Міністерство освіти і науки України

Національний технічний університет України „Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки

інформації та управління

Спеціальність

„ Програмне забезпечення систем ”

**ЗВІТ**

з науково-дослідної практики

на тему :

„ ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ ТРИВИМІРНИХ КЛІТИННИХ АВТОМАТІВ”

Місце проходження практики: кафедра АСОІУ НТУУ «КПІ ім. Ігоря Сікорського»

**Виконав студент**

(шифр, прізвище, ім’я, по батькові )

*гр.ІП-51м Котолупов Олег Владиславович*

**Керівник практики**

(посада, прізвище, ім’я, по батькові )

*доц. Ліщук К.І.*

**Керівник диплому**

(посада, прізвище, ім’я, по батькові )

*доц., Фіногенов О.Д.*

Дата захисту ­­­­­­­­­­­­ ­\_7.03.2017\_\_\_\_\_\_\_\_\_ Оцінка „\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”

Київ 2017

# Вступ

Метою даної магістерської дисертації є розробка застосунку, що реалізовуватиме роботу з тривимірними клітинними автоматами задля вивчення їх властивостей та пошуку автоматів, що виказують ознаки хаотичної поведінки та містять конфігурації, що здатні зберігати свою цілісність протягом багатьох ітерацій.

Об’єктом досліджень є тривимірні клітинні автомати із правилами, що базуються на грі “Життя”. Сітка даного клітинного автомату є кубічною та скінченною. Стан клітини на наступному кроці при цьому обраховується на основі кількості сусідів даної клітини. Сусідами клітини виступають 26 клітин, чиї координати відрізняються від координати центральної клітини не більше, ніж на одиницю. Кожна клітина може приймати один з двох станів: “жива” та “мертва”.

Предметом дослідження є пошук тривимірних клітинних автоматів 4 класу, які знаходяться на границі хаосу та впорядкованості, типовим прикладом яких серед двовимірних клітинних автоматів є гра “Життя”. Подібні автомати можуть містити стійкі структури на кшталт натюрмортів чи осциляторів з гри “Життя”, а також хаотичну реакцію на незначні зміни, що може відбуватись тисячі ітерацій.

Актуальність теми базується на низькій дослідженості характеристик тривимірних клітинних автоматів. Двовимірні клітинні автомати, зокрема гра “Життя”, вже досить добре вивчені; вони також активно застосовуються у різних галузях науки для моделювання певних процесів, а також вплинули на різні галузі, не пов’язані безпосередньо з клітинними автоматами. Тривимірні автомати не мають такого широкого застосування та не були детально вивчені, хоча можуть мати аналогічний потенціал.

# Огляд існуючих рішень.

## Загальний огляд клітинних автоматів.

### Основні поняття клітинних автоматів.

Клітинний автомат (КА) — дискретна математична модель, яка визначає сукупність та описується набором клітинок, що утворюють періодичну решітку, та заданими правилами переходу, що визначають стан клітини за теперішнім станом самої клітинки та тих її сусідів, що знаходяться від неї на певній відстані, яка не перевищує максимальну.

Поняття клітинних автоматів доволі обширне, тому можна знайти доволі багато різних визначень. Найпоширенішими є:

* математичний об'єкт з дискретним простором та часом;
* регулярна структура двійкових скінченних автоматів з однаковими правилами переходів, що виражені у вигляді булевих функцій від станів сусідніх автоматів;
* стилізовані, синтетичні світи, що визначені простими правилами, подібно правилам настільної гри;
* математична ідеалізація фізичної системи, в якій час та простір дискретні, а фізичні величини приймають скінченну множину значень.

Класичні КА в загальному випадку відповідають наступним критеріям:

* зміна значень всіх клітинок відбуваються одночасно після обчислення нового стану кожної клітинки решітки. Інакше порядок перебору клітин решітки при проходження ітеративного процесу суттєво впливав би на результат;
* решітка однорідна. Неможливо відрізнити жодні два місця на решітці по ландшафту. Однак на практиці решітка виявляється кінцевою множиною клітин (адже неможливо виділити необмежений об'єм даних). В результаті можуть мати місце крайові ефекти: клітини, що стоять на межах решітки будуть відрізнятися за кількістю сусідів. Щоб уникнути цього можна ввести періодичні крайові умови;
* взаємодії локальні. Лише околишні клітинки (як правило, сусідні) здатні вплинути на дану клітинку;
* множина станів клітинки кінцева. Ця умова потрібна, щоб для отримання нового значення стану клітини треба було виконати кінцеву кількість операцій (але це не заважає використовувати клітини для зберігання чисел із плаваючою комою для розв'язку прикладних задач).

Якщо з будь-якого початкового стану можна привести клітинного автомат в будь-яку задану конфігурацію шляхом варіювання значення загального вхідного параметра, такий КА називають повним.

У кожний момент часу кожен елемент КА приймає один стан зі скінченного набору станів. В залежності від цих станів в наступний момент часу набір елементів може прийняти новий стан. Якщо для елементів КА множини можливих станів відрізняються, такий клітинний автомат називається полігенним. Але на практиці використовуються комірки з еквівалентною множинами можливих станів алгебраїчною структурою – лінійні КА.

Елементи можуть бути геометрично розташовані різноманітним чином. Розмірність простору може бути довільною, а число елементів – як безкінечним, так і скінченним. В останньому випадку виникає додаткова міра свободи в граничних умовах. Вони можуть бути різними, але на практиці використовуються постійні у часі (найчастіше – нульові) або періодичні граничних умовах. У динамічних КА геометрія може змінюватися з часом, а якщо геометрія різна на різних ділянках простору, такі кліткові КА називають неоднорідними.

Для КА визначається поняття сусідів – елементів, від яких залежить елемент КА. Можна назвати поняття сусідства ключовим для КА. При тому сусідство розуміється не в геометричному сенсі, а в інформаційному. Хоча зазвичай інформаційний сенс накладається на геометричний. Сусідство одиничних автоматів встановлюється постійним для кожного одиничного автомата решітки і визначається спеціальним вектором – індексом сусідства. Як правило, розглядаються d-мірні регулярні решітки, в цілочислові точки яких поміщені копії деякого автомата Мура. Стан елемента в наступний момент часу обчислюється із стану самого елементу і його сусідів. Сусідство у більшій мірі визначається геометрією КА. Для різних цілей можлива зміна числа вхідних станів елемента. Якщо для кожного елемента КА число входів і виходів однакове, такий КА називається збалансованим.

Для кожного клітинного автомату визначені певні локальні правила. Відповідно до локального правила змінюється стан елемента КА протягом часу. КА, в якому локальні правила різні для різних елементів, називається різнорідним. Локальне правило може бути недетермінованим, тобто змінюватися в часі або мати випадкову природу. Зазвичай, це правило встановлює залежність між поточним станом елементу та його сусідів та майбутнім станом цього елементу на наступному кроці.

Основний напрям дослідження клітинних автоматів — алгоритмічна розв'язність окремих задач. Також розглядаються питання побудови початкових станів, при яких клітинний автомат вирішуватиме задану задачу. Залишається відкритим, наприклад, питання про можливість побудови машини Тюринга у грі «Життя».

### Класифікація клітинних автоматів.

Стівен Вольфрам у своїй книзі A New Kind of Science запропонував 4 класи, на які всі клітинні автомати можуть бути поділені в залежності від типу їх еволюції. Класифікація Вольфрама була першою спробою класифікувати самі правила, а не типи поведінки правил окремо. В порядку зростання складності класи виглядають наступним чином:

* Клас 1: Результатом еволюції майже всіх початкових умов є швидка стабілізація стану та його гомогенність. Будь-які випадкові конструкції в таких правилах швидко зникають.
* Клас 2: Результатом еволюції майже всіх початкових умов є швидка стабілізація стану, або виникнення коливань. Більшість випадкових структур в початкових умовах швидко зникає, але деякі залишаються. Локальні зміни в початкових умовах надають локальний характер на подальший хід еволюції системи.
* Клас 3: Результатом еволюції майже всіх початкових умов є псевдо-випадкові, хаотичні послідовності. Будь-які стабільні структури, які виникають, майже відразу ж знищуються оточуючим їх шумом. Локальні зміни в початкових умовах надають широкий, невизначений вплив на хід усієї еволюції системи.
* Клас 4: Результатом еволюції майже всіх правил є структури, які взаємодіють складним і цікавим чином з формуванням локальних, стійких структур, які здатні виживати тривалий час. В результаті еволюції правил цього класу можуть виходити деякі послідовності Класу 2, описаного вище. Локальні зміни в початкових умовах надають широкий, невизначений вплив на хід усієї еволюції системи. Деякі клітинні автомати цього класу мають властивість універсальності по Тьюрингу. Останній факт був доведений для Правила 110 і гри «Життя».

Крім цього, клітинні автомати також класифіковані за іншими критеріями. За порядком переходу елементів в новий стан клітинні автомати поділяються на синхронні та асинхронні. У синхронних КА всі клітинки переходять у новий стан одночасно за сигналом глобального таймера. При цьому як вхідні стани використовуються старі стани сусідніх клітинок. У асинхронних КА клітинки переходять у новий стан у випадковому порядку, причому новий стан клітинки відразу може використовуватися її сусідами як вхідний.

За можливістю руху клітин автомати поділяються на рухливі та нерухомі. Рухливі КА характеризуються можливістю зміни положення клітинки в решітці під час еволюції системи. У нерухомих КА положення клітини під час еволюції залишається постійним.

За ступенем залежності від випадкових факторів автомати поділяють на детерміновані та імовірнісні. У детермінованих КА стан комірки n+1 в наступний момент часу однозначно визначається станом цієї клітинки і її найближчих сусідів у попередній момент часу. У цьому випадку стан даного елемента в момент часу n +1 є однозначною функцією F від двох змінних — стану цього елемента і суми станів його найближчих сусідів у попередній момент часу n. При такому визначенні клітинний автомат не має пам'яті. КА з пам'яттю можна отримати, припустивши, що функція F залежить, наприклад, також від стану елемента в ще більш ранній момент часу.

КА, в яких стани комірок в наступний момент часу визначаються на основі деяких ймовірностей, називаються імовірнісними КА (ІКА). У класичних ІКА правила переходів мають абстрактний характер і не пов'язані однозначно з реальними процесами, що відбуваються в модельованій системі. У таких автоматах при моделюванні процесу для кожної клітинки датчиком випадкових чисел генерується випадкове число Q (0 < Q < 1), що порівнюється з імовірністю w реалізації цього процесу. Якщо Q < w, то процес реалізується.

Іноді використовуються правила, записані у вигляді звичайних диференціальних рівнянь (клас КА-ЗДР). У цьому випадку стани комірок задаються набором змінних, значеннями яких можуть бути будь-які дійсні числа. Для таких автоматів диференціальні рівняння розв'язуються для кожної комірки окремо протягом фіксованого відрізку часу, при цьому кожна клітинка може мати різні початкові умови. Цей клас КА дуже щільно прилягає до диференціальних рівнянь в частинних похідних.

Моделі типу КА-ЗДР займають проміжний стан між КА і ІКА, а також між простими КА і ДР в частинних похідних. Основною ідеєю КА-ЗДР є розбиття модельованої області на рівновеликі комірки і розв'язання системи ЗДР незалежно в кожній клітинці з різними початковими умовами. У деяких моделях просторове розташування комірок неістотне, а в інших кількість сусідніх комірок і розмірність простору відіграють вирішальну роль (випадки поширення хвиль або виникнення стаціонарних просторових структур у нерухомому середовищі). У моделях КА-ЗДР передбачається, що клітинка містить дуже велику кількість частинок, що дозволяє застосовувати ЗДР і неперервні функції. Ця обставина залишає тільки один спосіб для моделювання дифузії, а саме просте опосередкування концентрації по сусіднім коміркам.

За структурою КА поділяють в залежності від кількості вимірів. Найбільш вживані одно- та дво-вимірні.

Як ґратки беруть поле, комірки якого є трикутники, чотирикутники чи шестикутники.

В одновимірному (лінійному) КА решітка являє собою ланцюжок клітинок (одновимірний масив), в якій для кожної з них, крім крайніх, є по два сусіди. Для усунення крайових ефектів решітка «загортається» у тор. Це дозволяє використовувати наступне співвідношення для всіх клітин автомата:

y '[i] = f (y [i-1], y [i], y [i +1]),

де f – функція переходів клітинки;

y '[i] – стан i-ої клітинки в наступний момент часу;

y [i-1] – стан (i-1)-ої клітинки в даний момент часу;

y [i] – стан i-ої клітинки в даний момент часу;

y [i +1] – стан (i +1)-ої клітинки в даний момент часу.

У двовимірному (площинному) КА решітка реалізується двовимірним масивом. У ній кожна клітина має вісім сусідів. Для усунення крайових ефектів решітка так само, як і в попередньому випадку, «загортається» у тор. Це дозволяє використовувати наступне співвідношення для всіх клітинок автомата:

y '[i] [j] = f (y [i] [j], y [i-1] [j], y [i-1] [j +1], y [i] [j +1] , y [i +1] [j +1], y [i +1] [j], y [i +1] [j-1], y [i] [j-1], y [i-1] [j-1]).

Варіюючи різні параметри, можна отримати КА необхідної конфігурації. Гнучкість конфігурації та універсальність обчислень забезпечили високу популяризацію кліткових автоматів у різних сферах. Свобода у виборі параметрів конфігурації дуже зручна для використання, але це накладає додаткову складність у класифікації та систематизації знань теорії кліткових автоматів. Тим не менше, найбільш використовуване на практиці лише невелике сімейство конфігурацій кліткових автоматів. Як правило, кожен з них має свою назву. Наведено невеликий список найбільш використовуваних варіантів конфігурацій:

* Мозаїчний автомат. КА, що використовує у локальному правилі кожного елемента не тільки стан елемента та його сусідів, але і значення загального вхідного параметра, який може змінюватися час від часу. Зміна цього параметра веде до перевизначення набору правил зміни станів у всьому просторі елементів КА. Якщо з будь-якого початкового стану можна привести клітковий автомат в будь-яку задану конфігурацію шляхом варіювання значення загального вхідного параметра, такий КА називають повним.
* Ітеративний автомат. КА, в якому лише один елемент використовує для зміни свого стану значення вхідного параметра
* Односторонній клітковий автомат. Такий автомат припускає лише односторонню взаємодію елементів. Наприклад, в одновимірному масиві елементів значення кожного елемента залежить лише від його стану і від стану лівого (або правого) сусіда. Незважаючи на удавану вироджуваність звичайного КА, односторонні КА досить універсальні і використовуються для розпізнавання мовних форм.
* Л-система. Цей тип КА використовується для моделювання біологічних систем. Це динамічні КА (як правило, одно- чи двовимірні), в яких з часом один елемент може замінятися декількома або може бути видаленим із системи згідно з заданими правилами.
* Відмовостійка система. У таких системах моделюється робота КА в реальних умовах: з деякою ймовірністю кожен елемент КА може перейти в стан, що не відповідає локальному правилом. Завданням є створення алгоритмів, для яких робота КА буде правильною в незалежності від допущених помилок.

## Одновимірні клітинні автомати.

### Основні відомості та код Вольфрама.

Найпростішим нетривіальним клітинним автоматом є одновимірний клітинний автомат з двома можливими станами, а сусідами клітини будуть суміжні з нею клітини. Три клітини (центральна та її сусіди) породжують 23 = 8 комбінацій станів цих клітин. Далі на основі аналізу поточного стану трійки приймається рішення про те, чи буде центральна клітина порожньою чи повною на наступному кроці. Всього існує 28 = 256 можливих правил. Ці 256 правил кодуються відповідно коду Вольфрама - стандартною угодою, правилу, яке було запропоновано Вольфрамом. У деяких статтях ці 256 правил були досліджені та порівняні. Початкова умова для кожного автомату - одна центральна клітина - заповнена, решта - порожні. По осі Y відкладається дискретний час, а по осі X відкладаються стани клітин автомата.

Код Вольфрама являє собою систему присвоєння імен, що часто використовується для одновимірних правил клітинного автомата, введену Стівеном Вольфрамом в роботі 1983 року та використану ним в своїй книзі A New Kind of Science.

Код заснований на спостереженні, що таблиця із зазначенням нового стану кожного осередку в автоматі, в залежності від станів в його околиці, може бути інтерпретована як k-значне число в S-ковій позиційній системі числення, де S - це число станів, що кожна клітинка в автоматі може мати, k = S2n + 1 - число конфігурацій околиці, а n - радіус околиці. Таким чином, код Вольфраму для конкретного правила є числом в діапазоні від 0 до S^S2n + 1 - 1, перетворений з S-кової в десяткову систему числення. Вона може бути розрахована наступним чином:

* Виписати всі S2n + 1 можливих станів конфігурації околиці даної клітини.
* Інтерпретувати кожну конфігурацію у вигляді числа, як описано вище, впорядкувати їх в порядку убування номерів.
* Для кожної конфігурації вкажіть стан, який дана клітина матиме згідно з цим правилом, на наступній ітерації.
* Інтерпретувати отриманий список станів знову як S-ковий номер і перетворити це число в десятковій системі. Отримане в результаті десяткове число є кодом Вольфрама.

Код Вольфрама не визначає ні розмір (чи форму) околиці, ні число станів - вони передбачаються відомими з контексту. При використанні кодів самостійно без такого контексту, зазвичай передбачається, що вони відносяться до класу елементарних клітинних автоматів - одновимірних клітинних автоматів з двома станами клітин та з (суміжною) трьохклітинною околицею, які Вольфрам широко досліджує в своїй книзі. Відомі правила в цьому класі включають правило 30, правило 110 і правило 184. Правило 90 також цікаве тим, що воно створює трикутник Паскаля по модулю 2. Код такого типу суфіксом R, наприклад, як "Правило 37R", вказує на клітинний автомат другого порядку з такою ж структурою околиці.

У той час як в строгому сенсі цього слова кожен код Вольфрама в допустимих межах визначає різні правила, деякі з цих правил є ізоморфними і, як правило, вважаються еквівалентними. Наприклад, правило 110 ізоморфне правилам 124, 137 і 193, які можуть бути отримані з оригіналу відображенням вліво-вправо та іншою нумерацією станів. За угодою, кожен такий клас ізоморфізму представлений правилом з найменшим номером коду в ньому.

Недоліком коду Вольфрама та використання десяткової системи числення, зокрема, є те, що він ускладнює пошук таких ізоморфізмів порівняно з деякими альтернативними нотаціями. Незважаючи на це, він став стандартним де-факто способом позначення одновимірних клітинних автоматів.

### Правило 30.

Правило 30 - одномірне двійкове правило для клітинного автомата, вперше описане Стівеном Вольфрамом в 1983 році. З чотирьох типів поведінки, описаних в його книзі "A New Kind of Science", Правило 30 володіє класом поведінки III, показуючи аперіодичну, хаотичну поведінку.

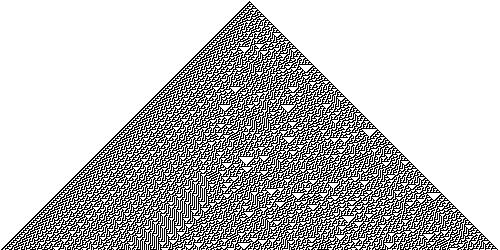
Правило являє інтерес, тому що воно породжує складні, багато в чому випадкові структури з простих, чітко визначених правил. Вольфрам вважає, що клітинні автомати в цілому і Правило 30 зокрема - ключ до розуміння того, як прості правила можуть породжувати складні структури і різну складну поведінку різних природних об'єктів. Наприклад, структуру, схожу на ту, що породжуються Правилом 30, можна знайти на раковині широко розповсюдженого тропічного молюска Conus textile. Також це правило було запропоновано для використання як шифратор послідовностей в криптографії.

Якщо записати код правила в двійковому вигляді, то дзеркальне відображення коду правила, інверсія бітів коду і дзеркальне відображення з інверсією бітів мають в десятковій системі числення коди 120, 225 і 135 відповідно. Таким чином, правила під цими номерами є ізоморфними Правилу 30.

Правило 30 в бінарному вигляді можна записати наступним чином (тут три цифри зверху — це стан лівого сусіда, поточної клітини та правого сусіда відповідно):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Поточний стан | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| Новий стан центральної клітини | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

На наступному зображенні відображено приклад еволюції автомату з одної початкової заповненої клітини (по горизонталі розміщені всі клітини одного покоління, по вертикалі — стан однієї клітини на кожному кроці):



Як можна бачити з малюнка, Правило 30 породжує послідовність, що здається в багатьох відносинах випадковою, при початкових умовах, які не можна назвати випадковими. Стівен Вольфрам запропонував використання вертикальних стовпчиків в еволюції клітинних автоматів при деякій заданій початковому стані як псевдовипадкову послідовність. Послідовності, отримані таким способом, задовольняють багатьом стандартним тестам на випадковість. Стівен Вольфрам використовує Правило 30 для генерації випадкових цілих чисел в пакеті Mathematica. Однак M. Sipper і M. Tomassini показали, що як ГПСЧ Правило 30 погано проходить тест на критерій згоди Пірсона (критерій χ²) в порівнянні з іншими псевдовипадковими послідовностями, які були отримані за допомогою інших клітинних автоматів.

Вольфрам припустив хаотичність еволюції за Правилом 30, грунтуючись, в основному, на її зовнішньому графічному вигляді. Однак пізніше було показано, що застосування Правила 30 задовольняє більш чітким визначенням хаосу, сформульованим Devaney і Knudson. Відповідно до критерію Devaney, Правило 30 демонструє ефект метелика - якщо задати 2 початкових стану, що розрізняються, наприклад, тільки одним бітом, то віддалені багатьма поколіннями нащадки цих 2 станів будуть абсолютно різні, періодичні зміни є щільними в просторі будь-яких змін топології Кантора. Також, Правило 30 має властивість перемішування. Відповідно до критерію Knudson, це показує чутливість до початкових умов і щільність орбіт процесу (будь-яка конфігурація в підсумку призводить до виникнення всіх можливих кінцевих конфігурацій клітин). Обидві ці характеристики хаотичної поведінки еволюції за Правилом 30 слідують з властивості Правила 30, яке легко перевірити: якщо дві конфігурації C і B розрізняються на одну клітку в позиції i, тоді після одного кроку нові конфігурації будуть відрізнятися в клітці i + 1.

### Правило 110.

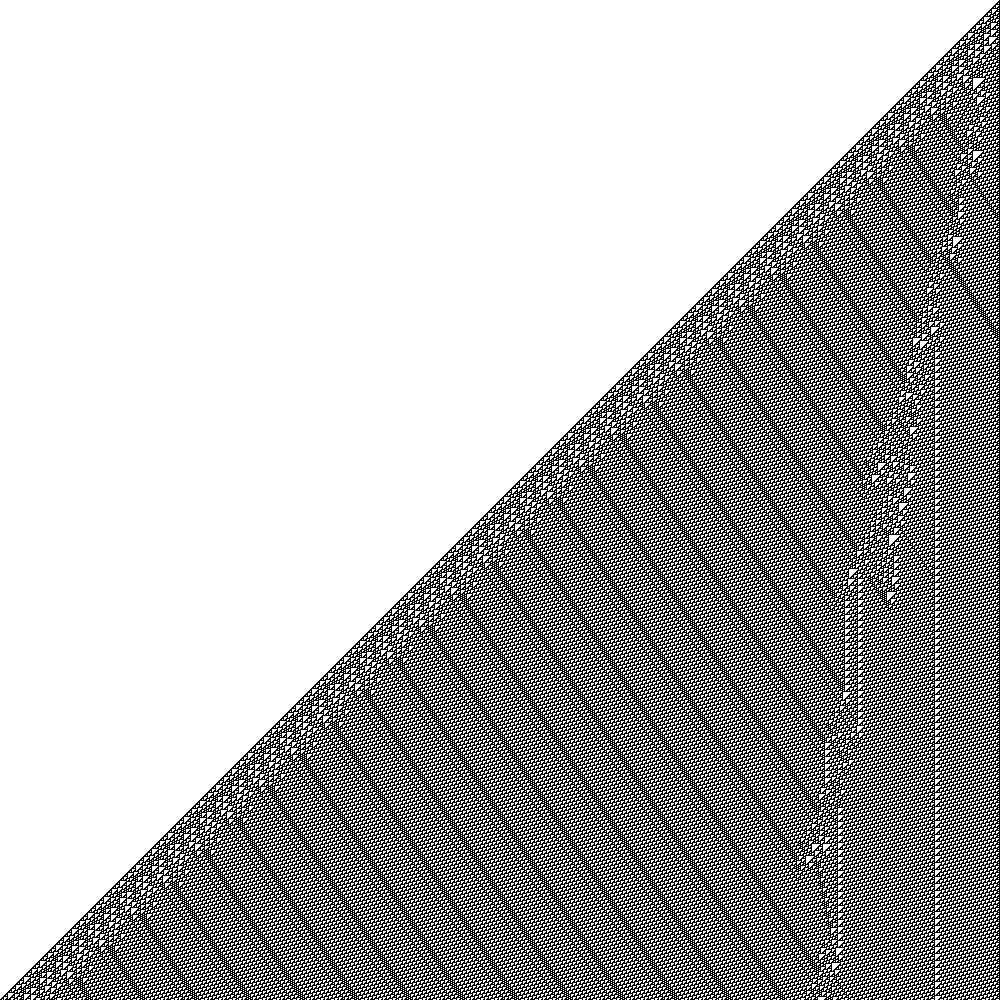
Правило 110 - елементарний одновимірний клітинний автомат з поведінкою, що знаходиться на межі хаосу і стабільності. В цьому відношенні Правило 110 ідентично грі «Життя». Відомо, що Правило 110 є Тьюринг-повним, що означає, що будь-яка обчислювальна процедура може бути реалізована за допомогою цього клітинного автомата.

Меттью Кук представив свій доказ цього факту на конференції Інституту Санта-Фе в 1998 році, але Вольфрам заборонив включати цей доказ в паперову версію матеріалів конференції, тому що не хотів, щоб воно було опубліковано до видання книги A New Kind of Science. У 2004 році доказ Кука було опубліковано в журналі Вольфрама «Complex Systems» (випуск 15, том 1), через 10 років після того як Кук вперше представив його.

Правило 110 в бінарному вигляді можна записати наступним чином (як і в минулому випадку, три цифри зверху — це стан лівого сусіда, поточної клітини та правого сусіда відповідно):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Поточний стан | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| Новий стан центральної клітини | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

Наступним чином виглядає еволюція автомату з одної початкової заповненої клітини (по горизонталі розміщені всі клітини одного покоління, по вертикалі — стан однієї клітини на кожному кроці):



На малюнку можна побачити, що нові клітини заповнюються лише ліворуч від початкової клітини. Це пояснюється тим, що згідно цьому правилу пуста клітина без правого сусіда ніколи не заповниться, а через те що клітини справа від початкової завжди пусті та завжди не мають правого сусіда, вони завжди залишатимуться пустими.

## Мураха Ленгтона.

### Класичний варіант мурахи Ленгтона.

Мураха Ленгтона - це двовимірний клітинний автомат з дуже простими правилами, винайдений Крісом Ленгтоном. Мураху можна також вважати двовимірною машиною Тьюринга з 2 символами і 4 станами.

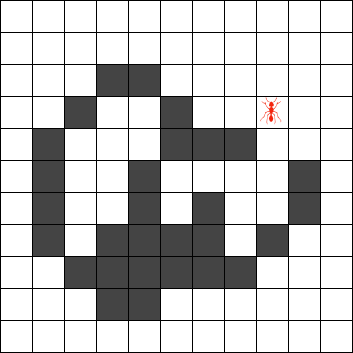
Розглянемо нескінченну площину, розбиту на клітини, пофарбовані деяким чином в чорний і білий колір. Нехай в одній з клітин знаходиться «мураха», яка на кожному кроці може рухатися в одному з чотирьох напрямків в клітку, сусідню за стороною. Мураха рухається згідно з такими правилами:

* На чорному квадраті - повернути на 90 ° вліво, змінити колір квадрата на білий, зробити крок вперед на наступну клітину.
* На білому квадраті - повернути на 90 ° вправо, змінити колір квадрата на чорний, зробити крок вперед на наступну клітину.

Ці прості правила викликають досить складну поведінку: після деякого періоду досить випадкового руху мураха, мабуть, починає неодмінно будувати дорогу з 104 кроків, повторювану нескінченно, незалежно від початкової розмальовки поля. Це наводить на думку, що «магістральна» поведінка є стабільним аттрактором мурашки Ленгтон. Можливо, що «магістраль» є єдиним аттрактором при переміщенні мурашки.

Мураха Ленгтона також може бути описаний як клітинний автомат, в якому майже все поле пофарбовано в чорно-білий колір, а клітина з «мурахою» має один з восьми різних кольорів, що кодують відповідно всі можливі комбінації чорного / білого кольору клітини і напрямки руху мурашки.

На зображенні нижче показано результат роботи клітинного автомату після декількох десятків ітерацій. В процесі роботи “мураха” здійснює рух по нетривіальній траєкторії, яка, як і стан клітин автомату, важко передбачається.



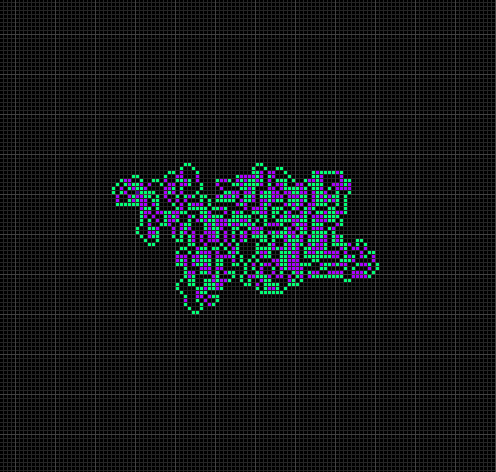
“Магістраль” - послідовність руху, до якої мураха переходить після певної кількості ітерацій — виглядає наступним чином (в нижній частині малюнку, тут червона клітина — поточне положення мурахи):



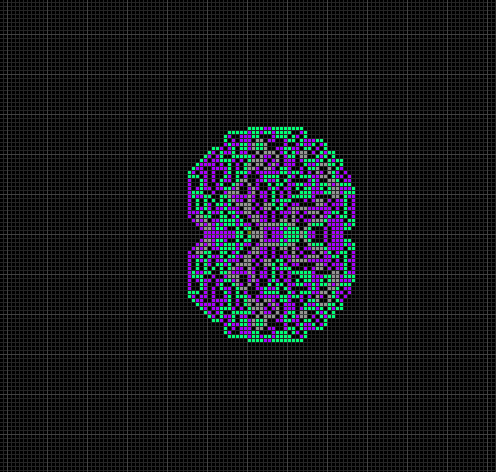
### Розширення ідеї мурахи Ленгтона.

Існує просте розширення мурахи Ленгтона, в якому використовується більше двох кольорів клітин. Кольори змінюються циклічно. Для таких мурах існує також проста форма назви: для кожного наступного кольору використовується буква L або R (Л і П), в залежності від того, чи повертає мураха направо або наліво. Таким чином, мураха Ленгтона - це мураха RL.

Наведемо декілька прикладів. Так виглядає результат роботи автомату з мурахою RLR, для якого характерна хаотична поведінка та досі не доведено, що мураха в цьому випадку створює “магістраль”:

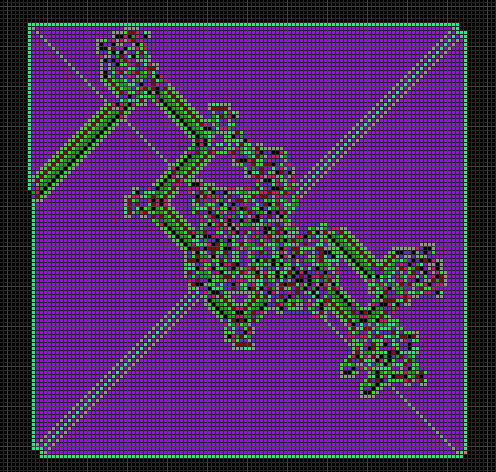


Деякі з цих узагальнених мурах Ленгтона малюють візерунки, які стають все більш симетричними. Один з простих прикладів - мураха RLLR. Одна достатня умова цього полягає в тому, що ім'я мурашки, що розглядається як циклічний список, складається з послідовних пар повторюваних букв LL або RR (циклічність списку означає, що остання буква може спаровуватися з першою). Так виглядає автомат мурахи LLRR:

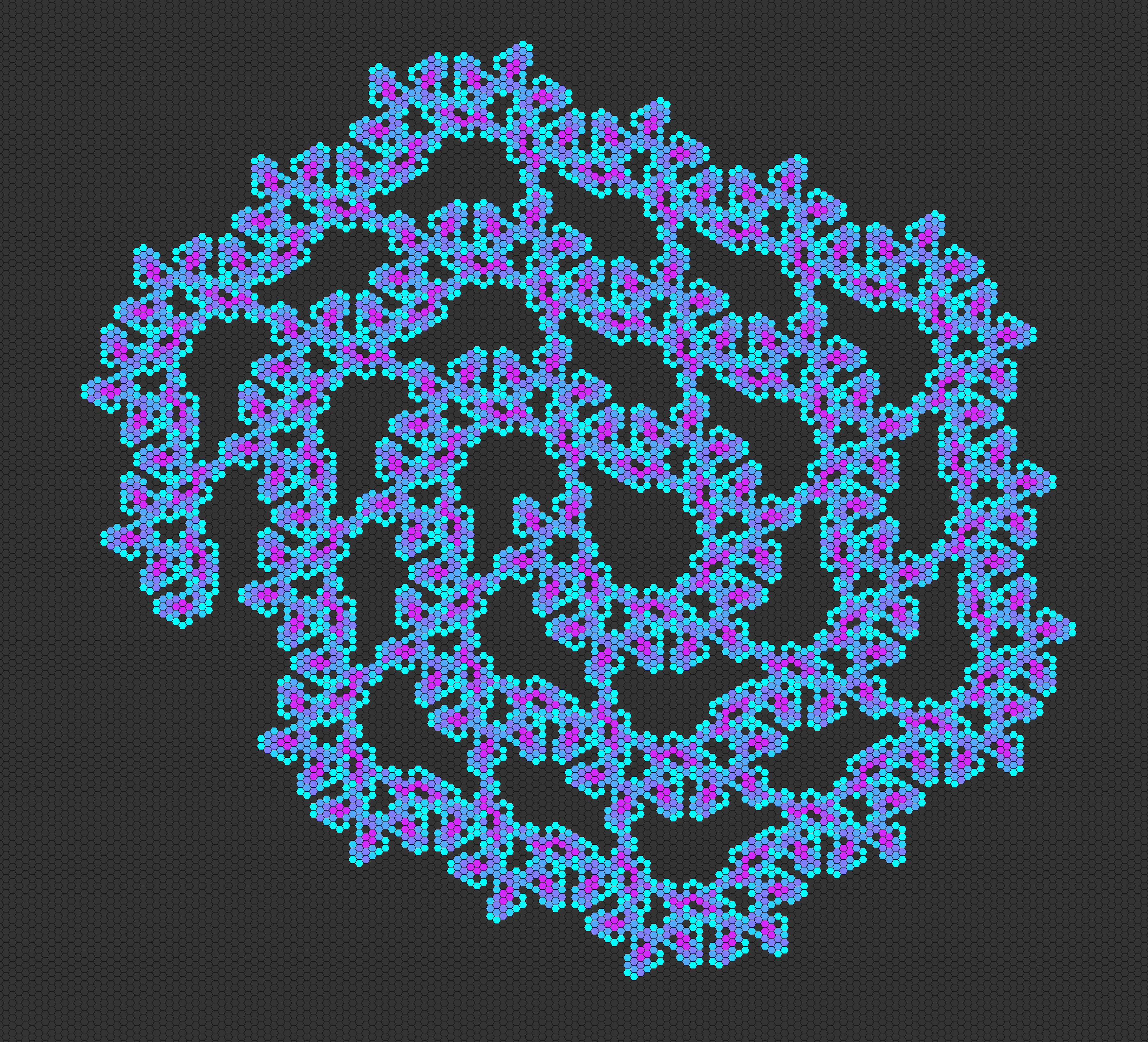


На зображенні можна чітко бачити симетричність структури. В даному випадку окрім чорного кольору також використані зелений та фіолетовий.

Деякі конфігурації мурахи здатні заповнювати певну площу одним кольором, збільшуючи цю площу з часом. Типовим прикладом цієї поведінки є мураха LRRRRRLLR — вона заповнює квадрат навколо себе одним кольором (в даному прикладі — фіолетовим):



Також іноді використовується гексагональна структура автомату. Для неї можливо більше ніж два повороти, тому процес їх задання складніший. Для подібних структур часто характерне поступове розширення повторюваного візерунку, як в наступному прикладі:



Таким чином, використовуючи різні конфігурації поля та мурахи можна отримати результат, котрий в багатьох випадках виказує ознаки клітинного автомату четвертого класу, з можливими повторюваними структурами та ознаками хаосу.

## Гра “Життя”.

### Основні правила гри “Життя”.

Гра «Життя» (англійською Conway's Game of Life) - клітинний автомат, винайдений англійським математиком Джоном Конвеєм в 1970 році.

Джон Конвей зацікавився проблемою, запропонованою в 1940-х роках відомим математиком Джоном фон Нейманом, який намагався створити гіпотетичну машину, яка може відтворювати сама себе. Джону фон Нейману вдалося створити математичну модель такої машини з дуже складними правилами. Конвей спробував спростити ідеї, запропоновані Нейманом, і врешті-решт йому вдалося створити правила, які стали правилами гри «Життя».  
Вперше опис цієї гри було опубліковано в жовтневому (1970 рік) випуску журналу Scientific American, в рубриці «Математичні ігри» Мартіна Гарднера (Martin Gardner).

Гра “Життя” є двовимірним клітинним автоматом та, як і багато інших розглянутих автоматів, має квадратну сітку та два можливих стани клітини: “жива” та “мертва”. Існують два основних правила, що керують роботою цього клітинного автомату: умова загибелі та умова народження.

* Якщо пуста (“мертва”) клітина має рівно 3 сусіда, на наступному кроці вона заповнюється (переходить в “живий” стан);
* Якщо заповнена (“жива”) клітина має менше 2 чи більше 3 сусідів, вона “вмирає” (стає пустою) від “самотності” чи “перенаселення” на наступному кроці.

Обидві ці умови зав’язані на одному критерії — кількості сусідів. Вона визначається як кількість живих клітин серед восьми сусідніх (зверху, знизу, ліворуч, праворуч та в кутах). При цьому стан клітини на наступному кроці визначається станом сусідей та поточної клітини на даному кроці. Таким чином цей автомат можна віднести до синхронних.

Гра “Життя” відноситься до клітинних автоматів 4 класу, містячи складні постійні структури та хаотичне розповсюдження клітин навіть при незначних змінах. Було доведено, що цей клітинний автомат є Тьюринг-повним, тобто за допомогою нього можна реалізувати розрахунок майже будь-якої математичної операції. Його дослідженням займаються вже декілька поколінь вчених, знаходячи все складніші стійкі структури, що мають певні цікаві особливості.

Певна ділянка поля, в якій заповнені конкретні клітини, називається конфігурацією клітинного автомату. Далі буде розглянуто різновиди цих конфігурацій та стійких структур, а також назви, які їм були дані в процесі дослідження.

### Натюрморти.

Натюрморти - конфігурації «Життя» або іншого клітинного автомата, які не змінюються в процесі еволюції. Іншими словами, натюрморт є осциллятором періоду 1 (осциллятори будуть розглянуті пізніше).

Існує кілька близьких за змістом термінів, що позначають конфігурації, що не змінюються в процесі еволюції (конфігурації, які є власними батьками). Відмінності між ними пов'язані з відповіддю на такі питання:

1. Чи вважається натюрмортом конфігурація, що складається з двох незалежних натюрмтортов (наприклад, двох блоків на досить великій відстані один від одного)?
2. Чи вважається натюрмортом конфігурація, що складається з двох частин, кожну з яких можна видалити так, що друга частина залишиться батьком себе?

В існуючих словниках і онлайн-енциклопедіях наводяться наступні визначення:

* Стійкий зразок (англ. Stable pattern) - об'єкт, який є власним батьком;
* Натюрморт (англ. Still life, strict still life) - стійкий об'єкт, який є кінцевим і непустим, який не може бути розділений на дві стійкі частини;
* Псевдонатюрморт (англ. Pseudo still life) - стійкий об'єкт, який не є натюрмортом, в якому присутня хоча б одна мертва клітина, що має більше трьох сусідів загалом, але менше трьох сусідів у кожному зі складових об'єкт натюрмортів.

Точне визначення «стійкості» представляє інтерес в контексті перерахування натюрмортів: наприклад, згідно з наведеними визначеннями, кількість стійких конфігурацій розміру 8 (тобто складаються з 8 живих клітин) в «Житті» нескінченно, тому що пара блоків на будь-якій відстані один від одного є стійкою; проте, кількість натюрмортів обмеженого розміру вважається кінцевою. Наведемо два приклади:

Псевдонатюрморт в «Життя». Видалення одного з островів не впливає на стабільність другого острова.

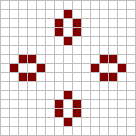
«Строгий» натюрморт. Стабільність кожного з островів залежить від наявності іншого острова.

Дослідниками встановлено число натюрмортів і псевдонатюрмортів розміру не вище 24 клітин. Завдання визначення типу стійкої конфігурації (натюрморт, псевдонатюрморт) вирішується за поліноміальний час шляхом пошуку циклів в пов'язаному кососиметричному графі. Далі розглянемо конкретні натюрморти.

Найбільш поширений натюрморт - блок - конфігурація у формі квадрата 2 × 2. Два блоки, розміщені в прямокутнику 2 × 5, утворюють бі-блок - найпростіший псевдонатюрморт. Блоки використовуються в якості складових частин у безлічі складних пристроїв, наприклад, в планерній рушниці Госпера. На малюнку ліворуч зображено блок, праворуч — бі-блок:



Другий за поширеністю натюрморт - вулик. Вулики часто виникають четвірками в конфігурації, що називають пасікою. Вулик та пасіку зображено на малюнках ліворуч та праворуч відповідно:

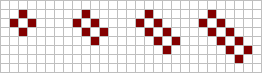


Третій за поширеністю натюрморт - коровай. Короваї нерідко з'являються парами. У свою чергу, подвійні короваї також з'являються в парах, званих пекарнями. Нижче зображено коровай, подвійний коровай та пекарню відповідно:

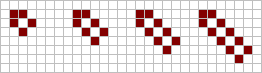


Ящик складається з чотирьох живих клітин в околиці фон Неймана центральної мертвої клітини. Додавання однієї живої клітини по діагоналі до центральної клітини перетворює ящик в човен, а додавання симетрично ще однієї клітини - в корабель. Природне подовження цих трьох конфігурацій дає баржу, довгий човен і довгий корабель відповідно. Подовження можна продовжувати як завгодно довго.

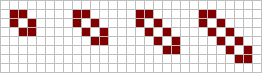
На наступному зображені показані (починаючи з лівої сторони) ящик, баржа та її подовження:



На цьому зображенні можна побачити човен, довгий човен та їх подовження:



Далі зображено корабель, довгий корабель та інші його подовження:



Приклади деяких іншіх натюрмортів:



Існують ї особливі різновиди натюрмортів, такі як пожирач та відбивач. Вони будуть розглянуті пізніше.

### Осцилятори.

Осцилятор - конфігурація клітинного автомата, яка після кінцевого числа поколінь повторюється в тій же самій орієнтації і позиції. Іншими словами, осцилятор - це будь-який зразок, який є попередником самого себе. Еволюція осцилятора триває як завгодно довго.

Залежно від контексту, космічні кораблі також можуть вважатися осцилляторами, але зазвичай вони розглядаються окремо.  
Мінімальна кількість поколінь, через яке осцилятор повертається у вихідну конфігурацію, називається періодом осцилятора. Осцилятор з періодом 1 зазвичай називається стійкою конфігурацією, так як він не змінюється. Таким чином, натюрморти є підмножиною осциляторів.

В «Житті» кінцеві осцилятори відомі для всіх періодів, крім 19, 23, 38 і 41. Хоча існують осцилятори періоду 34, всі відомі приклади вважаються тривіальними, оскільки вони складаються з по суті окремих компонент, осцилюючих з меншими періодами. Наприклад, осцилятор з періодом 34 можна отримати шляхом розміщення у всесвіті двох незалежних осциляторів з періодами 2 і 17. Осцилятор вважається нетривіальним, якщо він містить хоча б одну клітку, період осциляції якої дорівнює періоду осцилятора.

Наведемо приклади осциляторів. Найпоширенішим осцилятором є “блимавка”, та вона має період 2:



Серед інших осциляторів можна виділити крест – осцилятор з періодом 3:



Прикладом осцилятору с більшим періодом може слугувати, зокрема, фумарола. Її період складає 5 кроків:



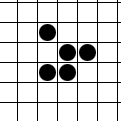
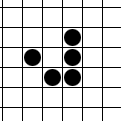
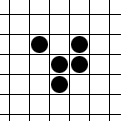
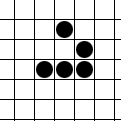
Знайдено й велику кількість інших осциляторів (наприклад, галактика Кока з періодом 8 чи пентадекатлон з періодом 15). Зокрема, для створення нового осцилятору достатньо об’єднати два вже відомих осциляторів в одну конфігурацію. Таким чином, можна зробити висновок, що на нескінченному полі кількість можливих осциляторів також нескінченна.

Осцилятори є прикладом більш складної стабільної структури, яка зазвичай відповідає 2 класу клітинних автоматів, проте здатна існувати у клітинному автоматі, що проявляє ознаки хаотичності.

### Космічні кораблі.

Конфігурація «Життя» або іншого клітинного автомату називається космічним кораблем, якщо через певну кількість поколінь вона знову з'являється без доповнень або втрат, але зі зміщенням відносно вихідного положення. Найменше таке число поколінь - період космічного корабля.

Першим виявленим космічним кораблем став планер. Планер був знайдений під час відстеження еволюції R-пентаміно (стартової комбінації з п’яти сусідніх заповнених клітин, що розпочинає найдовшу еволюцію серед усіх можливих таких комбінацій) в 1970 році Річардом Гаєм. Еволюцію планеру можна побачити на зображеннях нижче:



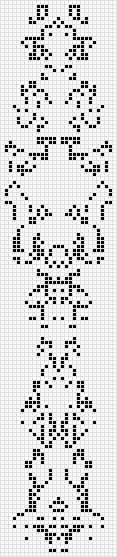
Космічний корабель періоду p, Який переміщується на (m, n) протягом його періоду, де m ≥ n, має тип (m, n) / p. Як було доведено Конуеєм в 1970 році, p ≥ 2m + 2n.

Космічні кораблі, що рухаються по горизонталі або вертикалі, називаються ортогональними кораблями. Якщо рух космічного корабля відбувається по діагоналі під кутом 45 °, такий корабель називається діагональними. Космічні кораблі, що рухаються під іншими кутами, називаються косими або наклонними. У 2010 році був сконструйований перший наклонний космічний корабель типу (5120, 1024) / 33699586.

Швидкістю світла в заданому клітинному автоматі називають найбільшу швидкість поширення інформації. Швидкість світла в «Житті» дорівнює швидкості переміщення шахового короля - швидкості в одну клітку за покоління по горизонталі, вертикалі або діагоналі. Зазвичай швидкість світла позначається літерою c.

Швидкість космічного корабля визначається відношенням відстані зміщення до періоду. Часто швидкість виражається через c. Так, швидкість планера в «Життя» дорівнює c / 4, так як він переміщується на одну клітину по діагоналі за чотири покоління. Найпростіший ортогональний космічний корабель, ВКК, рухається зі швидкістю c / 2.

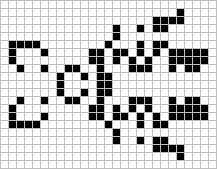
У загальному випадку, якщо космічний корабель в двовимірному клітинному автоматі на квадратній сітці переміщується на вектор (x, y) через n поколінь, його швидкість дорівнює v = с \* max (| x |, | y |) / n.

Найбільш поширеними різновидами космчних кораблів, окрім планерів, є легкі, середні та важкі космічні кораблі, зображені нижче (відповідно починаючи з лівого краю):



Крім цього, існує велика кількість більших космічних кораблів, як ортогональних, так і діагональних. Наприклад, ортогональний космічний корабель, виявлений Тімом Коу 11 червня 2016 року, що називається “макаронний монстр” - це перший космічний корабель швидкості 3с / 7. Він зображений на малюнку праворуч.

Існують певні структури, що тісно пов’язані з космічними кораблями. Тагалонг - конфігурація, яка не є сама по собі космічним кораблем, але може бути приєднана до космічного корабля, щоб сформувати новий космічний корабель. Іншими словами, тагалонг - це частина космічного корабля, яку можна видалити без руйнування корабля. Тагалонг, розташований перед космічним кораблем, називають пушалонгом. Нижче приведено приклад тагалонгу для двох легких космічних кораблів:



Конфігурація, що здатна змінювати напрямок руху космічного корабля, не руйнуючись при цьому, називається рефлектором. Конфігурацію, що здатна знищити космічний корабель і відновитися після реакції, називають пожирачем.

Рушниця - нерухома конфігурація, періодично випускає космічний корабель. Найвідоміша та перша знайдена рушниця — планерна рушниця Госпера:



Космічні кораблі можуть слугувати засобом передачі інформації. Здатність планерів передавати інформацію була основою для твердження, що гра “Життя” є Т’юринг-повною. Деякі космічні кораблі здатні на більше, ніж просто рухатись у просторі, зберігаючи свою структуру: вони здатні нескінченно створювати натюрморти чи інші кораблі в процесі руху, або навіть створювати рушниці, таким чином швидкість їх зростання прискорюється з часом. Такі складні структури і стали причиною, чому гра “Життя” досі цікавить математиків по всьому світу, а нові конфігурації з унікальними характеристиками знаходяться навіть після десятків років з винаходу цього клітинного автомату.

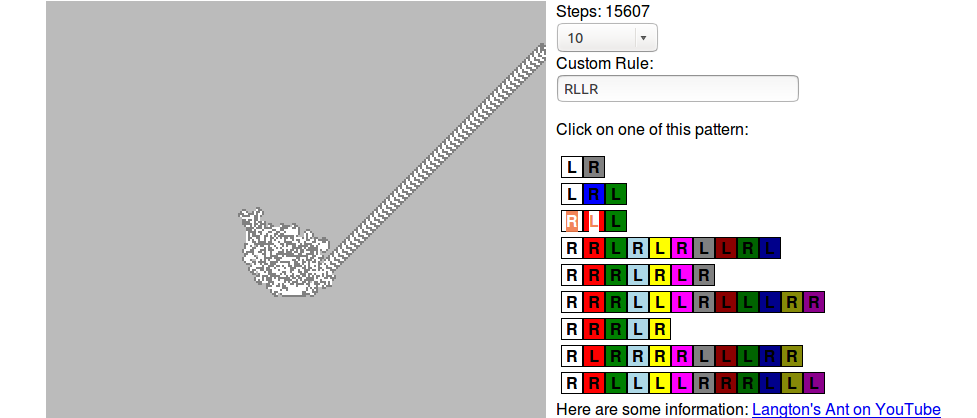
Існує й велика кількість варіацій гри “Життя”, в якій змінюється кількість сусідів, що необхідна для народження клітини, або її загибелі від самотності чи перенаселення. Також може змінюватись структура самого поля або його розмірність, додаватись випадкові елементи тощо.

## Програмні реалізації клітинних автоматів.

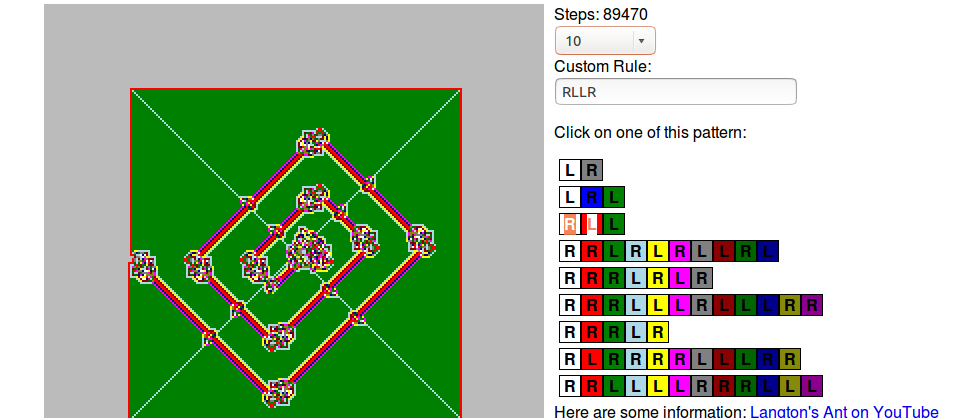
### Реалізації двовимірних клітинних автоматів.

Існує велика кількість програмних реалізацій різних клітинних автоматів. Найбільш поширені реалізації таких автоматів, як мураха Ленгтона та гра “Життя”. Особливо поширені реалізації цих автоматів у вигляді веб-сторінки на мові JavaScript. Зокрема, прикладом реалізації мурахи Ленгтона може слугувати реалізація Фабріса Вайнберга. Вона становить з себе веб-сторінку, на якій розміщено вихідний код програми та власне її виконання у браузері. Програма дозволяє як запустити класичний варіант мурахи, так і розширені варіанти з більшою кількістю станів. При цьому користувачеві надається можливість або обрати один з заздалегідь вказаних варіантів, що мають певні цікаві особливості, або задати правило власноруч, дозволяючи таким чином реалізувати будь-який варіант мурахи Ленгтона на квадратному полі. Реалізація Вайнберга дозволяє також обрати швидкість, з якою мураха рухається автоматом, розглядаючи більш незначні кроки чи загальну картину. Нижче розташовано декілька скріншотів даної програми.

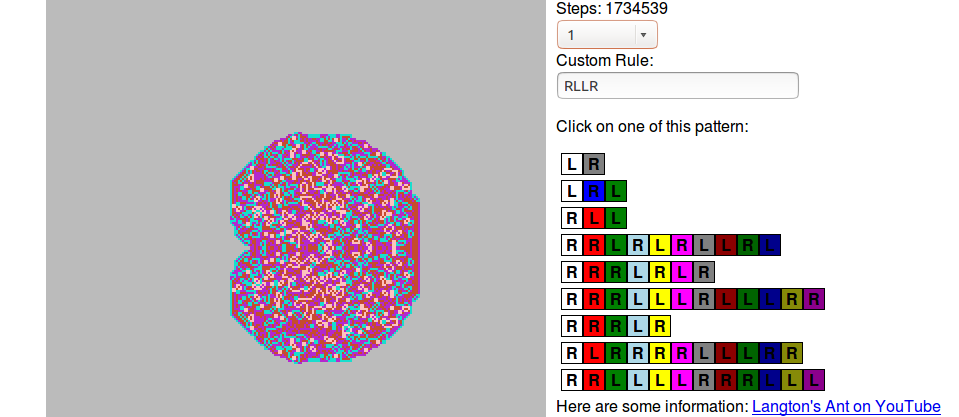
Приклад виконання програми із стандартними правилами:



Приклад виконання програми з використанням одного з заданих наборів розширених правил (RLRRRRLLLRR):

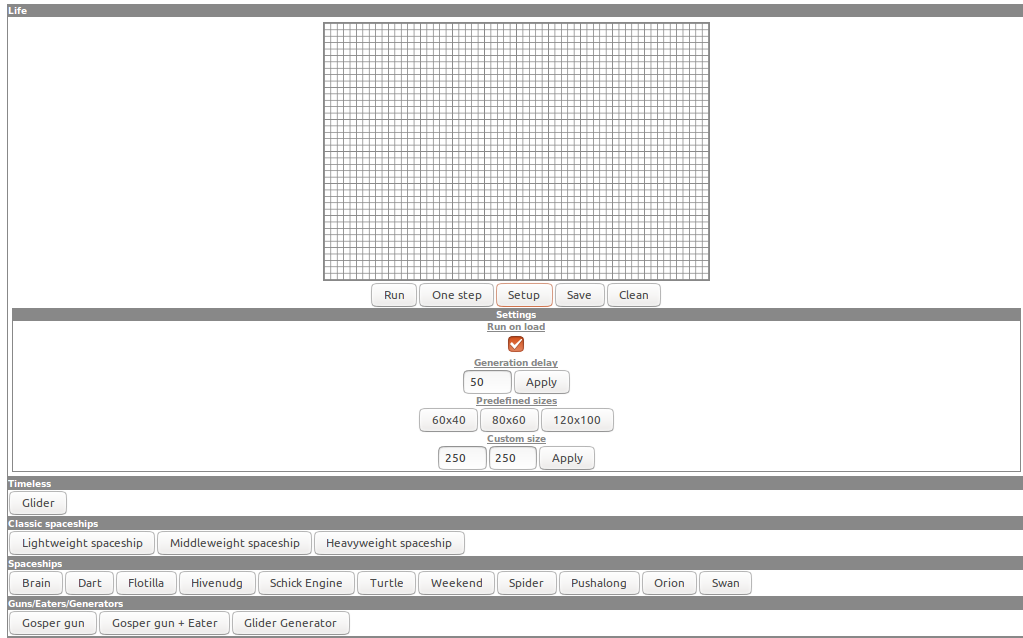


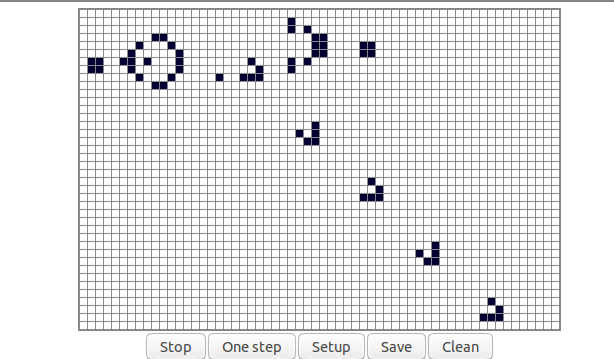
Приклад виконання програми із власним набором розширених правил (RLLR, з симетричним візерунком):

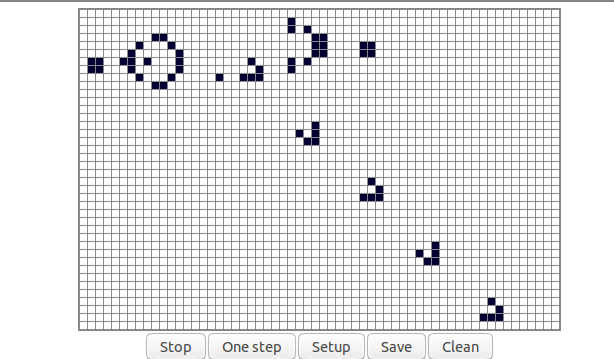


У випадку ж гри “Життя” різноманітних реалізацій ще більше. Гру “Життя” програмно реалізовували ще на старих комп’ютерах 70-х років, проте зараз можливо створити реалізацію зі зручним інтерфейсом та додатковими можливостями. Одним з прикладів таких програм може слугувати реалізація на веб-сайті michurin.ru, що становить з себе JavaScript варіант автомату з можливістю налаштування розміру поля, швидкості переходу до нового стану, покроковому огляду еволюції, а також можливістю обрати одну з заздалегідь створених конфігурацій, серед яких велика кількість натюрмортів, осциляторів та космічних кораблів, а також рушниця Госпера. Крім того, як і багато інших реалізацій автомату, він дозволяє змінювати стан клітин під час виконання, вносячи корективи в роботу автомату чи, наприклад, продовжуючи еволюцію, що вже стала тривіальною. Дана реалізація використовує інтерфейс російською мовою.

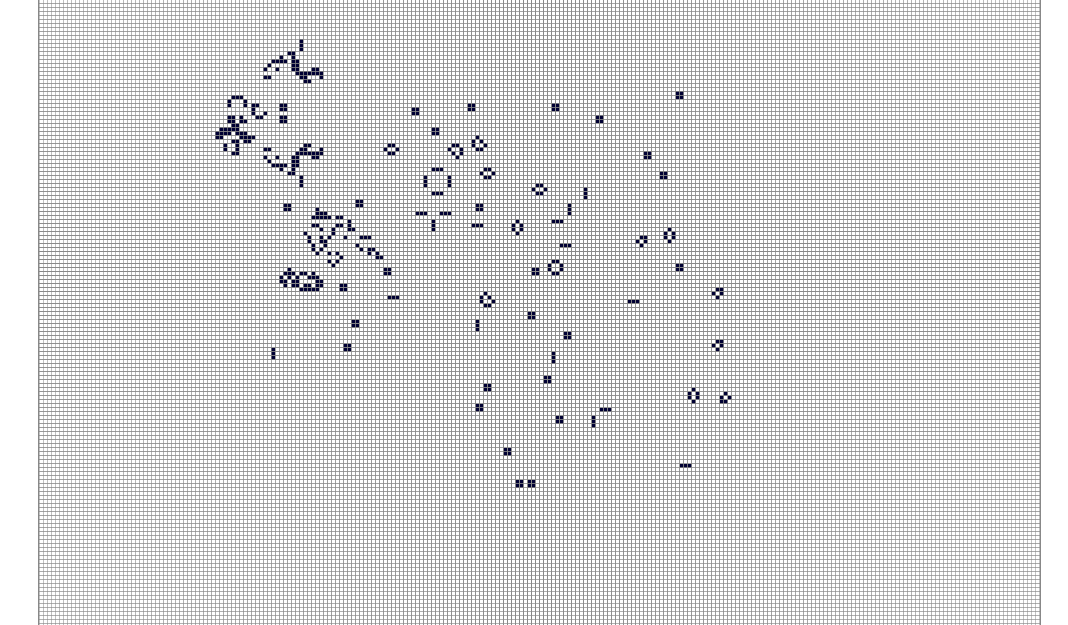
Зовнішній вигляд інтерфейсу програми:



Результат виконання одної з доступних конфігурацій — планерної рушниці Госпера:

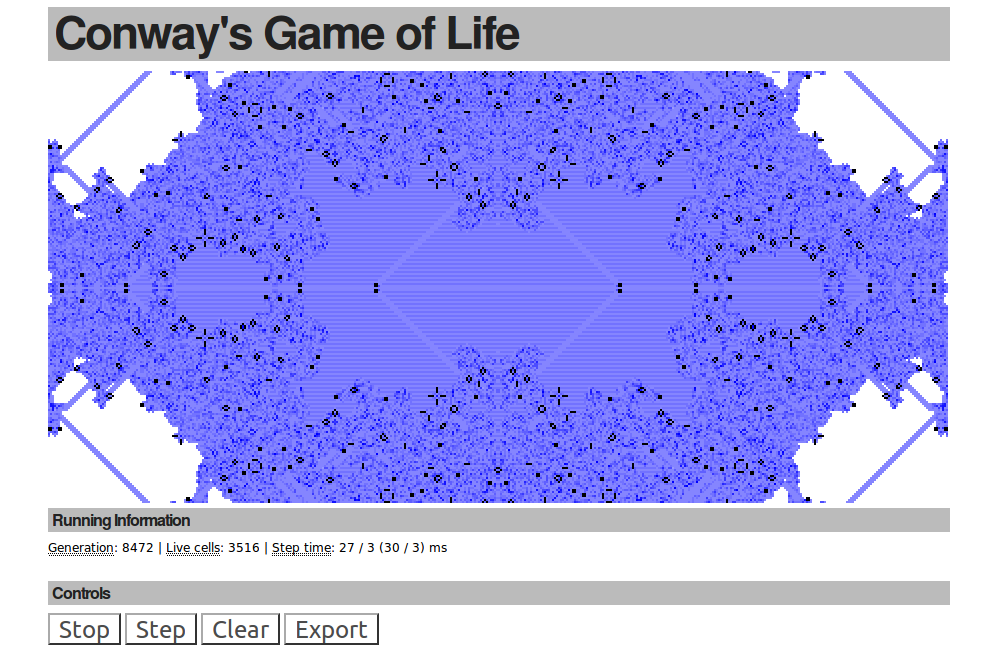


Результат виконання застосунку з випадковими початковими умовами:



Ще одним яскравим прикладом реалізації гри “Життя” може слугувати веб-сторінка на сайті pmav.eu. Ця реалізація не має певних можливостей, що присутні у попередній, проте має свої переваги. До недоліків реалізації варто віднести відсутність можливості налаштовувати розмір поля та швидкість переходу до нового стану (хоча ці дані й зазначаються на самій сторінці). Також варто відзначити значно менший набір конфігурацій, які можна швидко задати за допомогою інтерфейсу. Однак дана програма все ж має свої переваги; зокрема, в ній є можливість змінювати кольорову палітру відображення, зовнішній вигляд сітки, що розмежовує клітини, а також дозволяє відображати клітини, що раніше вже були живими, але на даний момент мертві, особливим кольором. Таким чином, цей застосунок більш підходить для розгляду розповсюдження конфігурації в процесі своєї еволюції, надаючи можливість отримати більш широке розуміння цього процесу.

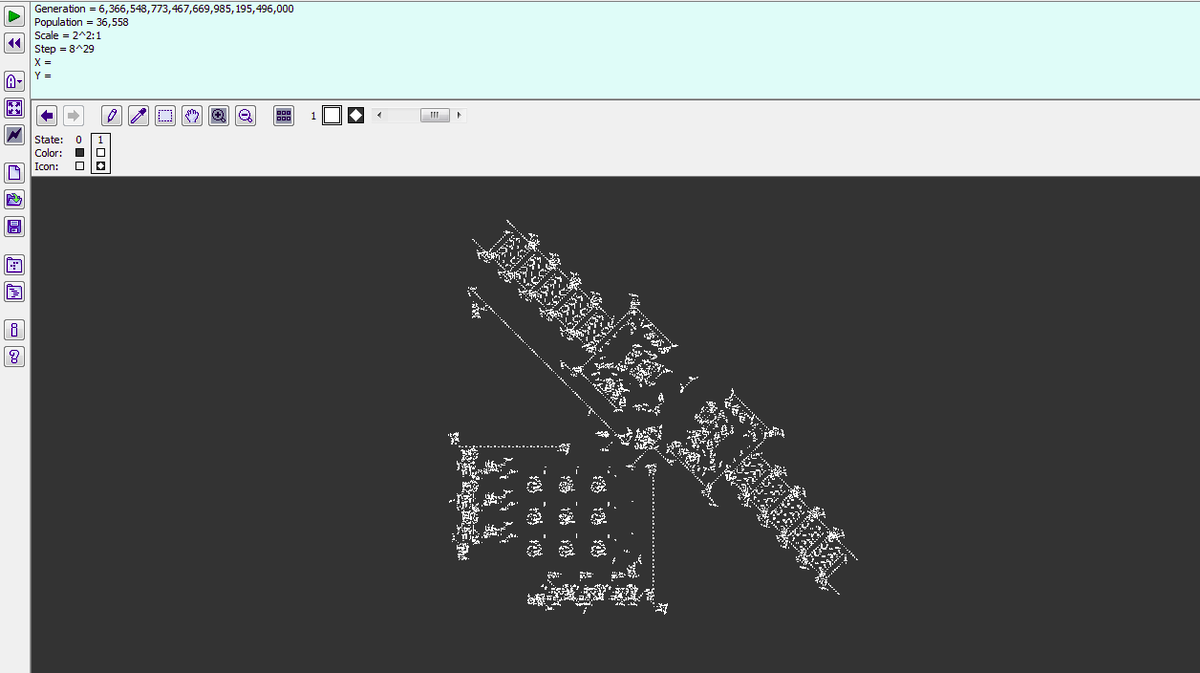
Ось таким чином виглядає кінцевий результат еволюції одного з шаблонів, наданих в даній реалізації. Блакитним та синім кольорами позначені клітини, що зараз є пустими, проте були заповнені в попередніх поколіннях:



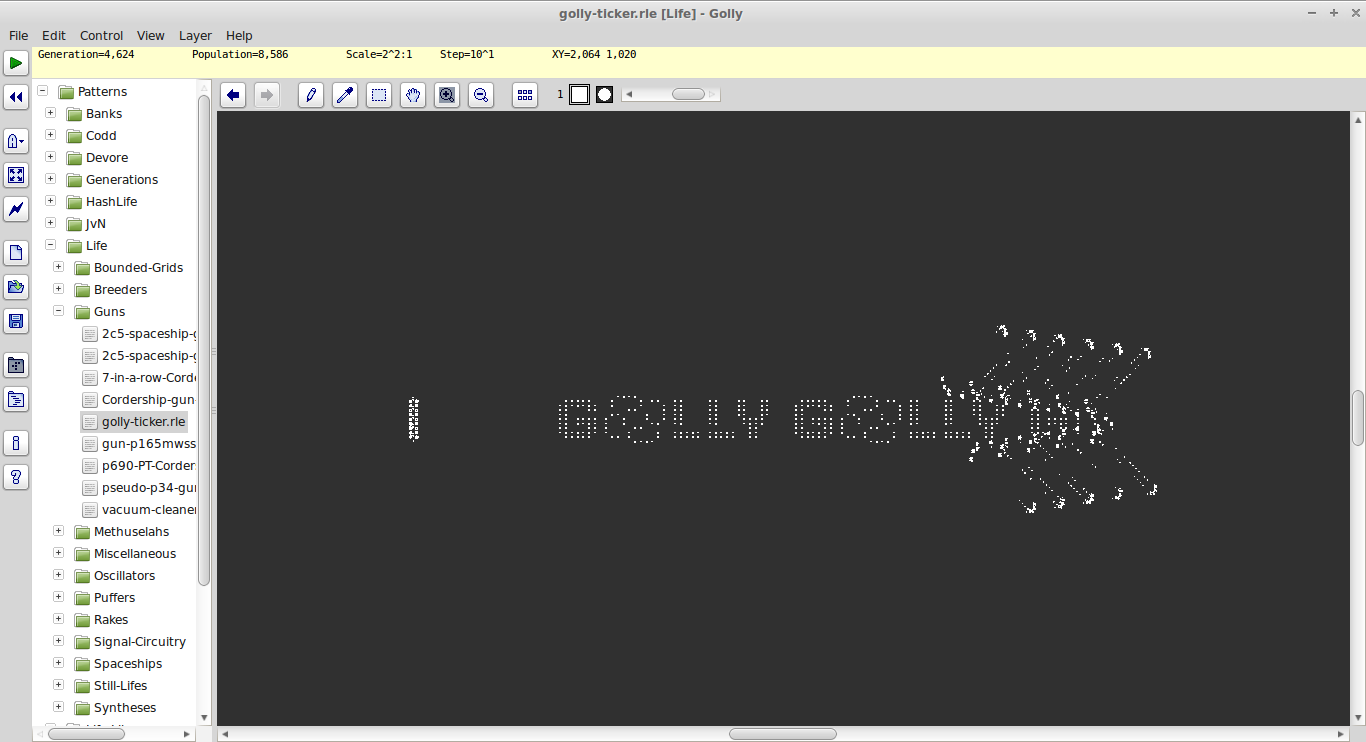
Однак існуючі реалізації гри “Життя” не обмежуються онлайн-вейсіями на JavaScript. Звісно, подібні програми недостатньо швидкі та обмежені ресурсами браузеру порівняно з реалізаціями на інших платформах та мовах програмування. Також, ймовірно, що вищезазначені варіанти не використовують найшвидші алгоритми обрахунку наступної ітерації, тому навіть не зважаючи на обмеження браузеру такі програми будуть повільнішими за більш розвинені аналоги. Найбільш популярним мультиплатформенним застосунком для роботи з клітинними автоматами, зокрема грою “Життя”, є Golly — програмою, що симулює гру “Життя” та інші двовимірні клітинні автомати зі зручним інтерфейсом та швидким виконанням. Для цього програма застосовує алгоритм розрахунку великих площ клітинних автоматів під назвою Hashlife. Цей алгоритм дозволяє використовувати певну надмірність інформації в еволюції структурованих систем, наприклад, космічних кораблів чи осциляторів. Він надає можливість програмі запам’ятовувати результат еволюції певних конфігурацій, заносячи його в хеш-таблицю, та відразу розташовувати його замість багатократного обрахунку одних і тих самих елементів у великій кількості.

Наприклад, якщо в даній конфігурації автомату використовується декілька тисяч легких космічних кораблів, алгоритму необхідно буде прорахувати їх переміщення лише один раз, зберегти конфігурацію у хеш-таблицю, а далі, коли буде потрібно знову обрахувати рух цього космічного корабля, алгоритм звернеться до хеш-таблиці і відразу розмістить результат еволюції цієї конфігурації для всіх існуючих кораблів цього типу. Це дозволяє пришвидшити обрахування складних, проте систематизованих конфігурацій у тисячі разів, дозволяючи створювати гігантські структури та обраховувати їх у реальному часі. Алгоритм навіть дозволяє обрахунок різних частин автомату з різними швидкостями, максимізуючи швидкість роботи без втрати коректності виконання.

За допомогою використання алгоритму Hashlife програма Golly здатна обраховувати складні структури у реальному часі, дозволяючи створювати дуже великі космічні кораблі та дивитись на процес їх руху. Зокрема, разом з програмою постачається набір конфігурацій, як для гри “Життя”, так і для інших клітинних автоматів. Серед них можна побачити типових представників таких класів структур, як осцилятори, космічні кораблі, рущниці та ще складніші структури. Ці структури дозволяють детально розглянути ефективність роботи описаного вище алгоритму на практиці: навіть при великій швидкості переходу до наступної ітерації програма здатна обраховувати результат в реальному часі. Нижче наведено приклад однієї зі складних структур, що здатні оброблятися алгоритмом Hashlife значно швидше, ніж класичним методом:



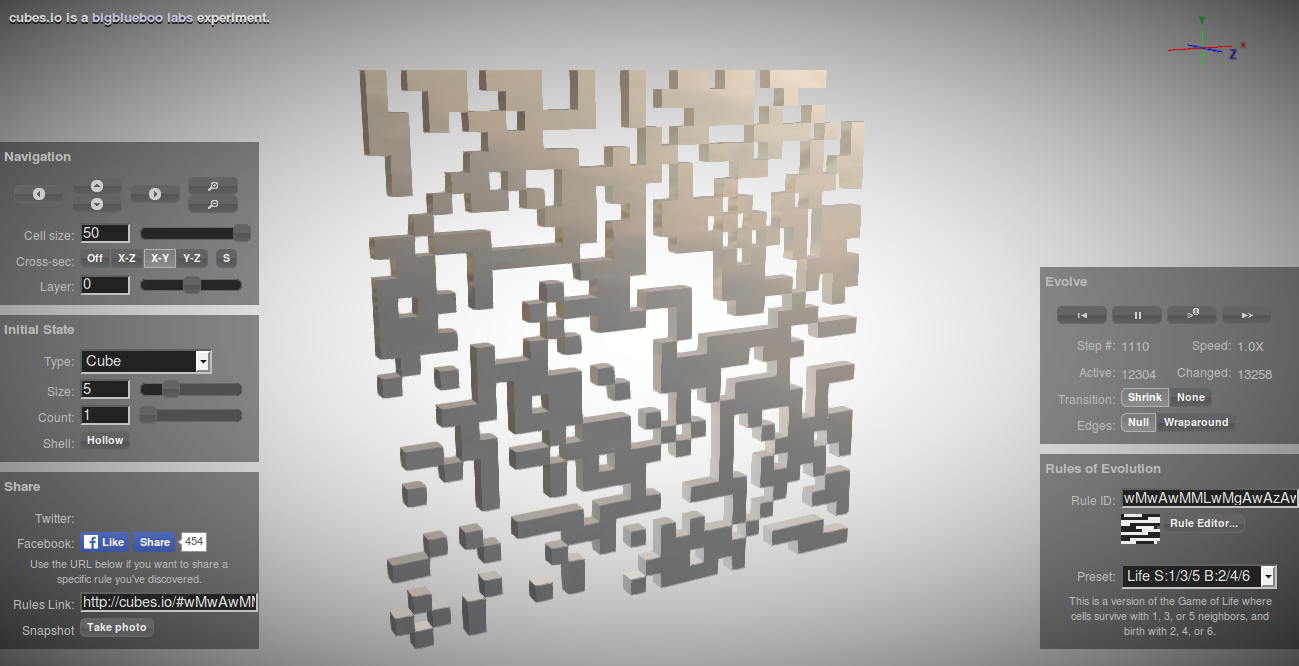
Функціонал програми дозволяє створювати власні конфігурації, призупиняти та змінювати швидкість переходу на нову ітерацію, зберігаючи при цьому швидкодію алгоритму Hashlife. Golly дозволяє масштабувати та розглядати як певні окремі елементи конфігурації, так і загальну структуру. При створенні власної конфігурації застосунок дозволяє копіювати вже створені структури, таким чином полегшуючи роботу з великими структурами. Створення нових космічних кораблів (наприклад, з новим кутом руху чи новою швидкістю, для якої раніше не було знайдено космічного корабля) зазвичай проводиться дослідниками саме в цьому застосунку, тому його можна назвати стандартом для гри “Життя”, хоча й існують інші альтернативи. Нижче зображено інтерфейс програми:



### Реалізації тривимірних клітинних автоматів.

Тривимірні клітинні автомати є менш розповсюдженими порівняно з двовимірними. Це пов’язано з більш високою складністю в аналізі таких автоматів, пошуку правил, за якими автомат виказує ознаки хаосу та створення стабільних структур. Крім того, відображення процесу роботи ускладнене тим, що на двовимірному екрані важко відобразити еволюцію тривимірної структури. Тому завдання візуалізації тривимірних клітинних автоматів й досі залишається актуальним.

Яскравим прикладом якісної реалізації цього завдання є cubes.io – WebGL-реалізація тривимірного клітинного автомату для браузерів. Дана реалізація не надає настільки широкого функціоналу, як Golly, зокрема, в ній взагалі відсутня можливість задавати власну початкову конфігурацію чи змінювати стан клітин в процесі роботи, а також програма працює лише з автоматами, клітини в яких мають лише два можливих стани. Проте дана реалізація має певні плюси: вона має зручний інтерфейс, що дозволяє регулювати форму початкової конфігурації, правила переходу до наступного стану, а також змінювати формат відображення. Як можна побачити на скріншотах нижче, програма дозволяє як розглядати структуру з різних сторін в трьох вимірах, так і дивитись на окремі шари у двох з трьох координат:



Таким чином, разом з можливістю призупинення та покрокового перегляду, застосунок надає можливість детально розглянути процес еволюції обраної конфігурації при даних правилах. Також програма динамічно обчислює освітлення та віддзеркалення зображення на поверхні клітин, що дозволяє чіткіше бачити окремі клітини та таким чином допомагає у візуалізації клітинного автомату. Навігацію камери реалізовано за допомогою миші та клавіатури, що дозволяє зручно повертати камеру на необхідний кут.

Загалом візуалізація тривимірного клітинного автомату на веб-сайті cubes.io є досить гарною репрезентацією процесів, що відбуваються в автоматах, що розглядаються, проте не дає достатнього контролю над їх еволюцією чи можливості керувати її та направляти в потрібному напрямку. Такі висновки дозволяють визначити ціль даної роботи як реалізацію системи, що дозволятиме в тому ж ступені коректно та зручно відображати тривимірні клітинні автомати, проте надасть більший контроль над еволюцією системи.

# Опис функціональних можливостей розроблюваного застосунку.

## Основна ідея роботи.

Тривимірні клітинні автомати є галуззю, в якій створено значну кількість програмних реалізацій, проте цей різновид клітинних автоматів все ж доволі слабо досліджений. Порівняно з двовимірними клітинними автоматами, тривимірні дозволяють реалізовувати набагато більшу множину правил, при цьому певні стійкі структури є складнішими за їх двовимірні аналоги, адже вони мають бути стабільними в трьох вимірах замість двох. Через це пошук набору правил, що надасть клітинному автомату властивості двовимірних аналогів, зазначених у попередньому розділі, ускладнюється. Існуючі реалізації тривимірних клітинних автоматів не надають достатнього функціоналу для їх дослідження та пошуку автоматів 4 класу, що виказують ознаки хаотичності та формування стабільних структур.

З цієї причини виникає необхідність в створенні нового програмного застосунку, здатного задовільнити потреби, що виникають при розгляді тривимірних клітинних автоматів. Зазначимо, яких саме цілей необхідно досягти при розробці даного застосунку:

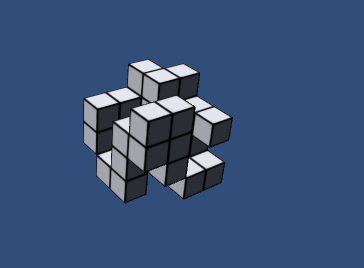
* Основною метою роботи програми є обрахунок еволюції тривимірних клітинних автоматів. У відриві від способу зображення клітинного автомату, обрахунок еволюції становить з себе процес визначення, які клітини мають загинути, а які — зародитись на наступній ітерації. Зазвичай для визначення наступного стану автомату використовуються чітко визначені правила, що у даному випадку базуються на кількості заповнених сусідів відповідної клітини. Так як розроблюваний застосунок підтримує ймовірнісні клітинні автомати, то клітина може зародитись, якщо вона має кількість сусідів, що дорівнює випадково визначеному для конкретного випадку числу, яке знаходиться в заздалегідь визначеному діапазоні. Границі загибелі клітини від самотності та перенаселення також визначаються випадково в кожному окремому випадку на основі заданих меж значень. Окрім цього, автомат може мати різний розмір, хоча й завжди матиме лише кубічну форму. Також важливо коректно обраховувати наступну ітерацію: зміни, що відбуваються при переході до наступної ітерації, не мають впливати на кінцевий результат, адже клітини, які змінять свій стан, визначаються лише на основі попереднього стану. Підтримка даного функціоналу є основною для застосунку, проте й найпростішою в розробці: загалом цей функціонал реалізується чітко заданими простими алгоритмами переходу до наступного стану.
* Даний застосунок має підтримувати коректне відображення процесу еволюції тривимірного клітинного автомату. Одною з основних функціональних можливостей застосунку, який потребується для досліждення, є можливість відображення як поточного стану автомату, так і процесу переходу до наступного стану. Основною проблемою в даному випадку є той факт, що, на відміну від двовимірних автоматів, відобразити на екрані відразу весь тривимірний автомат немає можливості. Тому необхідно вирішити проблему його відображення таким чином, щоб залишити можливість цілком розглянути структуру клітинного автомату незалежно від ітерації. При цьому велика кількість клітин з декількома можливими станами може ускладнити розуміння структури поточної ітерації та її зміни при переході до наступної ітерації. Тому необхідно забезпечити таку форму відображення, при якій окремі клітини можна чітко розрізнити, навіть при великій їх кількості, а перехід до наступного кроку дає чітке представлення, яка частина клітин змінила свій стан.
* Розпізнавання кінцевого стану клітинного автомату є проблемою, що постає не лише перед тривимірними, але й перед двовимірними клітинними автоматами. Вона полягає в тому, що клітинні автомати 4 класу часто припиняють свою еволюцію, переходячи до стабільного стану, що містить лише стійкі структури (наприклад, натюрморти та осцилятори у грі “Життя”). Проте зазвичай це не означає, що стан клітин автомату більше не змінюється. Зазвичай еволюція стає тривіальною, коли всі зміни відбуваються лише серед циклічних структур, що повторюються через певну кількість ітерацій. При формуванні великої кількості подібних структур повне повторення стану автомату може відбутись через декілька сотень ітерацій чи навіть більше. Тому розпізнавання цього етапу еволюції є складною задачею — необхідно порівняти поточний стан клітинного автомату з усіма попередніми ітераціями, адже якщо стан автомату повністю повторює один з минулих станів — це є головною ознакою зациклення. Застосунок повинен вирішувати цю задачу методом, що дозволить обчислювати тисячі ітерацій та перевіряти нетривіальність еволюції навіть для великих обмежених клітинних автоматів.
* Необхідно надати користувачу можливість змінювання стану клітин автомату в реальному часі. Однією з основних проблем програмних реалізацій тривимірних клітинних автоматів є складність їх задання за допомогою графічного інтерфейсу. Навіть якщо проблема відображення тривимірного автомату вже вирішена, можливість змінювання стану клітин у трьох вимірах потребує особливого підходу.
* Застосунок повинен надавати можливість зберігати поточний стан клітинного автомату та завантажувати його в разі необхідності. Подібний функціонал дозволить зберігати знайдені стабільні структури чи конфігурації, здатні до довготривалої еволюції, що значно перевищує складністю початковий стан.

Розглянемо детальніше, яким саме чином планується вирішити вищезазначені проблеми.

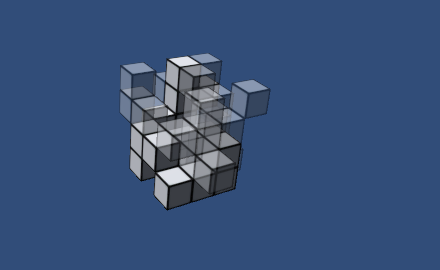
## Відображення стану тривимірного автомату.

Відображення поточного стану тривимірного автомату зазвичай виконується за допомогою тривимірних моделей, що зображають окремі клітини. Зокрема, у випадку клітинного автомату з двома можливими станами клітини, наявність в певній позиції моделі вказує на “живий” стан відповідної клітини, а відсутність моделі — на “мертвий” стан. Розповсюдженим способом зображення заповненої клітини є звичайна модель кубу з певною текстурою та шейдерами для більш деталізованого відображення стану автомату.

При подібному способі відображення стану виникає необхідність прийняти певні рішення, що впливатимуть на те, наскільки легко буде розрізнити окремі клітини та загалом зрозуміти структуру автомату. Зазвичай доцільно розміщувати моделі впритул один до одного, таким чином представляючи неперервну послідовність клітин. Наявність між клітинами проміжку може спростити розпізнавання меж клітин, проте ускладнить розгляд пустих клітин, адже через наявність цього проміжку зрозуміти, скільки пустих клітин знаходиться між двома заповненими, становиться складніше. Ще одним можливим варіантом є явне зображення сітки навіть між пустими клітинами. Це остаточно вирішує проблему з підрахунком кількості пустих клітин між двома заповненими, однак створює іншу проблему: через велику кількість зайвих ліній розглядати структуру клітин, що знаходяться далеко від краю автомату, становиться набагато складніше. Через це подібний підхід не є доцільним. Проте для чіткого розпізнавання меж заповнених клітин ребра кубу відмічені чорним кольором, що дозволяє не використовувати проміжки між клітинами. На зображенні нижче можна побачити приклад подібної системи відображення:

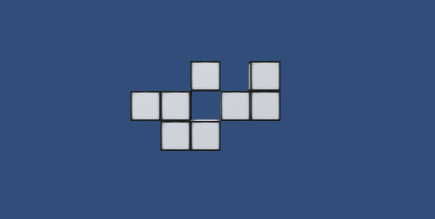


Окрім вибору власне способу відображення окремої клітини, виникає необхідність у вирішенні й інших питань. Наприклад, для створення ефекту об’єму клітин, який необхідний для зручного розгляду стану тривимірного автомату, необхідно застосовувати освітлення. Наявність джерела світла у системі відображення забезпечує перспективу моделей клітин, дозволяючи таким чином розрізнити, на якому рівні у тривимірній структурі знаходяться клітини, що розглядаються. Також необхідний репрезентативний спосіб зобразити появу та зникнення клітини. При зміні стану до користувача необхідно чітко донести інформацію про те, як саме змінюється стан автомату. Для цього можна використати певну анімацію моделей клітин, що з’являються чи, навпаки, зникають при переході до наступної ітерації. Серед можливих анімацій варто зазначити поступову зміну прозорості від повністю невидимого до звичайного стану при появі та навпаки при зникненні клітини, а також зменшення та, відповідно, збільшення клітини з часом. Нижче можна побачити приклад анімації зміну стану клітин методом змінення їх прозорості. Зображення показує стан автомату в процесі переходу.



Однак для реалізації повноцінного засобу розгляду структури тривимірних автоматів недостатньо лише задати тривимірну модель його стану. Дуже важливо також надати засоби зміни точки огляду. Вочевидь, для детального огляду стану автомату необхідна можливість сповільнювати та пришвидшувати, а також зовсім зупиняти процес переходу до наступної ітерації. Проте незалежно від швидкості зміни ітерацій функціонал для огляду автомату з будь-якого кута є обов’язковим. Найпростіший спосіб реалізації цього функціоналу — повернення камери навколо центру клітинного автомату. Наближення та віддалення зображення зручно контролювати за допомогою коліщатка миші. Повернення ж зображення можна реалізувати за допомогою руху миші. Можливим варіантом є автоматичне повернення за кутами Ейлера при руху миші, але при цьому виникає проблема використання інших елементів інтерфейсу: для натискання певної кнопки чи інтеракції з іншими елементами інтерфейсу необхідно рухати мишею, проте це викликатиме поворот кута огляду, що ускладнить роботу з програмою. Тому більш доцільною системою є поворот у вказаному напрямку при затисканні лівої кнопки миші. Це дозволить вільно повертати зображення в будь-якому напрямку інтуїтивно зрозумілими рухами, при цьому не заважаючи користуватись інтерфейсом програми. У призупиненому стані це дозволить детально розгледіти поточну ітерацію, а під час роботи автомату дозволить побачити процес переходу до нового стану з кута, який можна динамічно обирати в процесі роботи.

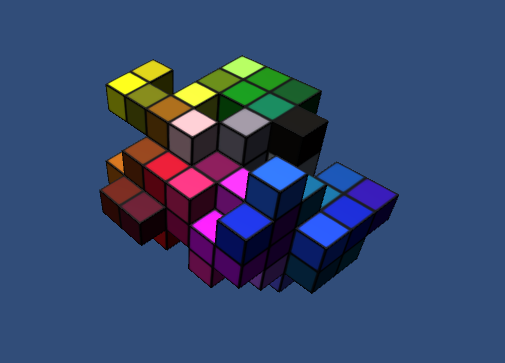
Проте навіть такі засоби можуть виявитись недостатніми для візуалізації внутрішньої структури автомату. При великій кількості заповнених клітин зовнішні клітини можуть закрити огляд внутрішніх, при цьому зміна кута огляду не допоможе. Тому важливим функціоналом, який потребує застосунок, є можливість відобразити лише один шар структури автомату. Найбільш зручним варіантом реалізації цієї функції є можливість обрати один з чотирьох форматів відображення: тривимірний та по одному формату на кожну можливу ортогональну площину відображення (вісі X/Y, X/Z та Y/Z відповідно). При відображенні в одній з ортогональних площин усі клітини стають невидимі, окрім одного шару клітин у відповідній площині. При цьому камеру необхідно розташувати перпендикулярно площині, що розглядається, таким чином формуючи псевдодвовимірне відображення автомату. Вибір конкретного шару для відображення можна реалізувати за допомогою повзунка, при цьому перехід від одного шару до іншого дозволятиметься робити динамічно, навіть під час роботи автомату. Завдяки цьому застосунок дозволить розглянути структуру автомату будь-якої складності у зручному форматі, у тому числі динамічно під час роботи автомату. Приклад відображення окремого шару автомату зображений на малюнку нижче:



Окрім вищезазначеного функціоналу, програма також підтримуватиме можливість відображення клітин автомату різними кольорами. Детальніше про те, яким чином клітинам буде задаватись необхідний колір, описано нижче. Подібну можливість можна використовувати декількома способами. Найтривіальнішим з них є помічення важливих клітин. У випадку використання кольорової ідентифікації клітин окремі клітини чи навіть регіони можна відобразити певним кольором. Це дозволить чітко бачити дану клітину чи регіон навіть на фоні сотень чи тисяч інших клітин. Наприклад, користувача цікавить стан конкретної клітини в процесі еволюції конфігурації. Він може позначити цю клітину окремим кольором, щоб чітко вирізняти її на фоні інших клітин. Далі користувач може розпочати еволюцію системи. Якщо розгляду клітини заважають інші клітини, доцільним є використання відображення лише шару, який містить цю клітину. Позначення ж цілого регіону одним кольором може бути використано для відстеження, чи заповнена хоч одна клітина з цього регіону, або при розгляді конкретної ділянки еволюції системи.

Іншим варіантом використання системи кольорової ідентифікації клітин є наслідування кольору при переході до наступного стану клітинного автомату. При застосування цього режиму якщо зароджується певна клітина при переході до наступної ітерації, то колір даної клітини визначається як середнє значення кольорів клітин-сусідів із випадковим зміщенням. Початкове задання кольору клітин можна провести до запуску еволюції. Після цього початкова кольорова матриця розповсюдиться автоматом, створюючи фрактальні кольорові візерунки. Це можна використовувати для формування тривимірних графічних структур, що будуть становити з себе випадковий результат еволюції початкової конфігурації. Також подібну кольорову систему можна використовувати для імітації різноманітних тривимірних процесів, наприклад, змішування різнорідних речовин. У цьому випадку кожна речовина буде позначена окремим кольором, а клітини проміжних кольорів відображатимуть пропорції та ступінь змішування речовин у даному регіоні.

Наведемо приклад структури, яку можна отримати шляхом використання кольорової диференціації клітин:



## Розпізнавання переходу автомату до тривіального стану.

### Випадок детермінованого клітинного автомату

При дослідженні властивостей клітинного автомату одним з критеріїв оцінки може слугувати час еволюції конфігурації, тобто кількість ітерацій від початкової до переходу в тривіальний стан. Тривіальним станом зазвичай називають стан, при якому подальша еволюція системи є легко передбачуваною. Наприклад, автомат знаходиться в тривіальному стані, якщо він містить лише стабільні структури, що не змінюються з часом (“натюрморти” в термінах гри “Життя”), циклічні структури (“осцилятори”) та циклічні структури, що рухаються в певному напрямку (“космічні кораблі”). Проблема визначення тривіального стану має бути вирішена для надання функціоналу з підрахунку часу еволюції конфігурації, що дозволить детальніше дослідити особливості тривимірного клітинного автомату.

Навіть для детермінованого автомату реалізація подібного функціоналу породжує певні проблеми. Найпростішим способом перевірки тривіальності системи є перевірка, чи змінився стан хоча б однієї клітини автомату. Якщо цього не сталося, це означає, що автомат або пустий, або складається лише з натюрмортів. Проте подібний підхід не дозволяє визначити перехід до тривіального стану, що містить циклічні структури. Тому цей метод необхідно розширити, порівнюючи поточний стан з усіма попередніми станами автомату. Якщо поточний стан повністю відповідає одному з попередніх, то у випадку детермінованого автомату це означатиме зациклювання еволюції, адже з кожного стану можна перейти лише в один наступний стан. Проте певні еволюції можуть тривати тисячі ітерацій, що у випадку великого розміру автомату може викликати проблему: кожна наступна ітерація потребуватиме все більше й більше часу на обрахунок. Тому необхідний ефективний спосіб порівняння станів автомату, що дозволить обраховувати подібні довготривалі еволюції.

Доволі простим рішенням проблеми є перетворення стану автомату в послідовність бітів для подальшого побітового порівняння поточного стану з попередніми. Так як клітинний автомат, що розглядається, може приймати лише один з двох можливих станів, то стан клітини можна описати одним бітом. В цьому випадку побітове порівняння двох значень є найшвидшим способом. Проте навіть у цьому випадку особливо великі автомати з довготривалою еволюцією можуть викликати проблеми на великій швидкості обрахунку чи слабкій конфігурації комп’ютера. Тому для прискорення обрахунку подібних еволюцій пропонується формувати хеш-значення поточного стану, та порівнювати саме його. Формування хеш-значення з поточного стану також займає певний час та має виконуватись на кожній ітерації, проте це прискорить порівняння двох станів між собою, яке на пізніших ітераціях виконуватиметься тисячі разів на кожному кроці, тому час на формування хешу стану буде виправданим. Так як ймовірність, що два різних стану автомату матимуть однаковий хеш, є майже нульовою, то порівняння хешу достатньо, щоб визначити, чи зациклився клітинний автомат, перейшовши таким чином у тривіальний стан.

Якщо ж клітинний автомат перейшов у тривіальний стан з наявністю “космічних кораблів”, тобто циклічних структур, що постійно рухаються в певному напрямку, цей варіант додатково розглядати немає необхідності. Якщо в процесі руху “космічний корабель” опиниться достатньо близько від іншої стабільної структури, то еволюція може продовжитись, таким чином цей стан не є тривіальним. Якщо ж “корабель” рухатиметься без перешкод, він з часом досягне краю автомату та більше не зможе зберігати свою постійність. Таким чином, скінченність автомату виключає необхідність ускладнення обрахунку переходу до тривіального стану у випадку виникнення рухомих стабільних структур.

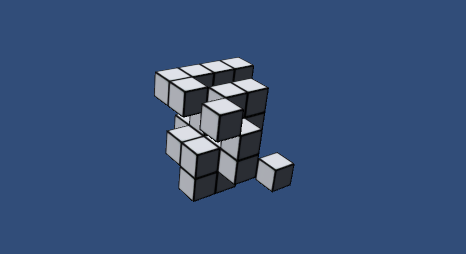
Прикладами тривіальних станів клітинних автоматів можуть слугувати натюрморти з гри “Життя”.

### Випадок недетермінованих (ймовірнісних) клітинних автоматів.

Вищезазначений підхід працює для детермінованих клітинних автоматів. Принциповою особливістю таких автоматів є те, що вони з певного стану завжди переходять в один конкретний стан. Проте ймовірнісні клітинні автомати не мають такої властивості. Через наявність елементу випадковості, певні стани з певною ймовірністю можуть перейти у два чи більше різних наступних стана. Через це навіть перехід до стану, що вже траплявся в процесі еволюції автомату, не гарантує, що еволюція клітинного автомату стала тривіальною. Загалом, у подібних умовах тривіальний стан описується значно складніше, адже певні структури, що випадково змінюються у часі, можна суб’єктивно також назвати тривіальними, якщо існує лише невелика кількість станів, в які дана конфігурація може перейти з певним розподілом ймовірностей.

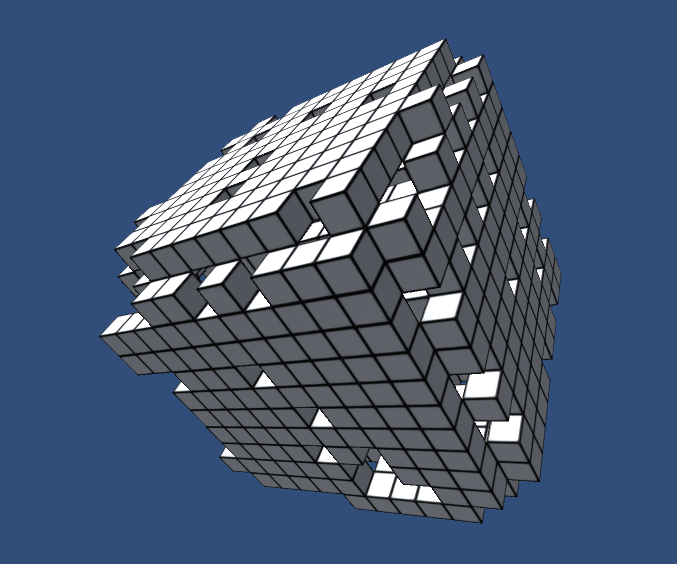
Розглянемо декілька можливих варіантів еволюції ймовірнісних клітинних автоматів. Найпростішим варіантом розвитку є поступове зменшення кількості заповнених клітин та “смерть” клітинного автомату, при якій всі клітини становляться пустими. Зазвичай подібний сценарій характерний для клітинних автоматів, в яких для зародження клітини потребується занадто велика кількість сусідів, як і для збереження життя вже існуючої. У цьому випадку якщо більшість клітин початкової конфігурації й має достатню для життя кількість сусідів, клітини на краю заповненої області їх не матимуть, що викличе каскадну реакцію гибелі клітин, а через низьку швидкість відродження кількість клітин буде знижуватись до нуля.

Наведемо приклад даного варіанту еволюції автомату. Нехай маємо автомат, що заданий наступними правилами: клітини зароджуються при кількості сусідів, що дорівнює випадковому значенню між 10 та 12, гине від самотності при кількості сусідів, що менше за 6-9, та від перенаселення, якщо сусідів, якщо їх більше за 12-14. У цьому випадку зображена нижче конфігурація зникне через 3 кроки. Так як для формування клітини необхідно щонайменше 10 сусідів, а клітина на пустому шарі може мати лише 9 сусідів із сусіднього шару, то на пустих шарах клітини ніколи не зароджуються. Конфігурація не може вийти за початкові межі та швидко гине.



Зворотнім варіантом є заповнення усього поля. Відповідно, якщо смерть клітини наступає лише при дуже великій кількості сусідей, може трапитись ситуація, коли навіть на границях автомату клітини не гинуть. Повністю автомат заповниться лише у тому випадку, якщо максимальна кількість сусідів, при якому клітина все ще не гине від перенаселення, перевищує можливу кількість сусідів (у випадку тривимірного автомату — 26). Однак автомат все одно може бути заповнений губчатою структурою, в якій заповнені клітини перемежовуються пустими, формуючи необхідну кількість сусідів для того, щоб ні одна клітина не гинула, але й жодна пуста клітина не може заповнитись.

Прикладом цього варіанту розвитку може слугувати автомат з наступною конфігурацією: зародження клітини відбувається при 6-7 сусідах, загибель від самотності — при менш ніж 4-5 сусідах, а загибель від перенаселення — при більш ніж 8-9 сусідах. Подібні правила дозволяють початковій конфігурації доволі швидко розповсюджуватись, при цьому клітини гинуть недостатньо швидко, тому структура заповнює увесь автомат. Сформована в результаті структура становить вкладені один в одного куби з заповнених клітин, що розмежовані шаром товщиною в одну пусту клітину. Подібна структура постійно змінюється всередині, тому її важко назвати тривіальним станом. Нижче зображена конфігурація, що отримана в результаті описаного процесу еволюції:



Окрім цих двох тривіальних варіантів існують й складніші типи еволюції. Певні правила клітинних автоматів здатні формувати стабільні структури в процесі еволюції. Перехід до тривіального стану може відбуватись шляхом зупинення будь-яких змін, що не входять до складу стабільних структур. Для цього не обов’язково необхідне заповнення всього автомату, а також при цьому не відбувається його остаточна смерть. Однак стабільні структури в ймовірнісних автоматах мають певні особливості. Якщо ми розглядаємо варіант ймовірнісних клітинних автоматів, при якому зародження клітини можливе в певному диапазоні кількостей сусідів, то структура може залишатись стабільною, зокрема, якщо ймовірнісний фактор ніяк не впливає на еволюцію системи. Це може відбутись, якщо всі пусті клітини, що є сусідами даної системи, мають меншу кількість сусідів, ніж необхідно для їх заповнення, а всі заповнені клітини мають кількість сусідів, яка є більшою, ніж верхня межа загибелі від самотності, та меншою, ніж нижня межа загибелі від перенаселення. В певних випадках такий діапазон кількості сусідів взагалі відсутній, і можливий варіант, що клітина одночасно має шанс загинути від самотності та перенаселення. Тоді існування стабільних структур взагалі неможливо — будь-яка структура через певний час випадково втрачає певні клітини і, як результат, стабільність.

Нижче зображено стабільну структуру, що виникає при тій самій конфігурації, що й у минулому прикладі. Структура становить з себе куб 2х2х2 клітин:

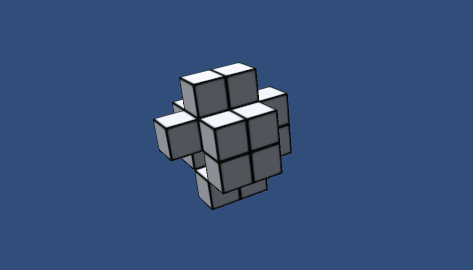


Циклічні ж стабільні структури створити набагато складніше: так як зародження клітини залежить від випадкової кількості сусідів, то на певному кроці циклу клітина може створитись чи не створитись, що значно ускладнює формування таких структур. Значно ймовірніша поява структур, де певна клітина може випадковим чином заповнюватись чи видалятись, при цьому це ніяк не впливає на стабільність усієї системи. Це може статись, якщо поява цієї клітини не збільшує кількість сусідів інших клітин таким чином, щоб вони покинули “стабільну” зону між загибеллю від перенаселення та самотності. Таким чином, в ймовірнісних клітинних автоматах поява “змінних” стабільних структур зазвичай відбувається, коли поява окремих клітин не є обов’язковою для стабільності системи.

Найбільший інтерес представляють правила, за яких автомат здатен до активного розповсюдження, що не призводить до швидкого переходу у тривіальний стан. Зазвичай це автомати 3 та 4 класу. Саме такі клітинні автомати можуть викликати проблеми при підрахунку часу еволюції конфігурації. Описана вище проблема порівняння поточної ітерації з усіма попередніми стає ще більш суттєвою, адже навіть одне повторення не гарантує зациклення, а відсутність повторення ще не гарантує відсутність зациклення. Наприклад, якщо в процесі еволюції конфігурації з’явилась стабільна структура з випадково зароджуючимися клітинами, не впливаючими на стабільність структури, то вона може завадити коректному детектуванню зациклення і, відповідно, переходу в тривіальний стан.

Існують також варіанти напівстабільних структур, що здатні значний час залишатись стабільними, проте мають ймовірність вийти з цього стану. Наприклад, структура має декілька клітин, що з певною ймовірністю зароджуються та гинуть, не впливаючи на стабільність структури. Проте, якщо всі клітини випадково виникнуть на одній ітерації, в сумі їх буде достатньо, щоб сусідня клітина загинула від перенаселення, а структура через це втратила стабільність. Такі структури можуть викликати ситуації, коли тривіальний на перший погляд стан може через декілька десятків чи навіть сотен ітерацій вийти з подібного зациклення та продовжити активну еволюцію. Після цього еволюція може або продовжитись протягом значного часу, або завершитись одним з описаних раніше способів (повністю спустіти, заповнити все поле чи перейти до повністю стабільного, або навіть іншого напівстабільного стану).

Наведемо приклад такої напівстабільної структури. Нехай маємо такі правила клітинного автомату: зародження клітини при 6-8 сусідах, загибель від самотності при 4-6 та від перенаселення при 8-10 сусідах. Зображена нижче конфігурація є напівстабільною при таких правилах. Структура є майже симетричною, але для повної симетричності не вистачає однієї клітини в лівій нижній частині зображення. Дана клітина зародиться на наступній ітерації з ймовірністю 1/3. Проте найлівіша та нижня ліва клітини також мають ймовірність 1/3 загинути від самотності. Якщо цього не трапиться, то структура стабілізується. Якщо ж хоча б одна з клітин зникне до того, як з’явиться нова клітина, структура дестабілізується і швидко перейде до пустого стану.



Цей приклад можна екстраполювати на менші ймовірності дестабілізації. Наприклад, якщо ймовірність виходу зі стабільного стану складає лише 10%, може пройти багато ітерацій, перед тим як система дестабілізується. Таким чином в напівстабільних системах відсутня гарантія того, що збереження структури протягом певної кількості ходів веде до її збереження на подальших ітераціях.

На основі розглянутих сценаріїв можна зробити висновок, що для ймовірнісних автоматів недоцільно використовувати той самий підхід до визначення часу еволюції, що й для детермінованих автоматів. Більшість структур, що є стабільними протягом нескінченно великих проміжків часу, взагалі не змінюються з часом. Формування стабільних структур з клітинами, що випадково змінюють свій стан, не порушуючи при цьому стабільність структури, є менш ймовірним. Виникнення ж зациклених структур у випадку ймовірнісного клітинного автомату є ще менш ймовірним. Так як подібний функціонал не є основною метою розробки програми, адже він лише слугує для зручного визначення часу еволюції системи, а ймовірність виникнення ситуацій, в яких з’являються складніші стабільні структури, є досить незначною, програма підтримуватиме лише визначення переходу до тривіального стану, що виражений повною відсутністю змін протягом значного часу. Надання складнішої системи визначення тривіального стану суттєво знизить продуктивність програми, що не є доцільним для обробки незначного проценту ситуацій другорядними підсистемами.

## Задання стану клітин автомату.

### Безпосереднє задання стану окремих клітин.

Можливість задання стну клітин є обо’язковим функціоналом програми. Він дозволить досліджувати окремі, заздалегідь задані конфігурації на предмет їх стабільності та відповідності певним критеріям (наприклад, великий час еволюції). Однак процес задання стану клітин тривимірного автомату ускладнюється тим, що користувач не може бачити усієї структури відразу та не здатен рухати покажчик у трьох вимірах одночасно. Перша проблема вирішується засобами відображення тривимірних автоматів, що були описані в одному з попередніх підрозділів. Друга проблема ж є розвитком цих ідей з додаванням елементів інтерактивності.

Головні проблеми, які потрібно вирішити при реалізації даного функціоналу, це спосіб відображення поточної позиції покажчику та спосіб зручного переміщення покажчика у тривимірному просторі. При відображенні поточної позиції покажчику необхідно розглянути два варіанти: коли покажчик знаходиться на заповненій клітині, та коли він знаходиться на пустій клітині. Доволі репрезентативним способом зображення покажчика є періодичне миготіння відповідної клітини. Якщо клітина пуста, на ній періодично з’являтиметься напівпрозорий куб, чия прозорість поступово мінятиметься від нуля до певної величини, відображаючи таким чином поточну обрану клітину. Якщо ж клітина заповнена, відбуватиметься зворотній процес: періодично куб в даній клітині ставатиме напівпрозорим, поступово змінюючи прозорість від повністю непрозорого до певної величини та назад.

Друга проблема ж вирішується шляхом використання двовимірної репрезентації одного з шарів автомату. Так як програма містить функціонал з відображення певного шару автомату, до нього можна додати можливість обирати конкретну клітину курсором миші чи стрілками клавіатури. Змінювати ж шар, на якому необхідно встановити стан клітини, можна за допомогою повзунка чи коліщатка миші. Подібна система дозволить обрати будь-яку клітину та змінити її стан зручним чином. Доцільним також є надання пожливості затиснути ліву кнопку миші та, проводячи по конкретному шару, заповнювати всі клітини, по яким проведено курсором. Це дозволить заповнювати великі масиви клітин швидше, ніж при індивідуальному їх заданні.

Окрім цього, необхідно надати можливість задання кольору клітин. Для цього можна використати спеціальний режим задання кольору. В цьому режимі замість зміни стану клітини відбуватиметься вибір даної клітини задля задання її кольору. Після вибору клітини спеціальна панель дозволить обрати колір цієї клітини на палітрі чи безпосередньо повзунками. Одночасно може бути обрана лише одна клітина.

### Початкове задання конфігурації автомату. Завантаження та запис.

Окрім можливості задати значення окремих клітин, необхідним є також функціонал з задання початкової конфігурації автомату. Так як при заданні початкової конфігурації може виникнути потреба в нетривіальному заповненні великої кількості клітин, то необхідне рішення, що дозволить зменшити необхідність подібного підходу к мінімуму.

Першим способом вирішення цієї проблеми є випадкове задання початкової конфігурації. В багатьох випадках для перевірки властивостей певного набору правил тривимірного автомату найбільш доцільним є саме використання випадкового початкового стану, адже воно дозволить швидко розглянути еволюцію автомату на різних початкових умовах, що не є виродженими випадками. Доцільним є заповнення лише певної частини автомату замість усього поля, адже таким чином можна побачити, наскільки розширюється займана площа у процесі еволюції. Якщо ж все поле буде заповнено випадковими станами клітин, простору для розширення еволюція не матиме, що може нівелювати цінність експерименту. Також можливо надати користувачеві функціонал з визначення щільності заповнення початкової ділянки поля, що можна визначати через ймовірність заповнення окремої клітини.

Окрім задання випадкового початкового стану, необхідно також надати функціонал зі збереження та завантаження певних конфігурацій. Якщо користувач досліджує не загальні властивості певного автомату, а властивості конкретних конфігурацій, в нього може виникнути необхідність багатократно задавати одну й ту саму конфігурацію. Збереження дозволить задати складну конфігурацію лише один раз, використовуючи випадкове задання чи безпосереднє заповнення окремих клітин, а далі зберегти конфігурацію для подальшого користування. Також збереження поточного стану може бути корисним для перезапуску еволюції, що дозволить повернути автомат в початковий стан до початку еволюції.

Завантаження поточного стану заповнить необхідні клітини згідно збереженої конфігурації, відновлюючи стан автомату. Інформацію про стан автомату можна зберігати шляхом запам’ятовування правил поточного автомату, його розмірності та бітового представлення станів клітин, подібного до того, що пропонувалось використовувати для підрахунку переходу до тривіального стану. Крім того, доцільним є збереження кольорової ідентифікації клітин автомату. Задання необхідної кольорової схеми автомату з метою зручного відображення клітинного автомату чи використання наслідування кольору може зайняти значну кількість часу, тому можливість зберегти кольорову конфігурацію може зекономити цей час.

Зберігати таку інформацію можна у базі даних, що дозволить зручно отримувати конфігурації за різними критеріями, такими як назва, час створення, розмірність та правила автомату. Іншим варіантом є збереження стану методом серіалізації. Подібний підхід дозволить створювати зручні в розповсюдженні файли поточної конфігурації, що міститимуть усі необхідні дані. Такі файли можна копіювати на зовнішній носій чи передати мережею Інтернет для розгляду та використання іншими користувачами. Також подібний підхід не потребує використання бази даних, що спростить встановлення застосунку — достатньо лише завантажити *.exe* файл та запустити його. Крім того, використання локальних файлів дозволить використовувати застосунок без підключення до мережі Інтернет.

Завантаження стану потребуватиме коректної обробки даних з серіалізованого файлу. Цього можна досягти шляхом перевизначення масивів, що зберігають тривимірні моделі клітин, з подальшим заданням їх стану та кольору. Також необхідною є можливість переглянути доступні збережені файли станів. У випадку даної програми відображується список доступних у спеціальному каталогу файлів за іменами, вибір одного з яких дозволить завантажити відповідну конфігурацію.

Таким чином застосунок надає повноцінний інструментарій для роботи з тривимірними клітинними автоматами: обрахунок їх еволюції, зручну систему відображення стану автомату, систему кольорової диференціації клітин, спрощену систему визначення переходу до тривіального стану, інструментарій по заданню стану та кольору клітини, а також можливість запису та завантаження стану клітинного автомату разом із правилами.

# Опис програмної реалізації.

## Опис скриптів та їх взаємозв’язку.

### Об’єктна структура програми.

Об’єктна структура застосунку становить з себе множину взаємопов’язаних класів, що у сукупності реалізують усю логіку програми. Більшість класів становить з себе підклас класу MonoBehaviour та використовується у якості скриптів. Ці скрипти закріплюються за певними об’єктами на сцені чи на префабах у якості компонентів, їм задаються певні початкові значення за допомогою редактору Unity, а об’єкти, за якими вони закріплені, коректно налаштовуються та розміщуються на сцені.

Скрипти, що використовуються в даному застосунку, можна розділити на декілька груп. Основною групою є класи, що відповідають за власне обрахунок та відображення тривимірної структури клітинного автомату. До цієї групи, зокрема, відноситься клас, що не є підкласом MonoBehaviour, таким чином не будучи скриптом. Це клас LifeGame. У цьому класі знаходиться уся відділена від графічного зображення та інтерфейсу логіка обрахунку тривимірного клітинного автомату. Клас містить набір значень, що описує правила автомату, повністю зберігає поточний стан автомату, а також надає функціонал з ініціалізації, змінення розміру автомату, підрахунку наступної ітерації, підрахунку кількості сусідів тощо.

Даний клас агрегований класом GameDisplay. Це абстрактний клас, головною задачею якого є відображення структури автомату на екрані та загалом керування цим класом. Цей клас є породженим класом MonoBehaviour, але так як він є абстрактним, то безпосередньо як скрипт він не може використовуватись. Даний клас надає загальний функціонал для всіх форм відображення клітинного автомату, наприклад, інстанціювання екземпляру класу LifeGame на основі переданих до скрипта даних, запис та завантаження поточного стану автомату, керування процесом ітерації автомату та виклик абстрактних методів його відображення.

В програмі на даний момент використовується лише один породжений клас цього скрипта — клас Game3D, що відповідає за тривимірне відображення об’єкту, однак для тестування використовувався клас TestGame, що також є породженим класом GameDisplay. Цей клас відповідав за текстове відображення клітинного автомату та найпростіший функціонал та використовувався для тестування роботи клітинного автомату, а не його зручного відображення.

Клас Game3D є основним скриптом відображення клітинного автомату у якості тривимірної моделі. Він створює тривимірні моделі клітин та контролює їх зовнішній вигляд на основі поточного стану клітинного автомату. Цей скрипт реалізує методи відображення автомату цілком чи лише окремого шару, а також надає широкий спектр інструментів для контролю над зображенням зовнішніми скриптами. Наприклад, він надає інструментарій для змінення прозорості та кольору окремих клітин, взаємодії з автоматом через моделі кубів, що відражають окремі клітини, контролю над анімацією затухання чи появи клітини (як розширення функціоналу зі зміни прозорості) та випадкового задання початкового стану клітинного автомату. Крім того, скрипт містить й всі методи, що він успадкував від класу GameDisplay. Скрипт Game3D являє собою об’єкт, до якого звертаються інші скрипти та елементи інтерфейсу для додаткового керування процесом відображення, наприклад, для призупинення та відновлення ітерації автомату.

Наступною групою скриптів є скрипти, що відповідають за контроль відображення користувачем. Зокрема, за рух камери навколо тривимірної моделі автомату відповідає клас CameraMovement. Він також є підкласом MonoBehaviour і використовується у якості скрипту. Його основною метою є надання користувачу можливості повертати тривимірне зображення за допомогою миші, роздивляючись його структуру з різних сторін. Цей клас отримує інформацію про формат відображення від скрипту Game3D. У випадку відображення лише одного шару клітинного автомату CameraMovement виконує плавний поворот камери до перпендикулярного площині кута. Такий підхід дозволяє користувачеві зрозуміти, з якого саме кута він дивиться на даний шар.

Змінення стану та кольору клітин відбувається у класі CellSelection. Як і попередній клас, він є породженим класом MonoBehaviour і використовується у якості скрипту. Основною метою класу є відслідковування, на яку клітину користувач навів курсор, контроль над анімацією, що помічає цю клітину, підтримку коректної поведінки цієї анімації при переведенні на іншу клітину чи зміні шару, а також можливість обрати одну з заповнених клітин для вибору її кольору. Вибір кольору відбувається за допомогою спеціального плагіну Color Picker, що надає зручний інтерфейс у вигляді палітри та змінює колір обраного об’єкту при його виборі з палітри. Можливість зміни стану клітини чи її кольору, а також анімація поточної позиції курсору працюють лише при відображенні окремого шару автомату. Для зміни стану клітин в об’єкті LifeGame та контролю над відображенням автомату цей скрипт також звертається до скрипту Game3D.

Останньою групою скриптів є такі, що відповідають за взаємодію між інтерфейсом користувача та відображеням клітинного автомату, а також за контроль над самим інтерфейсом. Основними класами цієї групи є класи UIController та ParameterController. Перший клас відповідає за відображення діалогових вікон запису та завантаження файлів. Він зберігає масив усіх контрольованих елементів інтерфейсу та вимикає можливість взаємодії з ними при виведенні одного з діалогових вікон, відновлюючи таку можливість після їх закриття.

Другий клас, ParameterController, відповідає за задання та виведення параметрів автомату. Він контролює взаємодію із полями введення значень розміру автомату, кількості сусідів, необхідної для зародження чи загибелі клітини тощо. При запуску програми чи завантаженні файлу в задачі даного класу входить вивести у текстові поля актуальні значення правил автомату та його розміру. Також цей клас ініціалізує коректну зміну розміру клітинного автомату при зміненні відповідного значення. Обидва даних класи є скриптами, тобто підкласами MonoBehaviour.

Класи SaveState та LoadState відповідають за запис та завантаження об’єктів класу LifeGame відповідно. Вони є проміжним класом для взаємодії між інтерфейсом користувача та класом GameDisplay, що виконує власне серіалізацію та десеріалізацію об’єкта. Клас LoadState також відповідає за виведення списку доступних файлів для завантаження.

Останній клас цієї групи, LayerText, що також є скриптом, контролює виведення номеру поточного шару, який відображається в даний момент, у відповідне поле над повзунком вибору шару. Усі класи даної групи взаємодіють з певними елементами інтерфейсу та, за винятком класу LayerText, із скриптом Game3D.

Для розуміння структури програми, окрім класової структури, необхідно також розглянути ієрархію об’єктів Unity та компоненти цих об’єктів. Але перед цим варто зазначити, які префаби використовуються в даній програмі. Єдиним префабом, що використовується в роботі, є префаб моделі клітини. Він містить компонент-рендерер, що відповідає за відображення тривимірної моделі (у даному випадку — куба), колайдер, що використовується для відстеження наведення на клітину курсору, а також скрипт ObjectColor з плагіну Color Picker, основною метою якого є задання обраного кольору об’єкту.

Кореневими об’єктами ієрархії є чотири об’єкти: EventSystem, ColorPicker, Canvas та GameMaster. Перший об’єкт є стандартним об’єктом для контролю над подіями, зокрема, подіями інтерфейсу користувача. Другий об’єкт, ColorPicker, формує елемент інтерфейсу, що відображає палітру вибору кольору. Цей об’єкт належить до плагіну Color Picker та власне реалізує його функціонал з визначення кольору окремих клітин.

Останні два об’єкти ж мають ширше призначення. Об’єкт GameMaster контролює еволюцію автомату та його відображення, а також містить ряд об’єктів-дітей, що виконують інші функції взаємодії з тривимірною моделлю автомату. Цей об’єкт містить скрипт Game3D, в який передаються базові значення параметрів автомату та посилання на певні елементи інтерфейсу, наприклад, кнопку старту/призупинення еволюції автомату та елементи інтерфейсу, що відповідають за задання шару та формату відображення автомату.

Дітьми GameMaster є два об’єкти. Перший з них — CubeContainer – є контейнером для всіх створюваних моделей кубів, що відражають поточний стан конкретних клітин. Він не має додактових компонентів та розташований в позиції об’єкта-батька. Другий об’єкт, Hinge, є об’єктом, що відповідає за керування відображенням та змінення стану автомату. Він також розташований в тій самій позиції, що й об’єкт батько, та містить два скрипти: CameraMovement та CellSelection. Обидва скрипти мають посилання на об’єкта-батька та задані параметри повернення камери та відображення курсору відповідно. Перший скрипт також має посилання на головну камеру. Головною камерою є об’єкт Main Camera, що є дитиною об’єкта Hinge. Він не містить додаткових компонентів, проте знаходиться на певній дистанції від об’єкта-батька. Камера повернута прямо на об’єкт Hinge. Скрипт CameraMovement здатен наближувати та віддаляти камеру від цього об’єкту, а також повертати цей об’єкт, таким чином надаючи можливість роздивитись тривимірну структуру з різних сторін. Ще одним об’єктом-дитиною Hinge є об’єкт Directional Light, що відповідає за направлене світло, яке надає зображенню об’ємність.

Об’єкт Canvas відповідає за зображення інтерфейсу. Цей об’єкт є стандартним об’єктом-контейнером для елементів інтерфейсу користувача та містить відповідні компоненти — скрипти Canvas, Canvas Scaler та Graphics Raycaster, що необхідні для коректної роботи інтерфейсу користувача. Об’єктом-дитиною Canvas є об’єкт Controller. Він містить компоненти-скрипти UIController та ParameterController, які містять посилання на необхідні елементи інтерфейсу та скрипт Game3D для контролю над ними. Також даний об’єкт містить чотири своїх об’єкти дитини: SaveWindow, LoadWindow, LeftPanel та PightPanel.

SaveWindow та LoadWindow є вимкненими на початку роботи програми. Ці об’єкти є діалоговими вікнами для запису та завантаження стану автомату відповідно. Вони містять відповідно скрипти SaveState та LoadState, за допомогою яких відбувається контроль над записом та завантаженням стану автомату. Дані скрипти містять посилання на скрипт Game3D, а у разі скрипту завантаження також й на скрипт ParameterController задля можливості оновити значення параметрів автомату у відповідних текстових полях. Діалогові вікна містять кнопки відміни, що звертаються до скрипту UIController для закриття вікна, та кнопки підтвердження, що окрім закриття вікна також звертаються до відповідного скрипта задля запису чи завантаження стану.

Об’єкти LeftPanel та RightPanel є лише контейнерами для елементів інтерфейсу, що знаходяться ліворуч та праворуч від тривимірної моделі автомату відповідно. Об’єкт LeftPanel вміщує такі елементи інтерфейсу, як перемикачі PickColor та InheritColors, що відповідають за можливість вибору кольору замість переключення стану та можливість наслідування кольору при підрахунку наступної ітерації; а також текстові поля та написи, що використовуються для задання параметрів автомату. При зміні значень у цих полях викликаються відповідно методу зі скрипту ParameterController, що відтворює зміни в об’єкті класу LifeGame. Перемикачі ж взаємодіють зі скриптами CellSelection та Game3D.

RightPanel вміщує такі елементи інтерфейсу, як кнопки запису та завантаження стану автомату, які звертаються до скрипту UIController для відкриття відповідних діалогових вікон; кнопку запуску/призупинення еволюції автомату, що звертається безпосередньо до скрипту Game3D; повзунок зміни поточного шару, який звертається до скриптів LayerText на відповідному написі та Game3D для власне змінення шару; повзунок зміни швидкості еволюції, що звертається безпосередньо до скрипту Game3D, та відповідний напис; а також список можливих форматів відображення, який при зміненні значення теж звертається до скрипту Game3D, з відповідним написом.

Увесь застосунок працює в одній сцені, без переходів на інші.

### Клас LifeGame

Як зазначалося раніше, клас LifeGame відповідає власне за обрахунок тривимірного клітинного автомату у відриві від способу його відображення. Це серіалізовуваний клас, що означає, що допускається його серіалізація та збереження у файл. Через це клас містить лише серіалізовувані атрибути — базові типи int, float та bool, а також їх масиви. До публічних атрибутів класу відносяться:

* *int size* – числове значення, яке відповідає довжині однієї сторони куба-поля автомату, вказаної в клітинах;
* *int spawnMin* – нижня границя діапазону кількості сусідів, при якому може зародитись клітина;
* *int spawnMax –* верхня границя того ж діапазону;
* *int lonelyRemoveMin* - нижня границя діапазону кількості сусідів, нижче якого клітина загине від самотності на наступній ітерації;
* *int lonelyRemoveMax* — верхня границя того ж діапазону;
* *int overcrowdRemoveMin* - нижня границя діапазону кількості сусідів, вище якого клітина загине від перенаселення на наступній ітерації;
* *int overcrowdRemoveMax* — верхня границя того ж діапазону;
* *bool[,,] field –* тривимірний масив булевих значень, що репрезентує клітинний автомат. Значення окремого поля відповідає стану клітини з відповідними координатами в сітці (*true* для заповненої та *false* для пустої). Масив завжди кубічний та має розмірність *size*;
* *float[,,,] cellColors –* чотиривимірний масив чисел з плаваючою точкою, що зберігає дані про колір кожної клітини. Перші три виміри масиву відповідають координатам клітини у сітці автомату та дорівнюють *size*, а останній завжди дорівнює трьом і відповідає трьом базовим кольорам (червоний, зелений, синій відповідно), значення яких зберігаються для даної клітини.

Клас має один конструктор, що в якості аргументу приймає значення для усіх публічних атрибутів, окрім *field* та *cellColors.* Ці значення присвоюються відповідним публічним атрибутам. Крім того, конструктор викликає метод *Reset.*

Публічний метод *Reset* використовується для ініціалізації масивів даних. Він створює нові масиви для атрибутів *field* та *cellColors* та заповнює їх базовими значеннями (*false* та 1 відповідно). Базове значення масиву *cellColors* відповідає білому кольору. Усі розмірності кожного масиву дорівнюють *size* окрім четвертої розмірності *cellColors* — вона дорівнює трьом для трьох базових кольорів. Метод не повертає ніяких значень та не приймає аргументів.

Метод *Resize* використовується для коректної зміни розміру автомату. Цей метод приймає в якості аргументу новий розмір, що буде записаний у змінну *size,* та не повертає ніякого значення. Даний метод створює новий масив того розміру, що зазначено у аргументі, та переносить в нього всі значення з попереднього, які туди вміщуються. Після цього ці масиви присвоюються як значення атрибутам *field* та *cellColors*, змінюючи таким чином фактичний розмір цих масивів.

Метод *Iterate* використовується для розрахунку наступного стану клітинного автомату. Він приймає в якості аргументу булеве значення, яке позначає, чи необхідно враховувати при підрахунку колір клітини. Спочатку метод створює новий масив та переносить в нього всі поточні значення. Далі значення нового масиву визначаються на основі правил автомату та кількості заповнених сусідів клітини в аналогічному масиві. Кількість сусідів отримується методом *GetNeighborsCount,* а середнє значення кольору сусідів – *GetNeighborsAverageColor.* Це середнє значення присвоюється кольору поточної клітини, якщо значення аргументу становить *true.* Після встановлення усіх значень створені масиви присвоюються у якості значень відповідним атрибутам. Метод повертає булеве значення, що позначає, чи відбулись якісь зміни при переході до нової ітерації.

Метод *Count* використовується для підрахунку кількості заповнених клітин. Він не приймає аргументів та повертає значення кількості клітин. Метод додає до спеціальної змінної одиницю за кожен елемент масиву *field*, що приймає значення *true*. Таким чином остаточна сума дорівнюватиме кількості заповнених клітин, тому саме значення цієї змінної повертається методом.

Метод *GetNeighborsCount* визначає кількість сусідів певної клітини. В якості аргументу він приймає три цілих числа, що позначають координати клітини, для якої обраховуються сусіди. Підрахунок сусідів відбувається за тим самим принципом, що й підрахунок загальної кількості клітин, але враховуються лише клітини у кубі 3х3х3 навколо вказаних координат, за виключенням клітини безпосередньо в цих координатах. Метод повертає кількість заповнених сусідів клітини як цілочисельне значення.

Метод *GetNeighborsAverageColor* слугує для визначення середнього кольору сусідів певної клітини. Як і у випадку попереднього методу, аргументами виступають координати клітини, для якої підраховується середній колір сусідніх клітин. При цьому враховуються лише заповнені клітини. Всі значення сусідніх клітин по кожному основному кольору сумуються та діляться на кількість клітин. Отримане в результаті значення основних кольорів і характеризує середній колір клітини. Метод повертає об’єкт класу Color, що містить визначений колір клітини.

Крім цього, існують додаткові методи для збору статистики, зокрема, метод *GetLifetime*. Він не приймає аргументів та повертає час еволюції системи у ітераціях. Для підрахунку цього значення він викликає метод Iterate доти, доки він не поверне негативне значення (тобто між двома ітераціями не відбулось змін).

### Клас GameDisplay.

Клас GameDisplay є базовим класом скриптів, що відповідають за відображення клітинного автомату. Він містить загальний інструментарій, необхідний всім подібним системам, та вказує декілька абстрактних методів для наслідування. До цих абстрактних методів, зокрема, відносяться *TotalOutput*, що відповідає за виведення стандартного, тривимірного відображення автомату, та метод *LayerOutput*, який відповідає за виведення конкретного шару при конкретному форматі виведення, які передаються до методу в якості аргументу. Обидва абстрактних методи не повертають значень.

Клас GameDisplay містить наступні публічні атрибути:

* *int size –* атрибут, що містить значення розміру клітинного автомату, що буде відображений цим класом;
* *Vector2 spawn –* атрибут, що містить базові мінімальне та максимальне значення кількості сусідів, за яким клітина зароджується;
* *Vector2 lonelyRemove –* атрибут, що містить базові мінімальне та максимальне значення кількості сусідів, нижче якої клітина гине від самотності;
* *Vector2 overcrowdRemove* – атрибут, що містить базові мінімальне та максимальне значення кількості сусідів, вище якої клітина гине від перенаселення;
* *float iteratePeriod –* кількість часу, що проходить між двома ітераціями;
* *int spawnFrequency –* відношення заповнених клітин до їх загальної кількості при заповненні поля випадковим чином;
* *int Layer –* атрибут, що заданий get- та set методами, як й усі наступні в даному класі. Він зберігає поточне значення шару, який відображується при пошаровому форматі, та викликає метод *LayerOutput* при присвоєнні цьому атрибуту нового значення (за умови того, що вибраний пошаровий формат відображення);
* *Orientations View –* атрибут типу Orientations (перелічення, що може приймати значення XYZ, XY, XZ та YZ), що позначає формат відображення автомату. Перший варіант відображає автомат у трьох вимірах, а при присвоєнні цього значення атрибуту викликається метод *TotalOutput,* а інші три варіанти відповідають відображенню одного шару в одній з трьох можливих площин та викликають метод *LayerOutput*;
* *bool IsActive –* атрибут, що здатен лише повертати значення, та показує, чи відбувається в даний момент еволюція автомату;
* *LifeGame LG* – атрибут, що повертає значення об’єкту, що містить інформацію про правила та поточний стан автомату;
* *bool IsInteractable* – атрибут, що показує, чи можна в даний момент взаємодіяти із відображенням автомату;
* *bool InheritColors –* атрибут, що показує, чи необхідно успадковувати колір при заповненні клітини.

Абстрактний клас містить реалізацію методу *Update,* що викликатиметься при кожній зміні кадру. Даний метод відраховує час *iteratePeriod* у разі якщо значення *IsActive* дорівнює *true.* Після вказаної кількості часу викликається метод *Iterate* класу LifeGame з аргументом *InheritColors,* таким чином обраховуючи нову ітерацію автомату з чи без використання наслідування кольору. Після цього викликається метод *TotalOutput* або *LayerOutput*, оновлюючи зображення на екрані відповідно змінам в автоматі.

Клас також надає пусту реалізацію методу *Reset* для наслідування породженими класами.

Метод *Randomize* заповнює певну зону в центрі автомату випадковими значеннями. Розмір зони визначається аргументом функції, а частота появи в зоні заповненої клітини — атрибутом *spawnFrequency*. Після цього, знову ж таки, викликається метод *TotalOutput* або *LayerOutput*, оновлюючи зображення після задання випадкового початкового стану. Метод не повертає ніякого значення.

Методи *SaveGame* та *LoadGame* відповідно записують та завантажують об’єкт класу LifeGame з файлу. Обидва методи не повертають жодного значення та приймають у якості аргументу відносний путь до файлу. Всі файли зберігаються в каталозі, що записаний у стандартній константі *Application.persistentDataPath*. Зазвичай цей шлях вказує на каталог користувача. Обидва методи спочатку відкривають файл, а далі відповідно записують чи зчитують інформацію з нього. Метод *LoadGame*  також викликає метод *Reset* для перезавантаження відображення, а потім *TotalOutput* або *LayerOutput*.

Нарешті, метод *ToggleActivity* не приймає та не повертає жодних значень та змінює значення *IsActivity* на протилежне, призупиняючи чи відновлюючи таким чином еволюцію.

### Клас Game3D.

Породженим класом від абстрактного класу GameDisplay є клас Game3D. Цей клас становить з себе реалізацію GameDisplay, що використовує тривимірні моделі для відображення поточного стану автомату. Він також організує взаємодію з певними елементами інтерфейсу та надає функціонал для інших скриптів, що дозволяє керувати відображенням автомату. У класі містяться наступні публічні атрибути:

* *GameObject startStop* – посилання на кнопку інтерфейсу, що відповідає за запуск та призупинення еволюції автомату;
* *Slider layerSlider –* посилання на скрипт Slider, що контролює повзунок вибору поточного шару для відображення;
* *Dropdown orientationList –* посилання на скрипт Dropdown, що контролює меню вибору формату відображення автомату;
* *GameObject cubePrefab –* посилання на префаб об’єкту, що слугує відображенням окремої клітини;
* *Transform cubeContainer –* посилання на об’єкт-контейнер для усіх створюваних кубів;
* *float alphaBase –* базове значення швидкості анімації, що базується на зміні прозорості куба. Використовується для обрахунку атрибуту *AlphaSpeed;*
* *float AlphaSpeed –* атрибут, що повертає значення швидкості зміни прозорості в анімаціях появи чи зникнення кубів;

Також цей клас містить атрибути, що повертають значення атрибуту *size* локального об’єкту LifeGame та розмір тривимірної моделі з префабу *cubePrefab.*

Клас перевизначає декілька методів батьківського класу. Метод *Reset* є одним з них. Спочатку цей метод знищує масив об’єктів *cubes* та усі об’єкти в ньому, якщо він вже існував. Після цього в цій самій змінній створюється новий масив, що заповнюється створеними з префабу *cubePrefab* значеннями. Для кожного кубу батьком встановлюється об’єкт *cubeContainer,* а значення прозорості встановлюється в нуль. Після цього встановлюється нове максимальне значення повзунка *layerSlider* для вибору шару відображення, а далі викликається метод *TotalOutput* або *LayerOutput* відповідно формату відображення.

Перевизначення методу *Awake* спочатку викликає його реалізації у батьківському класі, після чого методи *Reset* та *Randomize*, ініціалізуючи таким чином відображення автомату та заповнюючи його випадковими початковими значеннями.

Метод *Update* також розпочинає свою роботу з виклику реалізації батьківського класу. В цьому методі відбувається контроль над анімаціями зміни прозорості кубів. Всі куби, що знаходяться у словнику smooth, який співставляє компонент-рендерер з цільовим значенням прозорості, змінюють значення прозорості ближче до значення, зазначеного у словнику, зі швидкістю *AlphaSpeed*. Значення, що досягли визначеної прозорості, видаляються зі словнику. Для кубів, що стали абсолютно прозорими, колір встановлюється в білий.

Крім того, наслідується метод *ToggleActivity.* Він окрім зміни значення *IsActive* також змінює текст на кнопці *startStop* відповідно до дії, яку ця кнопка активуватиме*.* Також додається метод *ToggleInheritColors,* що також змінюватиме значення *InheritColors* на протилежне.

Клас надає реалізації методів *TotalOutput* та *LayerOutput.* Перший заносить у словник *smooth* із значенням 1 чи 0 відповідно стану клітини усі куби, таким чином розпочинаючи анімацію для значень, що змінились, а також встановлює їх колір, взятий з об’єкту LifeGame. Другий робить аналогічне, але лише для одного шару, що визначається форматом відображення *View* та номером шару *Layer.*

Вищезазначені методи використовують певні допоміжні методи. Статичні методи *ChangeAlpha* та *ChangeColor* відповідно змінюють прозорість та колір кубу, чий рендерер передається у якості аргументу. Для першого методу другим аргументом є значення прозорості, а для другого — значення трьох основних кольорів.

Методи *GetStateByCube* та *ChangeStateByCube* використовуються відповідно для зміни та отримання стану клітини, звертаючись до неї за посиланням на куб, що відображує цю клітину. Обидва методи приймають у якості аргументу об’єкт типу Transform відповідного кубу. Метод *GetStateByCube* повертає булеве значення, а метод *ChangeStateByCube* не повертає значень. Пошук необхідного значення виконується порівнянням у циклі.

Наступний метод, *RegisterCubeColor,* використовується для запису кольору кубу в об’єкт LifeGame. Метод приймає в якості аргументу рендерер кубу, який використовується для зчитування кольору, та не повертає значень. Як і у випадку попередніх двох методів, цей метод знаходить необхідну клітину у циклі та вносить до неї інформацію з матеріалу кубу.

Метод *IsCubeInLayer* використовується для визначення, чи знаходиться куб у поточному шарі. Метод також приймає в якості аргументу рендерер кубу, а повертає булеве значення, що показує, чи дійсно цей куб знаходиться у поточному шарі. Коли метод знаходить відповідну кубу клітину, він повертає *true* у разі тривимірного відображення, а в іншому випадку порівнює одну з координат (в залежності від формату відображення) із номером шару. Якщо куб не відповідає жодній клітині, метод повертає значення *false.*

Метод *Clear* ініціалізує анімацію зникнення усіх кубів. Він не приймає аргументів та не повертає значень. Метод перевіряє знаходження усіх кубів у словнику *smooth*. Для тих, які присутні в словнику, метод встановлює значення 0, інші додає до словнику з тим самим значенням.

Нарешті, методи *ChangeLayer* та *SetView* відповідають за зміну значень *Layer* та *View* відповідно. Перше значення зчитується з повзунка *layerSlider,* а друге — зі списку *orientationList.* Обидва методи не приймають аргументів, не повертають значень та викликають приватні методи *applyColliders* та *layerOffset.* Перший приватний метод відповідає за активацію колайдерів на поточному шарі та деактивацію всіх інших колайдерів, що дозволить взаємодіяти лише з активним шаром. Другий метод визначає, де необхідно розмістити *cubeContainer* при відображення даного шару, щоб це відображення завжди знаходилось на одній й тій самій відстані від камери.

Таким чином, скрипт містить широкий набір методів, що можуть використовуватись іншими скриптами чи викликатись за подіями, наприклад, натискання кнопки, та виконує всебічний контроль за відображенням автомату.

### Класи CameraMovement та CellSelection.

Класи CameraMovement та CellSelection відповідають за керування відображенням автомату та взаємодію з ним. Зокрема, CameraMovement надає можливість керувати камерою для розгляду структури автомату з будь-якої сторони. Цей клас підтримує відображення автомату цілком у тривимірній формі чи лише одного шару, в залежності від формату відображення. Керування камерою відбувається за допомогою миші: із затиснутою лівою кнопкою миші її рух повертає тривимірну модель автомату в обраному напрямку, а коліщатко миші здатне наближувати та віддаляти зображення. Даний клас містить наступні публічні атрибути:

* *float rotateSpeed –* визначає швидкість повернення камери під керівництвом користувача чи автоматично під час зміни формату відображення;
* *float zoomSpeed* – визначає швидкість руху камери при її наближенні чи віддаленні коліщатком миші;
* *Transform mainCamera –* посилання на головну камеру сцени, що використовується для змінення її позиції при наближенні чи віддаленні зображення;
* *Game3D game –* посилання на скрипт Game3D, який використовується для визначення формату відображення та іншої інформації.

Клас реалізує метод *Start*, що задає базові значення певних приватних змінних. Метод *Update* також реалізується у цьому класі, і саме в нього закладена логіка керування камерою. Спочатку в методі обраховується максимальна відстань від центру тривимірної моделі автомату до її краю. Далі це значення використається для визначення меж, в яких допускається наближення та віддалення зображення. Наступним кроком власне обраховується зміщення камери на основі руху коліщатка миші та визначених меж. Зміщення забезпечується зміною локальної позиції камери. Далі метод проводить дії в залежності від формату відображення автомату.

У випадку тривимірного відображення першим кроком є перевірка, чи наведений курсор на куб та чи допускається в даний момент взаємодіяти із моделлю. У разі якщо дана умова виконується під час натискання лівої кнопки миші, метод повертає модель клітинного автомату у напрямку руху миші зі швидкістю *rotateSpeed*.

У випадку з одного з форматів відображення окремого шару, метод не дає повертати модель за допомогою миші. Проте метод автоматично повертає модель перпендикулярно напряму камери. Для цього метод поступово повертає модель навколо двох осей одночасно, доки напрямки “вперед” та “вверх” моделі не займуть необхідну для даного формату позицію.

Клас CellSelection, у свою чергу, відповідає за зміну стану клітин та їх кольору. Він підтримує два режими роботи: зміна стану та вибір клітини для зміни кольору. Власне вибір кольору відбувається за допомогою плагіну Color Picker. У першому випадку метод контролює відображення курсору та зміну стану клітини при натисканні лівої кнопки миші. У другому метод керує також відображенням обраної клітини та зміною цього виділення при натисканні на іншу клітину.

Клас CellSelection містить наступні публічні атрибути:

* *float blinkPeriod –* час між двома миготіннями курсору;
* *float blinkDuration –* тривалість одного миготіння курсору;
* *blinkMax –* максимальне значення прозорості при миготінні;
* *blinkMin –* мінімальне значення прозорості при миготінні;
* *Game3D game –* посилання на скрипт Game3D, що використовується для взаємодії з тривимірною моделлю автомату;
* *ColorPicker colorPicker –* посилання на скрипт ColorPicker, що відповідає за контроль над обиранням кольору клітини.

Цей клас також реалізує метод *Start* для ініціалізації певних значень та метод *Update,* що містить основну логіку скрипту. Проте цей клас також містить два додаткових методи *isSelectedMesh* та *isMeshInLayer.* Перший метод перевіряє, чи вибраний даний куб та чи знаходиться скрипт у режимі вибору кольору. Другий метод перевіряє, чи існує куб та чи міститься він у поточному шарі. Для цього викликається метод *IsCubeInLayer* класу Game3D. Обидва методи повертають булеве значення та приймають у якості аргументу рендерер відповідного кубу.

Крім того, клас містить метод *TogglePickColor,* що не приймає аргументів та не повертає значень. Цей метод переключає скрипт між режимами вибору клітини для визначення кольору та зміни стану клітин. При переключенні в режим зміни стану клітин поточна обрана клітина робиться непрозорою, нівелюючи ефект миготіння.

Метод *Update* розпочинає роботу з підрахунку поточної прозорості курсору та обраної клітини. Це значення спочатку росте з часом, а далі знижується до мінімального значення, усі протягом часу *blinkDuration.* Після цього миготіння обраної клітини починається спочатку, а миготіння курсору призупиняється, доки не завершиться час *blinkPeriod.* Наступним кроком визначається, чи наведений курсор на куб. При цьому відслідковується, на який саме куб наведено курсор. Якщо при попередньому виклику методу *Update* куб був інший, то проводяться наступні дії: якщо куб не вибраний та при попередньому виклику методу курсор не був наведений на куб (тобто користувач тільки навів курсор на куб), то його прозорість встановлюється у базове для стану клітини значення; якщо скрипт знаходиться у режимі зміну стану клітин та затиснута ліва кнопка миші — стан поточної клітини змінюється викликом методу *ChangeStateByCube* класу Game3D (тобто при переміщенні курсору на сусідній куб із затиснутою лівою кнопкою миші стан клітини змінюється). Якщо ж курсор залишався на тому ж самому кубі, проте була натиснута ліва кнопка миші, то наступні дії залежать від поточного режиму скрипту.

При режимі вибору клітини для зміни кольору взаємодія відбувається лише із заповненими клітинами. Якщо обраний куб знаходиться у поточному шарі, то при виборі іншого кубу попередній обраний стає непрозорим, нівелюючи ефект миготіння. При натисканні на вже обраний куб, з нього знімається виділення, інакше виділення переводиться на натиснутий куб. При цьому також встановлюється відповідне значення в плагіні Color Picker для підключення обраного кубу до можливості зміни кольору. У режимі ж зміни стану клітини при натисканні лівої кнопки миші відбувається власне зміна її стану.

Нарешті, незалежно від натискання лівої кнопки миші чи переведення курсору на новий куб, відбувається зміна прозорості кубу, на який наведено курсор, реалізуючи анімацію його миготіння. Якщо поточний куб заповнений, то миготіння буде збільшувати прозорість, а якщо він пустий — зменшувати.

Якщо курсор не наведений на будь-який куб, але був наведений при попередньому виклику методу *Update,* прозорість встановлюється в базове значення для нівелювання ефекту миготіння курсору. Незалежно від наведення курсору на куб, якщо обраний куб не знаходиться у поточному шарі чи більше не є заповненою клітиною — виділення з нього знімається. У випадку ж відображення окремого шару у режимі вибору клітини для задання кольору та при неактивній еволюції автомату відбувається миготіння обраного кубу. На цьому виконання методу завершується.

Така складна логіка методу дозволяє контролювати відображення поточної позиції курсору за допомогою зміни прозорості, при цьому коректно повертаючи значення прозорості в базове при переведенні курсору на інший куб чи за межі моделі автомату. Також клас коректно відображає обрану для вибору кольору клітину та змінює вибір при натисканні на інший куб у відповідному режимі.

## Огляд інтерфейсу програми.

Розроблений програмний застосунок використовує графічний інтерфейс користувача. Для реалізації інтерфейсу використовується вбудована в Unity система розробки інтерфейсу, що базується на розміщенні програмістом необхідних елементів безпосередньо на екрані та заданні інтерактивним елементам функцій, що мають викликатись при виникненні певних подій (наприклад, натискання кнопки).

Інтерфейс програми можна поділити на три групи. Першою групою є ліва панель. Вона містить елементи інтерфейсу, що відповідають за можливість змінювати правила переходу до наступної ітерації клітинного автомату, а також за можливість змінювати розмір поля автомату, переключатись між режимами зміни стану клітин та вибору клітини для зміни кольору, та нарешті за можливість успадковувати колір при переході до наступної ітерації.

Другою групою є власне відображення клітинного автомату. Автомат зображується як група тривимірних моделей кубів, за замовчуванням білого кольору, з якими можна взаємодіяти у форматі відображення окремого шару шляхом натискання лівої кнопки миші на окремі куби, або у тривимірному відображенні шляхом руху миші із затиснутою лівою кнопкою миші. У першому випадку натискання кнопки миші призведе до зміни стану клітини чи її вибору для зміни кольору, в залежності від режиму. У другому випадку відбувається повернення зображення у обраному напрямку навколо середини автомату, що моделюється тривимірним зображенням. Крім того, зображення можна віддалити та наблизити коліщатком миші незалежно від режиму чи формату відображення.

Нарешті, третьою панеллю є права панель, що містить ряд кнопок для запису та завантаження конфігурація автомату, запуску а призупинення еволюції автомату, а також повзунки для змінення поточного шару і швидкості еволюції та випадаючий список для вибору формату відображення. Нижче зображено загальний вигляд інтерфейсу програми:



Розглянемо елементи інтерфейсу програми детальніше. У верхньому лівому куті екрану знаходяться два перемикача: Pick Color та Inherit Colors. Обидва перемикача змінюють свої значення при натисканні на них лівою кнопкою миші. Перший відповідає за переключення між двома режимами взаємодії із шаром автомату. Якщо перемикач виключений, то у форматі відображення окремого шару натискання лівої кнопки миші призведе до змінення стану клітини. Якщо ж перемикач Pick Color включений, то натискання лівої кнопки миші призведе до вибору відповідної клітини, у випадку якщо вона є заповненою. Вибрана клітина виділятиметься періодичною зміною прозорості.

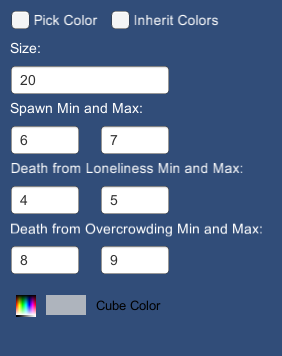
В свою чергу, перемикач Inherit Colors відповідає за використання чи ігнорування можливості успадковувати колір клітин при переході до наступної ітерації. Якщо перемикач виключений, при переході до наступної ітерації колір клітин, що вмирають, залишається незмінним, а нові клітини мають білий чи заздалегідь заданий колір. Якщо ж перемикач включений, клітини, що зароджуються, отримують колір, що приймає середнє значенні між кольорами усіх сусідніх клітин.

Наступним елементом інтерфейсу є поле для введення розміру автомату. Початковим значенням поля є розмір поточного автомату, зокрема довжина сторони поля кубічної форми у клітинах. Значення автоматично оновлюється при завантаженні нової конфігурації з файлу. При безпосередньому зміненні значення в полі автомат коректно змінює свій розмір, додаючи нові чи видаляючи зайві куби. Коректними значеннями поля є додатні цілі числа. Елемент знаходиться відразу під перемикачами.

Нижче знаходяться поля для введення кількості сусідів, необхідних для зародження клітини. Ліве поле використовується для задання мінімального значення, а праве — максимального. При обрахунку нової ітерації для кожної клітини обирається випадкове число з цього проміжку та перевіряється, чи має клітина дану кількість сусідів, в цьому разі клітина заповнюється. Коректними значеннями полів є додатні цілі значення, при цьому мінімальне значення має бути не більше максимального. Рівність значень забезпечить повну визначеність зародження клітин. Як і у випадку полю задання розміру автомату, значення автоматично оновлюється при завантаженні конфігурації з файлу та на початку роботи програму.

Ще нижче знаходяться поля задання кількості сусідів, нижче якої клітина гине від самотності, та кількості сусідів, вище якої вона гине від перенаселення. Принцип їх задання аналогічний заданню умови зародження клітини: ліве поле задає мінімальне значення, а праве — максимальне. Поля також повинні мати тільки цілі додатні значення, та мінімальне значення повинне не перевищувати максимальне. Значення також оновлюється при завантаженні файлу та задається автоматично при запуску програми.

Під цими полями знаходиться поле вибору кольору. Натискання на це поле відкриває палітру, на якій можна обрати певну точку, що відповідатиме обраному кольору. Аналогічно колір можна змінювати шляхом переміщення повзунків, що відповідають за основні кольори. Повзунки автоматично рухаються при виборі точки на палітрі. При закритті вікна обраний колір автоматично присвоюється виділеній клітині, але лише у режимі вибору кольору. Вибір кольору не матиме ніякого ефекту, якщо не виділена жодна клітина.

Нижче зображена уся ліва панель інтерфейсу:

У верхньому правому куті розташовані дві кнопки: Save Configuration та Load Configuration. Натискання цих кнопок відповідно відкриває вікна завантаження та запису поточної конфігурації.

Вікно запису містить поле для введення назви файлу та дві кнопки під ним. Ліва кнопка, Save, записує поточну конфігурацію в файл із назвою у полі введення. Поле повинне містити непустий рядок. Після запису вікно закривається. Права кнопка, Cancel, закриває вікно без збереження. Поки вікно відкрито, інші елементи інтерфейсу неактивні, а можливість взаємодії з моделлю автомату відсутня. Вікно завантаження аналогічне вікну збереження, має ті ж властивості та функціонал кнопок, проте замість поля введення назви файлу містить випадаючий список доступних для завантаження файлів. Для коректного завантаження файлу натиском кнопки Load необхідно обрати один з файлів. Список автоматично оновлюється при кожному відкритті вікна. Зображення вікон запису та завантаження розташовані нижче (запис з лівої сторони, завантаження — з правої).

