕМОНСТРАЦІЙНИХ МОДЕЛЕЙ

Перед початком проєктування системи, треба визначитися с платформою, для якої вона буде розроблятися: буде це нативний (настільний) застосунок, або веб-додаток?

Настільні застосунки – це тип програмного забезпечення, яке встановлюється безпосередньо на персональний комп'ютер. Користувач може запустити його коли завгодно, незалежно від інших програм. Вони займають місце на жорсткому диску і можуть працювати незалежно від підключення до Інтернету. Хоча деяким програмам це потрібно для функціонування за призначенням (наприклад, веб-браузери, такі як Chrome або Firefox), вони все ще вважаються настільними програмами, оскільки вони встановлені на користувацькому ПК.

Веб-додатки використовуються через Інтернет за допомогою веб-браузерів. Вони зберігають дані не на комп'ютері, а на віддаленому сервері. Веб-браузер дозволяє отримати доступ до програми та її вмісту, а також запускає всі сценарії, що відповідають за її функції.

Для нашої системи найбільш придатною є форма веб-додатку, адже вона забезпечує кросплатформність. Це означає, що неважливо, з якої платформи або пристрою користувач буде запускати наш додаток – він буде працювати однаково, адже всі обчислення будуть проводитися на віддаленому сервері. Так як робота нашої системи вимагає чималої обчислювальної потужності, яка не завжди може бути доступна користувачам, це є істотною перевагою. Також веб-додатки простіше підтримувати та оновлювати – користувачу не потрібно встановлювати нову версію програми прі кожній її зміні. Це дасть нам можливість, наприклад

додати у майбутньому нові типи правил переходу для КА, які одразу стануть доступні користувачам.

2.1 Архітектура веб-додатку

Користувачі Інтернету можуть бачити певну сторінку у своїй системі завдяки низці взаємодій між різними програмними компонентами, користувацькими інтерфейсами, системами проміжного програмного забезпечення, базами даних, сервером та браузером. Структуру та особливості цих взаємодій описує архітектура веб-додатків. Вони складаються з двох різних наборів програм, що виконуються одночасно на різних машинах, та розробляються таким чином, щоб мати можливість обмінюватися даними та гармонічно працювати для забезпечення роботи додатку. Зазвичай ці два набори наступні:

- Користувацька (або клієнтська) частина, яка визначає вигляд та функціонал веб-додатку, з якими буде безпосередньо взаємодіяти користувач.
- Серверна частина, яка працює з базою даних та виконує основну логіку додатку.

Тип архітектури веб-додатку залежить від того, як логіка програми розподілена між сторонами клієнта та сервера. Існує три основних типи архітектури веб-додатків: односторінкова, мікросервісна та безсерверна.

2.1.1 Мікросервісна архітектура

Мікросервісна архітектура – це підхід до розробки додатка як набору невеликих сервісів, кожен з яких працює у своєму власному процесі та взаємодіє з легковажними механізмами, часто через API. Кожна з цих служб побудована навколо власної бізнес логіки і може бути розгорнута незалежно від інших за допомогою повністю автоматизованого механізму розгортання.

Існує мінімум централізованого управління цими службами, яке може бути написане різними мовами програмування [8].

2.1.2 Безсерверна архітектура

Безсерверна архітектура – це хмарна модель розробки, яка дозволяє розробникам збирати та запускати додатки без потреби керування сервісами. Безсерверною її називають не через те, що в ній зовсім немає серверів, а лише через абстрагованість серверів від процесу розробки додатку. Рутинною роботою з надання, обслуговування та масштабування серверної інфраструктури займається постачальник хмарних послуг (наприклад, AWS), а розробникам достатньо просто упакувати свій код у контейнери для розгортання [9].

2.1.3 Односторінкова архітектура

Найпопулярнішою серед розробників на даний момент є односторінкова архітектура веб-додатків. На відміну від класичних веб-додатків, в яких у відповідь на кожну дію користувача браузеру було потрібно завантажувати цілу нову сторінку, односторінковий веб-додаток (англ. single-page application, SPA) взаємодіє з користувачем завдяки постійному оновленню поточної веб-сторінки даними з веб-сервера. В них увесь потрібна браузеру клієнтська частина коду (зазичай це зв'язка JavaScript + HTML + CSS) завантажується або одразу, разом з одним завантаженням сторінки [7], або динамічно довантажуються з серверу за необхідності. Протягом всієї роботи зі сторінкою вона не перезавантажується та не перенаправляє користувача на інші. Це приводить до чималого прискорення роботи додатку.

Треба зауважити, що розглянуті типи архітектури не є взаємовиключними: кожен з мікросервісів може утворювати власний односторінковий додаток, так само як і в безсерверній архітектурі може бути одна чи багато сторінок на користувацькій частині. Але у сервера в нашій

системі основна задача буде лише одна: обчислювати стани автомату та повертати їх на фронтенд, тому розділяти його на мікросервіси чи передавати керування стороннім сервісам не потрібно. Отже майбутня система буде односторінковим веб-додатком.

2.2 Вибір технологій розробки користувацької частини

Зараз, коли мова йде про розробку клієнтську частину веб-додатків, майже завжди йдеться про JavaScript, через його надзвичайно зручну для розробників SPA концепцію фронтенд фреймворків. З їх допомогою створюється шаблон сторінки, і замість того, щоб додавати динамічні частини сторінки, ми просто ставимо заповнювачі (placeholders). Потім, коли користувач виконує дії, які потребують оновлення даних на сторінці, на сервер у фоновому режимі надсилається запит, що вимагає лише необхідні дані (наприклад, лише назву, вміст та зображення). Коли сервер відповідає, він легко замінює заповнювачі на сторінці фактичними даними.

У чому ж перевага такого підходу? По-перше, з точки зору користувача, сторінка плавно оновлюється до запитаних їм даних, не перезавантажуючи сторінку, а оскільки нам потрібно було лише завантажувати дані, це було набагато швидше і менше завантажувалося даних. По-друге, серверу тепер потрібно лише обробляти запити на дані, йому більше не потрібно турбуватися про те, що відбувається з даними, тобто наш сервер обробляє запити набагато швидше, і ми можемо управляти більшим трафіком, не збільшуючи розмір нашого сервера.

До того ж, наш серверний додаток відповідає фронтенду лише даними, а також покращує продуктивність, це також означає, що дані можна використовувати не лише на веб-сайті. Отже бекенд частина коду автоматично функціонує як API (application programming interface, інтерфейс програмування програм). Це означає, що якщо розробники згодом

вирішили, що хочуть створити мобільний додаток для свого сайту або, можливо, дозволити іншим веб-сайтам чи програмам читати дані з нього, вони можуть просто дати їм доступ до API, оскільки це лише дані, і немає обмежень щодо того, як їх потрібно відображати.

На рисунку 2.1 показаний графік динаміки популярності пошуку різних фронтенд фреймворків.

Рисунок 2.1 – Динаміка популярності пошуку фреймворків [11]

З графіку можна побачити, що найчастіше використовуються React, Angular та Vue. Розглянемо кожен з них, щоб обрати найбільш придатний для нашого додатку.

2.2.1 React

Фреймворк React заохочує розробників використовувати реактивний підхід та парадигму функціонального програмування. Додатки, написані на React, розділені на кілька компонентів. Однокомпонентний файл містить як ділову логіку, так і розмітку HTML (що насправді є розміткою JSX, яка трансформується у функції JavaScript). Для зв'язку між компонентами ви можете використовувати Flux або подібну бібліотеку JS.

Незважаючи на те, що гнучкість та свобода у виборі інструментів є його головною перевагою, у React все ж виникають проблеми через свою гнучкість. Коли вам доводиться вибирати з багатьох додаткових бібліотек, ви стикаєтесь з дилемою, що саме ви повинні використовувати з React. Ви можете витратити багато часу, намагаючись з'ясувати найкращі варіанти. В основному, досі немає надійного процесу роботи з React.

Вам доведеться створити власний робочий процес за допомогою React. Це більш складний шлях, ніж просто використання того, що дають вам інші фреймворки JS: безліч готових інструментів, які вбудовані в фреймворк.

2.2.2 Angular

Angular 2+ – це переломний момент в історії Angular framework. Починаючи як надійний конкурент Backbone, AngularJS ледь не застарів, коли вийшов React. Angular істотно змінив свою архітектуру, щоб мати можливість змагатися з React. То що пропонує Angular зараз?

Однією з помітних змін у Angular є перехід з архітектури Model-View-Whatever (можна вважати однією з форм MVC) до архітектури на основі компонентів. Якщо раніше ви могли вставити посилання на цілу бібліотеку AngularJS в основний файл HTML, то тепер ви можете встановити окремі модулі. Ви можете легко ухилитися від встановлення деяких модулів Angular, наприклад @angular/forms або @angular/http, якщо вони вам не потрібні.

Коли ви створюєте програму за допомогою Angular, ви розбиваєте її на кілька компонентів із вкладеними або дочірніми компонентами. Кожен компонент Angular розбивається на три файли. Бізнес-логіка для компонента записана в основному файлі (.ts), тоді як макет та стилі, пов'язані з цим компонентом, записані в інших двох файлах (.html та .css).

Окрім компонентів Angular, розробнику потрібно звикнути до сервісів, системи DI (React не має DI) та директив (спеціальні атрибути HTML).

Найпростішими словами, на відміну від React Angular хоче, щоб фронтенд-код розроблявся строго встановленим чином.

Angular також має перевагу в тому, що додатки на ньому створюються за допомогою статично типизованої мови TypeScript, що на відміну від класичного JavaScript забезпечує безпеку типів, що дуже зручно при роботі зі складними моделями даних.

2.2.3 Vue

Vue, що використовується для розробки користувальницьких інтерфейсів та SPA, є відкритою бібліотекою JS з архітектурою Model-View-View-Model. Популярність Vue завоював завдяки його універсальності, високій продуктивності та оптимальному користувацькому досвіду роботи з веб-додатком. Шаблонний синтаксис Vue поєднує впізнаваний HTML зі спеціальними директивами та функціями. Синтаксис дозволяє розробнику створювати компоненти View.

Хоча екосистема Vue досить широка, і має всі необхідні інструменти, щоб розпочати розробку з Vue, вона все ще не така велика, як React або Angular. Щоб бути точнішим, просто порівняйте кількість плагінів, доступних для React та Vue.js: Різниця у сотнях. Існуючі плагіни, які можна використовувати з іншими фреймворками, часто також не підтримуються.

Після аналізу цих трьох фреймворків, для реалізації клієнтської частини системи, що розробляється, було обрано Angular через можливість використання TypeScript, що зробить зручною роботу з багатьма типами та моделями, присутніми у системі (клітини, сітка, правила) та гнучку компонентну структуру.

2.3 Вибір технологій розробки серверної частини

Бекенд (серверна сторона коду) – це частина веб-сайту, яку не бачить користувач. Вона відповідає за зберігання та впорядкування даних, а також

за те, щоб на стороні клієнта все насправді працювало. Бекенд взаємодіє з інтерфейсом, надсилаючи та отримуючи інформацію, яка відображається як веб-сторінка. Кожного разу, коли користувач заповнюєте контактну форму, вводить веб-адресу або робить покупку (будь-яка взаємодія користувача на стороні клієнта), його браузер надсилає запит на сторону сервера, який обробляє його, звертається до бази даних, якщо це потрібно, та повертає інформацію у вигляді фронтенд коду, який браузер може інтерпретувати та відображати.

Сайт повинен мати додаткові компоненти бекенда, щоб зробити його динамічним веб-додатком, тобто веб-сайтом, вміст якого може змінюватися залежно від того, що є в його базі даних, і який може бути змінений за допомогою даних, введених користувачем. Це відрізняється від статичного веб-сайту, для якого не потрібна база даних, оскільки його вміст, як правило, залишається незмінним.

Як було зазначено в першому розділі, серверна частина системи моделювання КА повинна вміти працювати паралельно, щоб якомога швидше обчислювати стани багатьох клітин. Також їй потрібно працювати зі складною структурою правил, яку найкраще буде реалізувати за допомогою ООП. Враховуючи ці вимоги, розглянемо найпопулярніші мови програмування, на яких розробляють серверну частину веб-додатків, рейтинг яких зображений на рисунку 2.2.

Rank	Change	Language	Share	Trend
1	↑	Python	26.42 %	+5.2 %
2	↓	Java	21.2 %	-1.3 %
3	↑	Javascript	8.21 %	-0.3 %
4	↑	C#	7.57 %	-0.5 %
5	↓↓	PHP	7.34 %	-1.2 %
6		C/C++	6.23 %	-0.3 %
7		R	4.13 %	-0.1 %

Рисунок 2.2 – Динаміка популярності мов для розробки бекенду

З наведеної статистики можна побачити, що найпопулярнішими бекенд-мовами є Python, Java, JavaScript та C#. Розглянемо кожну з них, щоб обрати найбільш придатний для нашого додатку.

2.3.1 Python

Після створення Pythony 1991 році Гвідо ван Россумом, він виріс і став однією з провідних багатоцільових мов програмування. Бекенд-розробники використовують його акуратний та зручний для читання коди для створення функціональних скриптів для виконання серверних задач. Однією з переваг Python є велика схожість з англійською мовою, що робить його дуже читабельним. Тож написання та читання коду на Python відносно легко як для нових, так і досвідчених програмістів. Python користується підтримкою величезних бібліотек, що зменшує необхідність писати код власноруч.

Але незважаючи на популярність цієї мови, використовувати її для розробки нашої системи не має сенсу з багатьох причин. Найголовніша полягає в тому, що в Python майже неможливо створити справді багатопотокову програму через його глобальний інтерпретатор, який

загалом є м’ютексом, що дозволяє виконувати одночасно лише один потік. Як результат, багатопотокові програми, можуть бути повільнішими, ніж однопотокові, які, до речі, на цій мові не дуже швидкі через те, що вона інтерпретується, а не компілюється.

2.3.2 Java

Java є найпопулярнішою мовою програмування у світі – і це недарма. Java не тільки надзвичайно універсальна (її використовують у розробці ПЗ для багатьох пристроїв, від смартфонів до смарт-карт); але й доводить свою надійність розробникам вже протягом 20 років.

Що робить Java настільки універсальною, це віртуальна машина Java (JVM). У багатьох мовах програмування компіляція програми створює код, який може працювати по-різному в залежності від комп’ютера, на кому він запущений. Для Java це не проблема через JVM. Віртуальна машина Java діє як проміжний шар, який може запускати код на будь-якому комп'ютері, незалежно від того, де згаданий код був скомпільований.

Java – це об'єктно-орієнтована мова програмування та вона надає можливості для достатньо зручної розробки багатопотокових програм, але її найвідоміша біда – збиральник сміття – робить додатки на ній повільнішими, ніж на багатьох інших мовах. Також відсутність у Java популярних конструкцій таких як нативне узагальнення методів та класів, перевантаження операторів, структури та поєднання, робить розробку на цій мові не такою зручною, як наприклад на дуже схожому на неї C#.

2.3.3 JavaScript та Node.js

Ми вже докладно розповіли про деякі найпопулярніші фронтенд-фреймворки JavaScript, але як щодо серверної частини? Будучи найпопулярнішою мовою програмування, JavaScript також є однією з найбільш універсальних технологій розробки програмного забезпечення. Традиційно використовуваний як інструмент розробки веб-інтерфейсу, він

також став основним інструментом крос-платформної як мобільної розробки, так і серверної. Одним із інструментів, який вказував на цей зсув у веб-розробці, був Node.js.

Node.js насправді не є фреймворком чи бібліотекою, а середовищем виконання, заснованим на механізмі JavaScript V8 Chrome, який робить його дуже потужним та швидкодійним, а відсутність блокування потоків вводу та виводу дає можливість обробляти декілька запитів одночасно. Але попри це, платформа не підтримує багатопотокове програмування, а мова JavaScript не є об'єктно-орієнтованою, що зробить створення системи правил моделювання КА досить важкою задачею.

2.3.4 C# та ASP.NET

ASP.NET – це відповідь Microsoft на Java Sun Microsystem (нині Oracle). Цей фреймворк використовується для створення серверної частини веб-додатків на таких мовах, як Visual Basic (VB), C#, та F#. Його архітектурний шаблон MVC (Model-View-Controller) дозволяє виконувати обов'язки бекенд-розробки контролером, який взаємодіє з моделлю для обробки даних. Потім результат представляється у поданні для відображення у вигляді зовнішньої веб-сторінки. Зроблений з відкритим кодом у 2016 році .NET може інтегруватися з iOS, Linux та Android за допомогою .NET Core. Код дуже стабільний і надійний, що робить його популярним вибором для бізнесу.

ASP.NET Core забезпечує чудову підтримку використання асинхронних шаблонів програмування. Зараз async реалізований у всіх стандартних класах .NET та у більшості сторонніх бібліотек. Це значно покращує продуктивність фреймворку в обробці запитів, а вбудована в мову C# конструкція Parallel.Foreach дозволяє паралельно виконувати будь-яку кількість обчислень, що є критичним у розробці нашої системи. Також

підтримка ООП та шаблон MVC робить дуже зручною роботу навіть зі складною системою класів.

З огляду на проведений вище аналіз, було вирішено, що найбільш придатною для розробки серверної частини нашої системи є мова C# у поєднанні з фреймворком ASP.NET.

2.4 Алгоритми для демонстрації роботи системи

Яким би гарним не був інтерфейс додатку, при початку роботи з новою системою у користувачів можуть виникнути складнощі з конструюванням власних моделей та правил. Отже, щоб познайомити користувачів із нашою системою та продемонструвати її можливості, корисно додати до неї заздалегідь створені приклади реалізації клітинних автоматів. Для цього ми обрали модель розширення ґратчастого газу в вакуумі для знайомства користувача з блоковими КА, які є менш відомими, ніж класичні КА з сусідством Мура.

Найвідомішим прикладом використання блокових автоматів та сусідства Марголуса є моделювання ґратчастих газів – стилізованих моделей, що складається з частинок, які рухаються з однаковими швидкостями. Кожна клітина КА може знаходитися в двох станах: містити або не містити в собі частинку газу. На кожному часовому кроці здійснюється два процеси, поширення та зіткнення. [3]

На етапі розповсюдження кожна частинка рухатиметься до сусідньої клітини з визначеною швидкістю. За винятком будь-яких зіткнень, частинка зі швидкістю, що рухається вгору, після кроку часу збереже цю швидкість, але буде переміщена на сусідню ділянку над початковою ділянкою. Так званий принцип виключення запобігає переміщенню двох або більше частинок по одній і тій же ланці в одному напрямку.

На етапі зіткнення правила зіткнення використовуються для визначення того, що відбувається, якщо кілька частинок досягають однієї і тієї ж ділянки. Ці правила зіткнення необхідні для збереження масового збереження та збереження загального імпульсу; модель блокових клітинних автоматів може бути використана для досягнення цих законів збереження. Принцип виключення не заважає двом частинкам рухатися по одній і тій же ланці в протилежних напрямках, коли це відбувається, дві частинки проходять одна проти одної, не стикаючись.

Класичним відображенням ґратчастих газів у клітинний автомат є модель HPP, названа так на честь запропонувавших її вчених Гарді, Поме та Пациса (Hardy, Pomeau та de Pazzis). Таблиця правил переходів для цієї моделі зображена на рисунку 2.4 (стани, що симетричні відносно повороту, опущені). Частинки газу , описані цим правилом, рухаються вздовж діагоналей, розлітаючись у різні сторони при зіткненні.

Рисунок 2.4 – Правило HPP-GAS [2]

Але для демонстрації роботи нашої системи це правило буде певним чином розширено: ми додамо стан, що буде відповідати стінкам закритої скляної пляшки, що буде імітувати простір з вакуумом. При зіткненні зі стінками пляшки частинки газу будуть відштовхуватись від них у протилежному до свого руху напрямку.

Також, щоб дати користувачам приклади задання правил для КА, заснованих на сусідстві Мура, було вирішено додати до системи статичні набори, що задають правила «Гри життя» Конвея, як найвідомішого з усіх

КА, та правила 126 для одновимірного КА. Формальні описи цих правил були наведені у першому розділі, результати роботи системи з цими правилами будуть наведені у розділі 4.

Висновки до розділу 2

В другому розділі були розглянуті основні типи архітектури розробки веб-додатків, а саме мікросервісна, безсерверна та односторінкова. Після їх аналізу для розробки нашої системи була обрана архітектура односторінкового веб-додатку.

Були проаналізовані популярні технології розробки різних складових веб-додатків та отримано оптимальний стек технологій розробки:

- Для клієнтської частини була обрана мова Javascript та її фреймфорк Angular.
- Для серверної частини була обрана мова C# у поєднанні з фреймфорком ASP.NET.

Також були обрани моделі для подальшої демонстрації роботи системи після завершення її розробки: модель розширення ґратчастого газу в вакуумі, «Гра життя» та правило 126.

РОЗДІЛ ТРЕТІЙ. РОЗРОБКА СИСТЕМИ МОДЕЛЮВАННЯ КЛТИННИХ АВТОМАТІВ

3.1 Проєктування загальної структури системи

Дана система буде реалізована у вигляді односторінкового веб-додатку, з серверною частиною, написаною на мові C# за допомогою бекенд-фреймворку ASP.NET та з користувацькою частиною, написаною мовою TypeScript за допомогою фронтенд-фреймворку Angular. Узагальнена схема роботи системи зображена на рисунку 3.1. У цьому розділі та далі будемо вважати правилом КА одну пару вхідного стану сусідства клітини (або узагальнення декількох станів спільною умовою) та її вихідного стану, а комплекс цих пар, що застосовуються під час одного проміжку часу – *набором правил* КА. Таке правило, що залежить від стану сусідства Мура даної клітини та має в якості вихідного стану наступний стан цієї клітини будемо називати *правилом Мура*, а таке, що описує перехід з одного стану сусідства Марголуса до іншого будемо називати *блоковим правилом*.

Рисунок 3.1 – Узагальнена схема роботи системи

Серверна частина буде мати API, через який фронтенд буде надсилати до неї початковий стан КА, заданий користувачем, обраний їм набор правил та кількість кроків, які необхідно обчислити. Далі вона буде послідовно застосовувати отриманий набор правил до шарів КА, починаючи з отриманого початкового стану, генеруючи таким чином послідовність шарів.

Коли послідовність досягне питомого розміру, вона буде надіслана назад до фронтенду.

Користувацька частина матиме 2 сторінки. На першій користувач зможе створювати власні набори правил або редагувати існуючі. При створенні нового набору буде можливість обрати тип його сусідства: Мура чи Марголуса. Кожне правило Мура буде містити умову відповідності вхідного стану сусідства клітини цьому елементу та вихідний стан, який набуде клітина при відповідності її сусідства цій умові. Користувач матиме можливість обрати однин з трьох типів вхідних умов для кожного правила:

- «Пряма» умова: користувач обирає, який стан має мати кожен з 8 сусідів клітини та вона сама.
- Тоталістична умова: користувач обирає, які клітини з сусідства беруть участь в обчисленні його суми та чому вона повинна дорівнювати.
- Складена умова: поєднання двох умов будь-якого типу (складеного також) за допомогою логічних операторів NOT, AND, OR та XOR (для оператора NOT потрібне лише одне правило)

На другій сторінці користувач матиме можливість обрати одне з створених правил та задати на канві початковий стан клітинного автомату за допомогою інструментів малювання. Після натиснення кнопки «Simulate» фронтенд регулярно надсилатиме ці дані на серверну частину, щоб отримати безперервний потік наступних станів КА, які будуть почергово відображатися на сітці, доки користувач не натисне кнопку «Stop».

3.2 Розробка серверної частини

Для представлення в програмному коді правил переходу між станами КА були створені два класи: *MooreRule* та *BlockRule*.

3.2.1 Правила Мура

Клас *MooreRule* складається з двох полів:

- *Condition* – умова типу *IMooreCondition*, яка визначає відповідність сусідства якоїсь клітини цьому правилу
- *Result* – число типу sbyte, що позначає стан, який прийме клітина, якщо її сусідство відповідає умові.

Для зручності роботи з сусідствами був створений узагальнений клас *Area<T>,* який є обгорткою над масивом з 9 елементів що входять до сусідства Мура (8 сусідів + сама клітина) та його наслідники:

- *CellStateArea : Area<sbyte>* – для зберігання та обробки станів клітин під час симуляції та завдання умов для прямих правил переходу.
- *BoolArea : Area<bool>* – для маркування клітин сусідства, які слід сумувати у тоталістичних правилах.

Також клас *MooreRule* містить метод *TryApply(CellStateArea cellNeighbors),* який повертає *Result,* якщо передане сусідство відповідає умові та *null* в іншому випадку.

Інтерфейс *IMooreCondition* містить декларацію методу *IsApplicable(CellStateArea cellNeighbors),* який повертає *true*, якщо передане сусідство відповідає цій умові та *false* в іншому випадку, і має три реалізації:

- *DirectCondition – IsApplicable(CellStateArea cellNeighbors)* повертає true якщо кожна клітина *cellNeighbors* має таке ж значення як відповідна клітина поля *ConditionArea* (клітини *ConditionArea*, що мають значення -1 не беруть участь у порівнянні).
- *SumCondition – IsApplicable(CellStateArea cellNeighbors)* повертає true якщо сума клітин *cellNeighbors*, що знаходяться на позиціях, помічених true в полі *CellsToSum*, дорівнює значенню поля *RequiredSum*.

- *ComplexCondition – IsApplicable(CellStateArea cellNeighbors)* повертає true якщо умови з полів *LeftCondition* та *RightCondition*, об'єднанні логічним оператором *Operator* (NOT, AND, OR та XOR) також повертають true.

3.2.2 Блокові правила

Для роботи з блоковими правилами був створений клас *Block*, який є обгорткою над масивом з 4 елементів, що входять до сусідства Марголуса на кожному кроці. Даний клас реалізує інтерфейс *IEquatable<Block>*, що дозволяє напряму порівнювати еквівалентність двох блоків при первірці належності якогось сусідства Марголуса до якогось правила.

Відображенням самого блокового правила є клас *BlockRule* (рис. 3.6). Він має наступні поля:

- *sbyte Phase* - позначає номер фази, до якої може бути застосовано це правило (0 чи 1). Якщо значення цього поля дорівнює -1, то правило може бути застосовано до обох фаз.
- *Block From* – визначає, який поточний стан повинно мати сусідство Марголуса, щоб до нього можна було застосувати дане правил.
- *Block To* – визначає наступний стан сусідства Марголуса, якщо до нього було застосоване дане правило.

3.2.3 Симуляція еволюції КА

Для зберігання стану КА було створено клас Matrix (рис. 3.7), який є обгорткою над двовимірним масивом чисел типу *sbyte* та має наступні методи:

- *int GetCycledI(int i), int GetCycledJ(int j)* – повертає відповідну перетворену координату з урахуванням «заклеювання» країв матриці. Тобто координата -1 перетвориться на координату останньої комірки масиву, а координата, що дорівнює його довжині перетвориться на 0.

- *CellStateArea GetMooreNeighborhood(int i, int j)* – повертає поточний стан сусідства Мура для клітини з вказаними координатами.
- *Block GetMargolusBlock((int, int) blockCorner)* – повертає поточний стан сусідства Марголуса по переданим координатам верхньої лівої клітини блоку.
- *UpdateMargolusBlock((int i, int j) blockCorner, Block blockToInsert)* – оновлює стан блока відповідно до застосованого блокового правила.

Сама симуляція відбувається в допоміжному статичному класі *CAHelper*. Він має два методи, один з яких відповідає за симуляцію КА за набором правил Мура, а другий – за набором блокових правил:

- *sbyte[][][] SimulateMoore(Matrix start, IEnumerable<MooreRule> ruleSet, int steps, sbyte defaultValue)* – застосовує набор правил Мура *ruleset* до шарів КА, починаючи із стану *start* протягом кількості кроків, що дорівнює *steps*. На кожному кроці за допомогою конструкції *Parallel.ForEach* усі клітини обробляються паралельно та незалежно одна від одної. До кожної з них по черзі намагаються застосуватися правила з отриманого набору, і перше правило, умові якого відповідатиме сусідство клітини, поверне її новий стан. Якщо жодне правило не можна застосувати до клітини, її новий стан буде дорівнювати отриманому defaultValue (якщо *defaultValue* == -1, стан клітини не зміниться). Метод повертає тривимірний масив, який є послідовністю станів КА, тобто масивом матриць-шарів.
- *sbyte[][][] SimulateBlock(Matrix start, IEnumerable<BlockRule> ruleSet, int steps)* – застосовує набір блокових правил *ruleset* до шарів КА, починаючи із стану *start* протягом кількості кроків, що дорівнює *steps*. На кожному кроці визначається поточна фаза (0

чи 1) та на основі неї формується список клітин, які складають верхні ліві кути блоків соседства Марголуса на даній фазі. За допомогою конструкції *Parallel.ForEach* усі блоки обробляються паралельно та незалежно один від одного. До кожного з них по черзі намагаються застосуватися правила з отриманого набору, і перше правило, поле From якого збігатиметься з поточним блоком, поверне новий стан для цього блоку.. Якщо жодне правило не можна застосувати до цього блоку, його стан не зміниться. Метод повертає тривимірний масив, який є послідовністю станів КА, тобто масивом матриць-шарів.

3.2.4 Зв'язок з фронтендом

Зв'язок серверу з бекендом забезпечує клас ApiController, який задає дві точки входу до серверної частини: *simulateMoore/* та *simulateBlock/,* які викликають відповідні методи допоміжного класу *CAHelper.* Дані приймаються об’єктами у форматі *JSON*, схему полей для яких задають класи *MooreStartConditions* та *BlockStartConditions* (рис. 3.2).

Можна побачити, що для передачі інформації про набори правил використовуються проміжні класи **Model* (рис 3.2). Це було зроблено з метою зменшення кількості вкладених класів, використання простіших структур при обміну даними та вирішення проблеми передачі правил Мура, яка полягає в тому, що вони усі мають різні поля та невідомо, скільки умов якого типу вибере користувач в надісланому на бекенд наборі. Тому до кожної з імплементацій інтерфейсу *IMooreCondition* було додано конструктор, який приймає модель типу object та за допомогою JSON-десереалізації перетворює її на об’єкт цієї імплементації. Також було створено допоміжний клас *RuleHelper,* який в залежності від змісту поля *ConditionType* класу *MooreRuleModel* перетворює його внутрішню

узагальнену модель умови на об’єкт типу *IMooreCondition* за допомогою описаних вище конструкторів.

Рисунок 3.2 – Діаграма класів моделей правил, *MooreStartConditions* та *BlockStartConditions*

3.3 Розробка клієнтської частини

3.3.1 Моделі правил

Моделі правил на фронтенді повністю еквівалентні моделям правил на бекенді, адже саме ними вони обмінюються одне з одним. Але на фронтенді були додані класи наборів правил – *MooreRulesSet* та *BlockRulesSet,* що складаються з двох полів:

- *Rules* - *MooreRuleModel[]* для набору правил Мура або *BlockRuleModel[]* для набору блокових правил
- *ColorMap* – схема кольорів, що буде застосовуватися при відображенні цього правила.

Клас *ColorMap* містить два поля:

- *statesToColors: Map<number, string>* - при створенні правил користувач має можливість створювати правила з довільною кількістю станів. Це поле зберігає відповідність обраному користувачем числовому значенню певного стану кольору, яким

будуть розмальовуватися клітини, що мають даний стан, при симуляції.

- *currentState: number* – при створенні початкового стану КА та редагуванні шаблонів правил користувачеві потрібно буде «розмальовувати» певні клітини певними кольорами. Це поле зберігає поточний колір «пензлика» яким редагаються клтини поточного правила чи стану КА, що буде еволюціонувати за цим правилом.

3.3.2 Сервіси

Для пересилки зберігання створених користувачем правил та обміну даним про них між компонентами був створений сервіс *RulesService.* У полях mooreRuleSets та blockRuleSets що мають типи відповідно Map<string, MooreRulesSet> та Map<string, BlockRulesSet> зберігаються набори правил з призначеними їм іменами. Сервіс має наступні методи:

- *getRuleSetsNames(): string[]* – повертає список імен усіх збережених наборів правил
- *getRuleSetType(ruleSetName: string): SimulationType* – повертає тип набору правил з даним іменем *(*SimulationType – це перелік з двома можливими значеннями: *Moore* та *Block)*
- *getMooreRulesSet(ruleSetName: string): MooreRulesSet* – повертає набір правил Мура з заданим ім’ям
- *getBlockRulesSet(ruleSetName: string): BlockRulesSet* – повертає набір блокових правил з заданим ім’ям
- *setBlockRuleSet(ruleSetName: string, rule: BlockRulesSet)* – зберігає набір блокових правил під вказаним ім’ям
- *setMooreRuleSet(ruleSetName: string, rule: MooreRulesSet)* – зберігає набір правил Мура під вказаним ім’ям

Сервіс *DataService* забезпечує зв'язок з серверною частиною та зберігає проміжні дані. За допомогою техніки DI у його конструктор впроваджуються HTTP-клієнт та *RulesService.* Він має наступні методи:

- *fetchSimulationResults(startMatrix: Array<Array<number>>, ruleSetName: string, steps: number) – отримує з сервісу правил набір правил, що буде використовуватися, та в залежності від його типу надсилає відповідний запит до серверу та повертає отриману від послідовність обчислених станів КА.*
- *saveState(matrix: Array<Array<number>>) – зберігає стан КА при переході між сторінками.*
- *getSavedState() : Array<Array<number>> – повертає збережений сан КА.*

3.3.3 Компоненти

Так як фронтенд розроблявся за допомогою фреймворку Angular, його графічний інтерфейс складається з компонентів. Можна побачити, що веб-додаток має дві сторінки: «Simulation» та «Rules Editor», переключення між якими відбувається за допомогою навігаційної панелі у верхній частині екрану або через перехід до відповідних адрес */simulation* та *rules-editor/.*

Компонент екрана редагування правил та їх наборів (рис. 3.3) містить список збережених наборів правил, згрупованих по їх типу, у лівій частині екрану. Кожна група має кнопку «Add rule set», натискання якої створить порожній набор правил відповідного типу. У правій частині знаходяться налаштування кольорової схеми поточного набору правил. Вона є списком елементів, кожен з яких встановлює відповідність числового відображення стану клітини у її колір. Користувач може додавати елементи, редагувати та видаляти їх. При натисканні на елемент, стан, записаний в ньому, обирається як поточний і буде використатися при редагуванні правил. Також кольорова схема містить допоміжний елемент, що має стан -1. Він не буде

відображатися при симуляції, а при редагуванні правила буде позначати, що стан клітини, помічний даним кольором не враховуватиметься при визначенні відповідності клітини цьому правилу.

Рисунок 3.3 – Вигляд сторінки редагування правил та їх наборів

В центральній частині екрану знаходиться безпосередньо компонент редагування обраного набору правил, який трохи відрізняється в залежності від типу набору. У верхній частині розташовані назва поточного набору, яку користувач може редагувати, кнопка збереження, кнопка видалення, та кнопка додавання нового правила.

У випадку блокового набору правил при натисканні на кнопку «Add rule» в центральну частину додається компонент блокового правила (рис. 3.4). Він містить 2 квадрати 2x2 клітини кожен та кнопку видалення. Лівий квадрат – це вхідний стан, який повинно мати сусідство Марголуса, щоб до нього можна було застосувати це правило, правий – його наступний стан, якщо правило було застосовано. Користувач може натискати на будь-яку клітину і вона забарвиться у колір поточного обраного стану.

Рисунок 3.4 – Компоненти блокових правил

При створенні або редагуванні набору правил Мура, кнопка «Add rule» є випадаючим списком, що дозволяє користувачу обирати, яка умова буде використана в цьому правилі, та при виборі одного з наведених нижче варіантів додає на екран компонент відповідного правила. Кожен компонент містить відображення певної умови, вихідний стан клітини та кнопку видалення. При натисканні на клітину вихідного стану з'являється меню вибору кольору із поточної кольорової схеми, заданої користувачем у правій панелі. Вигляд умови залежить від обраної опції списку «Add rule»:

- «Direct Rule» створює правило з прямою умовою (рис. 3.5), яка є зображенням сусідства Мура, тобто квадратом 3x3 клітини, стан кожної з яких користувач може задавати (якщо стан якоїсь клітини не слід враховувати, її можна відмітити станом «-1»)

Рисунок 3.5 – Компонент правил Мура з прямою умовою

- «Sum Rule» створює правило з тоталістичною умовою (рис. 3.6), яка є зображенням сусідства Мура, кожна з клітин якого може мати лише два стани: *true*, якщо її слід враховувати при обчисленні суми сусідства, та *false* в іншому випадку. Стан клітини змінюється на протилежний при натисканні на неї. Також рядом з ним є поле, де користувач може ввести число, що позначатиме суму, яку повинно мати сусідство, щоб до нього можна було застосувати це правило.

Рисунок 3.6 – Компонент правил Мура з тоталістичною умовою

- «Complex Rule» створює правило зі складеною умовою (рис. 3.7), яка містить три елементи: випадаючий список для вибору оператора, яким будуть поєднанні внутрішні умови, та два випадаючих списки для вибору типу лівої та правої умови. Після вибору одного з типів на місці списку з'явиться елемент редагування відповідної умови.

Рисунок 3.7 – Компонент правила Мура зі складеною умовою

Найважливішим елементом компоненту екрана симуляції (рис. 3.8) є сітка, на якій відображається поточний стан клітинного автомату. Вона є HTML-елементом *Canvas* та реалізована за допомогою бібліотеки *p5.js*. Був створений клас *CellGrid*, який розширяє клас *p5,* та має наступні поля та методи:

- *colorMap: ColorMap* – кольорова схема поточного набору правил, задана користувачем при його створенні
- *gridWidth: number, gridHeight: number* – ширина та висота сітки
- *currentLayer: Array<Array<number>>* - поточний стан клітинного автомата, що зараз відображається на екрані, заданий через числове відображення станів, задане користувачем

- *setup()* – перевизначення стандартного методу *p5*, який створює елемент *Canvas*, задає його параметри та обчислює розміри майбутньої сітки виходячи з розміру екрану користувача
- *draw()* – перевизначення стандартного методу *p5,* який реалізує найголовнішу її функціональність: безперервне малювання за заданим алгоритмом. За допомогою цього методу *Canvas* відображає поточний стан клітинного автомату в відповідній кольоровій схемі та динамічно оновлюється відповідно до кожної зміни поля *currentLayer*. На кожному кроці за допомогою методу *drawBackground* малюється порожня сітка, та для кожного числа з масиву *currentLayer* малюється клітина потрібного кольору.
- *mouseClicked(), mouseDragged()* – перевизначення стандартних методів *p5,* що дозволяють користувачу «малювати» по сітці поточним кольором за допомогою миші, задаючи тим самим початковий стан КА перед симуляцією, або оновлюючи стан при зупинці симуляціїї
- *clearGrid()* – очищує сітку, заповнюючи її першим зі створених користуачем станів
- *fillWithStatic()* – заповнює сітку випадковими значеннями з кольорової схеми поточного правила
- *drawBackground()* – допоміжний метод, який визначає правила малювання границь клітин
- *setCell(mouseX: number, mouseY: number)* – допоміжний метод для користувацького малювання, що оновлює стан клітини в полі *currentLayer.*
- *drawCell(i: number, j: number)* - допоміжний метод, який визначає правила розфарбування клітин в залежності від кольорової схеми та поточного стану поля *currentLayer.*

Рисунок 3.8 – Вигляд сторінки симуляції

Зверху від сітки розташована панель налаштувань, де знаходяться кнопки початку та зупинки симуляції, очищення сітки та заповнення її випадковими значеннями. Також на ній розташований випадаючий список, з якого користувач може обрати один з існуючих наборів правил перед редагуванням початкового стану та початком симуляції, та випадаючий список, зо дозволяє обрати колір для малювання по сітці з кольоровий схми обраного набору правил.

Висновки до розділу 3

В даному розділі було спроектовано та розроблено веб-додаток системи комп'ютерного моделювання клітинних автоматів. Були детально описані складові класів серверної та користувацької частин програми, наведені

алгоритми роботи основних модулів системи та описані принципи їх взаємодії.

Також були наведені:

- UML діаграми класів бекенду
- діаграма зв'язку компонентів фронтенду
- скріншоти вигляду основних компонентів користувацького ітерфейсу

Для розробки серверної частини було використано мову C# у поєднанні з фреймворком ASP.NET, для розробки клієнтської частини – мову JavaScript та її фреймворк Angular.

РОЗДІЛ ЧЕТВЕРТИЙ. ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

4.1. Демонстрація роботи заздалегідь створених наборів правил

Для того щоб користувач міг познайомитися з розробленою системою та знав її можливості, було вирішено додати до неї декілька заздалегідь створених наборів правил, список яких був описаний у другому розділі. Користувач може подивитися, як вони побудовані, модифікувати їх за потреби або запустити симуляцію одного з них одразу після запуску додатку. Також користувач може змінювати не увесь набор правил, а лише його кольорову схему

Перш за все, був доданий набір правил для створення найвідомішого клітинного автомату, а саме «Гри життя» Конвея, формальний опис якої був наведений у першому розділі. Набор її правил у розробленій системі зображено на рисунку 4.1. Можна побачити, що воно складається усього з двох елементів:

- тоталістичного правила «якщо сума сусідства клітини за виключенням неї самої дорівнює трьом, клітинна стає живою»
- тоталістичного правила «якщо сума сусідства клітини клітин за виключенням неї самої дорівнює двом» та прямого правила «якщо сама клітина є живою, то вона залишаться живою», які об’єднані між собою у складене правило за допомогою логічного оператору «AND»

Кольорова схема складається з двох станів: 0 – «мертва клітина», та 1 – «жива клітина». Стан -1 є допоміжним у заданні правил та не може бути відредагований користувачем, так як він не з’явиться на екрані симуляції.

Рисунок 4.1 – Задання правил «Гри життя»

Також був створений набір для демонстрації роботи одновимірних клітинних автоматів за правилом, заданим у коді Вольфама 126 (рис. 4.2). Він складається з дев'яти прямих правил, 8 з яких описують правило отримання нового покоління клітин, а останнє є службовим правилом, за допомогою якого покоління живих клітин залишаються на екрані.

Рисунок 4.2 – Задання правила 126

Для демонстрації можливостей роботи з блоковими клітинними автоматами було додано найпростіше правило опису ґратчастого газу HPP, у відповідності до якого клітки рухаються вздовж діагоналей, розлітаючись у протилежних напрямках при зіткненнях. Опис цього правила в нашій системі зображений на рисунку 4.3. В ньому наявні усі 16 можливих вхідних

станів сусідства Марголуса та вихідний стан для кожного з них. Так як жоден вихідний стан не повторюється, цей КА є оборотним.

Рисунок 4.3 – Задання правила HPP

Демонстрацію роботи набору правил «Гра життя» найкраще проводити починаючи із випадкового заповнення сітки за допомогою спеціальної кнопки з панелі налаштувань:

Рисунок 4.4 – Випадкове заповнення сітки

Після натискання кнопки «Start» можна спостерігати еволюцію автомату, яка пройде, наприклад, етап, зображений на рисунку 4.5, на якому

можна бачити характерні для цього КА фігури, як-от глайдери, блоки, блінкери та інші.

Рисунок 4.5 – Один з етапів еволюції «Гри життя»

Результат роботи набору правил 126 для одновимірного клітинного автомату (початковий стан – єдина жива клітина у верхньому рядку сітки) можна побачити на рисунку 4.6.

Рисунок 4.6 – Результат роботи правила 126

4.2. Демонстрація редагування правила та його роботи після змін

Як було описано в другому розділі, за основу для моделі розширення ґратчастого газу в вакуумі буде взято наявне в системі правило HPP, до існуючих в якому станів, що позначають частинки газу та пустий простір повинен бути доданий стан, що позначає стінки пляшки, яка буде обмежувати простір з вакуумом всередині. Для цього перейдемо на сторінку набору правил HPP, та натиснемо на кнопку «Add color», як зображено на рисунку 4.7.

Рисунок 4.7 – Додавання нового стану

Виберімо колір для нового стану за допомогою спеціального інструменту (рис 4.8), а також змінимо колір частинок газу.

Рисунок 4.8 – Інструмент вибору кольору

Після цього треба розширити набор новими правилами, що будуть описувати поведінку частинок при зіткненні зі стінками пляшки. На рисунку 4.9 показано натискання кнопки додавання правила та порожній елемент правила, що з’явився після цього.

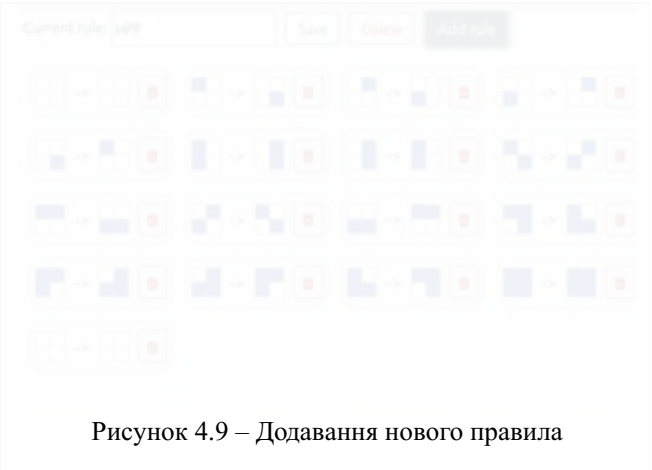

Рисунок 4.9 – Додавання нового правила

Задамо прості правила взаємодії газу із пляшкою: якщо на шляху діагонального руху частинки опиняється шматок пляшки, незважаючи на стан інших клітин частинка починає рухатися у зворотному напрямку. У розробленій системі ці правила виглядають так як зображено на рисунку 4.10.

Рисунок 4.10 – Модифіковане правило HPP

Для демонстрації роботи цих правил перейдемо на екран симуляції та намалюємо на його сітці пляшку за допомогою стану 2 та концентровану хмару газу всередині неї за допомогою стану 1(рис. 4.11).

Рисунок 4.11 – Початковий стан симуляції: пляшка із концентрованим газом всередині

Після цього натиснемо кнопку «Start» та розпочнемо симуляцію. Ми побачимо, як частинки газу спочатку розлітаються в різних напрямках впорядкованими хмарками (рис. 4.12), але після зіткнення з нерівними стінками пляшки їх рух стає більш хаотичним та вони починають літати по всьому її об’єму (рис. 4.13).

Рисунок 4.12 – Клітини розлітаються у різних напрямках впорядкованими хмарками

Рисунок 4.12 – Частинки газу рухаються хаотично, відштовхуючись від нерівних стінок пляшки

Отже, може побачити, що система працює як очікувалось, та відповідає усім поставленим вимогам. Графічне задання правил показало себе набагато швидшим та зручнішим, ніж табличне та навіть ніж задання предвстановлених наборів правил у програмному коді розробленої системи.

Висновки до розділу 4

В даному розділі були продемонстровані основні можливості розробленої системи. Були описані набори правил, що наявні в системі за замовчуванням: «Гра життя» Конвея, правило 126 для одновимірного клітинного автомату та правило HPP для блокового автомату.

Також був описаний процес створення власних правил на прикладі алгоритму, обраному у другому розділі, а саме алгоритм моделювання розширення ґратчастого газу в вакуумі, для якого був показаний процес створення власного правила шляхом модифікації існуючого. Були наведені результати моделювання роботи цього алгоритму у розробленій системі.

ВИСНОВОК

Метою даною дипломної роботи було розробити систему, що надавала б користувачам можливість створювати різноманітні клітинні автомати та моделювати процес їх еволюції.

У першому розділі було оглянуто предметну область, й опрацьовано деякі теоретичні аспекти, що стосуються теми даної роботи. Також було проаналізовано деякі існуючі системи моделювання клітинних автоматів та на основі цього аналіз сформульовано вимоги до розроблюваної системи.

У другому розділі були розглянуті технології, за допомогою яких можна було реалізувати поставлену задачу. Окремо розглядався технології написання серверної та користувацької частин коду. Після аналізу усіх варіантів, з них були обрані найбільш придатні для розроблюваної системи.

У третьому розділі буди детально описані складові програмного коду, що реалізує питому систему. Були описані поля та методи складових класів серверної та користувацької частини, принципи роботи основних алгоритмів. Також були наведені скріншоти окремих компонентів, що складають користувацький інтерфейс.

У четвертому розділі була продемонстрована робота розробленої системи на заздалегідь створених правила. Також був показаний процес створення власних правил та роботи систем на їх основі.

Під час розробки системи були враховані усі вимоги до неї, з наведених результатів її роботи можна побачити, що вона працює коректно та відповідає цим вимогам.

Схожість

Джерела з Інтернету 13

4	http://www.dut.edu.ua/uploads/p_1436_30244920.pdf	0.33%
5	https://mathematica.stackexchange.com/questions/73457/modifying-game-of-life-code-so-that-the-lattice-can-be-seeded-wit...	0.33%
7	https://ela.kpi.ua/bitstream/123456789/36404/1/Svetla_bakalavr.pdf	0.17%
8	http://www.chnu.edu.ua/res/chnu.edu.ua/Tezu%20inst%20fiz.pdf	0.17%
14	https://wikizero.com/uk/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D1%96%D1 4 джерела	0.1%
20	https://moodle.znu.edu.ua/pluginfile.php?file=/450642/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86 3 джерела	0.09%
22	https://znaimo.com.ua/%D0%9A%D0%BB%D1%96%D1%82%D0%B8%D0%BD%D0%BD%D0%B8%D0%B9_%D0%B0%D0%B2%...	0.09%
24	https://ukrbukva.net/92459-Ispol-zovanie-kletochnyh-avtomatov.html	0.08%

Джерела з Бібліотеки 135

1	СІВАЧЕНКО_Марина_81cc45_main_part	ID файлу: 1008240347	Навчальний заклад: National Technical 5 Джерело	1.05%
2	Студентська робота	ID файлу: 1008222980	Навчальний заклад: Lviv Polytechnic National University 22 Джерело	0.5%
3	БАРАН_Данило_27ab01_main_part	ID файлу: 1005718473	Навчальний заклад: National Technical University of ...	0.47%
6	Основна частина Овсієнко	ID файлу: 1000068001	Навчальний заклад: National Technical University of Ukrain...	0.3%
9	ТОЛОК_Семен_1af6e6_main_part	ID файлу: 1000748197	Навчальний заклад: National Technical Unive 4 Джерело	0.16%
10	Студентська робота	ID файлу: 1008015371	Навчальний заклад: Lviv Polytechnic National University	0.13%
11	Студентська робота	ID файлу: 1005760550	Навчальний заклад: Yuriy Fedkovych Chernivtsi National University	0.11%
12	Dzenik_last	ID файлу: 1004140973	Навчальний заклад: National Technical University of Ukraine “Kyiv Polytechn...	0.1%
13	Студентська робота	ID файлу: 1000768692	Навчальний заклад: Cherkasy State Technological University	0.1%
15	Студентська робота	ID файлу: 5201158	Навчальний заклад: Yuriy Fedkovych Chernivtsi National Uni 2 Джерело	0.1%
16	Дудка.Вадим.ІВ71	ID файлу: 1008185647	Навчальний заклад: National Technical University of Ukrain 2 Джерело	0.1%

17	Revnyk_Diplom	ID файлу: 1000704870	Навчальний заклад: National Technical University of Ukraine "К	11 Джерело	0.09%
18	ЯВОРСЬКИЙ_Іван_1cd0c6_main_part	ID файлу: 1008249854	Навчальний заклад: National Technical U	18 Джерело	0.09%
19	Студентська робота	ID файлу: 2091455	Навчальний заклад: Lviv Polytechnic National University		0.09%
21	Студентська робота	ID файлу: 1000094924	Навчальний заклад: Lviv Polytechnic National University	30 Джерело	0.09%
23	Кузнєцов К А	ID файлу: 1000801712	Навчальний заклад: National Technical University of Ukraine "К	29 Джерело	0.08%
25	Pedorenko_magistr_ip92mp_2	ID файлу: 1005755864	Навчальний заклад: National Technical University of Ukrain...		0.08%
26	Студентська робота	ID файлу: 1003790539	Навчальний заклад: National University of Life and Environmenta...		0.08%
27	ГОЛОТА_Михайло_ece1c2_main_part	ID файлу: 1003852687	Навчальний заклад: National Technical U	2 Джерело	0.08%
28	Студентська робота	ID файлу: 1960142	Навчальний заклад: Lviv Polytechnic National University		0.08%