АНОТАЦІЯ

Кваліфікаційна робота на тему «Проектування та розробка розподіленої системи на основі моделі клітинного автомату» написана Босяком Валерієм, студентом Тернопільського Національного Технічного Університету імені Івана Пулюя, факультет комп’ютерно-інформаційних систем та програмної інженерії, група СПм-61.

Відомості про обсяг: сторінок – __, рисунків – 14, таблиць – 0, частин – 4, додатків – 3, посилань – 13.

Метою кваліфікаційної роботи є проектування та розробка розподіленої системи на основі клітинного автомата Д. Конвея «Гра життя» з використанням мови програмування Scala та екосистеми Akka для побудови складних реактивних та розподілених систем.

Отриманими результатами є проект який можна використовувати як зразок при проектуванні складних розподілених систем або кластерів та клітинних автоматів. Також отримані результати мають практичну та наукову цінність для демонстрації роботи клітинних автоматів.

Ключові слова: РЕАКТИВНІСТЬ, РОЗПОДІЛЕНІСТЬ, ВІДМОВОСТІЙКІСТЬ, КЛАСТЕР, КЛІЄНТ, СЕРВЕР, АРХІТЕКТУРА, АКТОР, AKKA, SCALA, GODOT, КЛІТИННИЙ АВТОМАТ, ГРА ЖИТТЯ, КОНВЕЙ.

SUMMARY

The master's thesis on "Design and development of distributed system based on cellular automaton model" was written by Bosiak Valerii, a student of Ternopil National Technical University named after Ivan Pulyuy, Faculty of Computer Information Systems and Software Engineering, SPm-61.

Volume information: pages – __, Figures – 14, tables – 0, parts – 4, appendices – 3, links – 13.

The aim of the master's thesis is to design and develop a distributed system based on the D. Conway cellular automaton "Game of life" using the Scala programming language and the Akka ecosystem for building complex reactive and distributed systems.

The results obtained are a project that can be used as a model in the design of complex distributed systems or clusters and cellular automata. The results obtained are also of practical and scientific value for demonstrating the operation of cellular automata.

Keywords: REACTIVITY, DISTRIBUTION, FAULT TOLERANCE, CLUSTER, CLIENT, SERVER, ARCHITECTURE, ACTOR, AKKA, SCALA, GODOT, CELLULAR AUTOMATON, GAME OF LIFE, CONWAY.

ЗМІСТ

ВСТУП 7
1 АНАЛІТИЧНА ЧАСТИНА 9
1.1 Опис сфери застосування розподілених систем 9
1.2 Опис сфери застосування клітинних автоматів 12
1.3 Постановка задач розробки 13
1.4 Вибір технології розробки 14
2 ТЕОРЕТИЧНА ЧАСТИНА 18
2.1 Мікросервісна архітектура 18
2.2 Архітектура цільової системи 20
3 ПРАКТИЧНА ЧАСТИНА 23
3.1 Середовище та інструменти розробки кластера 23
3.2 Розробка кластерної частини 24
3.3 Розробка клієнта 28
3.4 Unit тестування 32
3.5 Інтеграційне тестування 32
4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 33
4.1 Охорона праці 33
4.2 Безпека в надзвичайних ситуація 36
ВИСНОВКИ 40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 41
ДОДАТКИ 42

ВСТУП

Сучасний світ важко уявити без складних розподілених інформаційних систем. Сьогодні вже не є можливим використання всюди монолітної архітектури. Інформаційний простір переповнюють петабайти даних, які треба обробляти та зберігати. Необхідно постійно масштабувати та розвивати інформаційні системи, щоб обробляти великі потоки даних.

Проектування та розробка розподілених систем це відповідальна та складна задача. Помилка на етапі проектування може призвести до значних фінансових та часових втрат в майбутньому. Необхідно підбирати правильні архітектуру та технології, досконало розбиратись в бізнес домені та технічній складовій.

Як приклад складної розподіленої системи, можна виділити систему призначену для обробки клітинного автомату. Одним з найпопулярніших клітинних автоматів є «Гра життя» Д. Конвея [1].

Клітинний автомат – дискретна модель обчислень, що вивчається в теорії автоматів. Клітинні автомати також називаються клітинними просторами, автоматами тесселяції, однорідними структурами, клітинними структурами, структурами тесселяції і ітераційними масивами. Клітинні автомати знайшли застосування в різних областях, включаючи фізику, теоретичну біологію і моделювання мікроструктур.

Мета розподіленої системи - розподілити періодичну решітку між членами кластеру та обчислювати наступні ітерації клітинного автомату. Важливими аспектами системи є узгодженість та коректність. Всі члени кластеру мають бути узгоджені між собою, щоб коректно обчислювати кожну ітерацію.

Для реалізації такої системи використовується мультипарадигмова мова програмування Scala з набором бібліотек [2]. З останніх варто виділити Akka фреймворк, спрямований на побудову відмовостійких, розподілених систем на основі системи акторів. Akka - це безкоштовний набір інструментів з відкритим вихідним кодом і середовище виконання, що спрощує конструювання

розподілених додатків на JVM. Akka підтримує декілька моделей програмування для паралельності, але підкреслює паралелізм на основі акторів [3].

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Опис сфери застосування розподілених систем

Розподілені обчислення – це область комп'ютерних наук, яка вивчає розподілені системи [4]. Розподілена система – це система, компоненти якої розташовані на віддалених комп'ютерних системах, які взаємодіють і координують свої дії, передаючи повідомлення один одному. Компоненти взаємодіють один з одним для досягнення спільної мети.

Трьома істотними характеристиками розподілених систем є: паралелізм компонентів, відсутність глобального годинника і ізольованість від відмов компонентів. Така система має володіти певною реактивністю, яка полягає в тому, що, коли компоненти системи виходять з ладу, це не означає, що вся система виходить з ладу. Приклади розподілених систем варіюються від систем на основі SOA до багатокористувацьких онлайн-ігор і peer-to-peer додатків.

Комп'ютерна програма, яка виконується в розподіленій системі, називається розподіленою програмою (а розподілене програмування - це процес написання таких програм). Існує безліч різних типів реалізацій механізму передачі повідомлень, включаючи чистий HTTP, RPC-подібні з'єднувачі і message queues. Розподілені обчислення також відносяться до використання розподілених систем для вирішення обчислювальних специфічних завдань. У розподілених обчисленнях завдання ділиться на безліч менших, кожна з яких вирішується одним або декількома комп'ютерами, які взаємодіють один з одним за допомогою передачі повідомлень.

Для розподілених обчислень використовуються різні апаратні та програмні архітектури. На більш низькому рівні необхідно з'єднати кілька процесорів будь-якої мережею, незалежно від того, реалізована ця мережа на друкованій платі або складається з слабо пов'язаних пристроїв і кабелів. На більш високому рівні

необхідно з'єднати процеси, запущені на цих процесорах, будь-якою системою зв'язку. Розподілене програмування зазвичай відноситься до однієї з декількох базових архітектур: клієнт-сервер, трирівнева, n-рівнева або однорангова; або категорії: слабко-зв'язні або сильно-зв'язні системи.

- Клієнт-сервер: архітектури, в яких клієнти зв'язуються з сервером для отримання даних, потім форматують і відображають їх користувачам. Зміни на клієнті передаються назад на сервер.
- Трирівневі: архітектури, які переміщують клієнтський додаток на середній рівень, щоб можна було використовувати клієнтів без стану (stateless). Це значно спрощує розгортання додатків. Більшість веб-додатків є трирівневими.
- N-рівневі: архітектури, які зазвичай відносяться до веб-додатків, які в подальшому перенаправляють свої запити іншим корпоративним службам. Цей тип додатків є найбільш поширеним в серверній архітектурі.
- Peer-to-peer: архітектури, в яких немає спеціальних машин, що надають послуги або керуючих мережевими ресурсами. Замість цього всі обов'язки рівномірно розподілені між усіма машинами (одноранговими). Однорангові вузли можуть служити як клієнтами, так і серверами. Приклади цієї архітектури включають BitTorrent і мережу Bitcoin.

Іншим основним аспектом архітектури розподілених обчислень є спосіб зв'язку і координації роботи між паралельними процесами. За допомогою різних протоколів передачі даних процеси можуть безпосередньо взаємодіяти один з одним, як правило, у відносинах "ведучий-підлеглий". Альтернативно, орієнтована на базу даних архітектура може дозволити виконувати розподілені обчислення без будь-якої форми прямого міжпроцесного зв'язку, використовуючи загальну базу даних. Архітектура, орієнтована на базу даних, зокрема, забезпечує аналітику реляційної обробки в схематичній архітектурі, що дозволяє ретранслювати живе середовище. Це дозволяє виконувати функції розподілених обчислень як в межах, так і за межами параметрів мережевої бази даних.

Причини використання розподілених систем і розподілених обчислень можуть включати:

- Сама природа програми може вимагати використання комунікаційної мережі, що з'єднує кілька комп'ютерів: наприклад, дані, отримані в одному фізичному місцезнаходження і необхідні в іншому місцезнаходження.
- Існує багато випадків, коли використання одного комп'ютера було б в принципі можливо, але використання розподіленої системи вигідно з практичних міркувань. Наприклад, може бути більш економічним отримати бажаний рівень продуктивності за рахунок використання кластера з декількох комп'ютерів низького класу в порівнянні з одним комп'ютером високого класу. Розподілена система може забезпечити більшу надійність, ніж не розподілена система, оскільки не існує єдиної точки відмови. Більш того, розподілена система може бути простіше в масштабуванні і управлінні, ніж монолітна система.

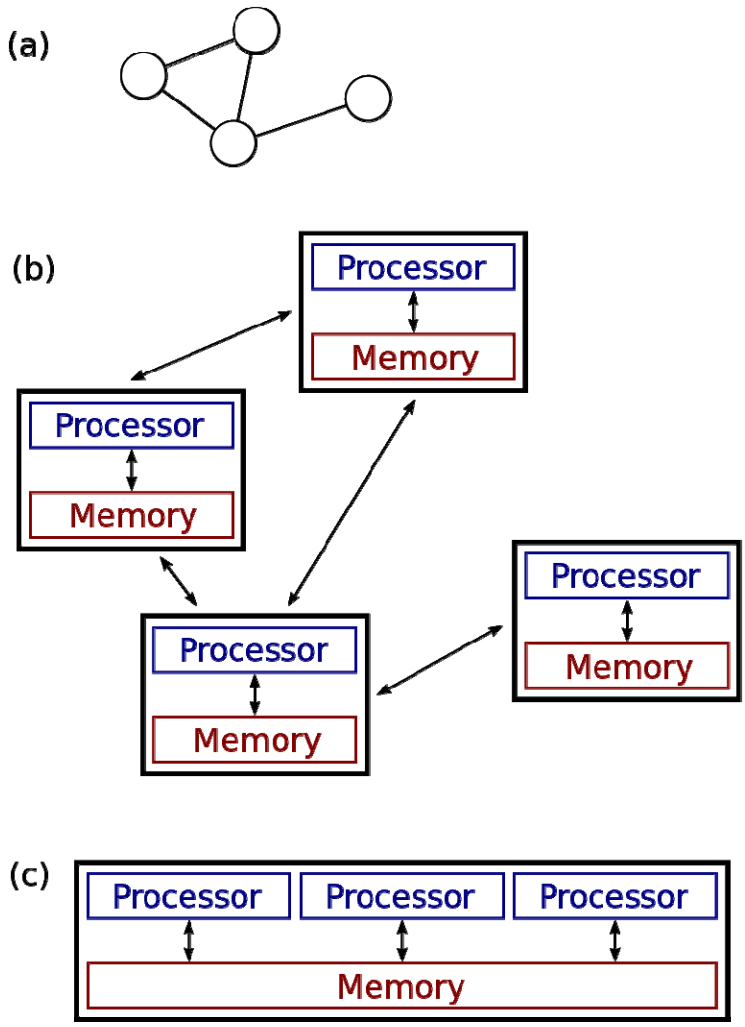

Рисунок 1.1.1 – a), b) – розподілена система, c) паралельна система

1.2 Опис сфери застосування клітинних автоматів

Клітинні автомати знайшли застосування в різних областях, включаючи фізику, теоретичну біологію і моделювання мікроструктур [5]. «Гра життя», також відома просто як життя, являє собою клітинний автомат, розроблений британським математиком Джоном Хортоном Конвеєм в 1970 році [6]. Це гра з нульовим гравцем, що означає, що її еволюція визначається її початковим станом, що не вимагає додаткового введення від користувача. Людина взаємодіє з грою, створюючи початкову конфігурацію і спостерігаючи, як вона розвивається.

Модель є повною по Тюрінгу і може імітувати універсальний конструктор або будь-яку іншу машину Тюрінга.

Всесвіт гри життя являє собою нескінченну двовимірну ортогональну сітку квадратних осередків, кожна з яких знаходиться в одному з двох можливих станів, живому або мертвому (або населеному і незаселеному відповідно). Кожна клітина взаємодіє зі своїми вісьмома сусідами, які є сусідніми осередками по горизонталі, вертикалі або діагоналі. На кожному кроці в часі відбуваються наступні переходи:

1. Будь-яка жива клітина з менш ніж двома живими сусідами вмирає.
2. Будь-яка жива клітина з двома або трьома живими сусідами живе до наступного покоління
3. Будь-яка жива клітина з більш ніж трьома живими сусідами вмирає (через перенаселення)
4. Будь-яка мертва клітина з рівно трьома живими сусідами стає живою клітиною (розмноження)

Початковий патерн становить зародок системи. Перше покоління створюється шляхом одночасного застосування вищевказаних правил до кожної клітини, живої або мертвої; народження і смерть відбуваються одночасно, і дискретний момент, в який це відбувається, іноді називають тактом або ітерацією. Кожне покоління є чистою функцією попереднього. Правила продовжують застосовуватися неодноразово для створення наступних поколінь.

1.3 Постановка задач розробки

Основною задачею розробки було обрано розробку розподіленої системи, яка б використовував принципи реактивної архітектури та реактивного програмування на прикладі клітинного автомата Д. Конвея «Життя». Розробка має

відбуватись з допомогою мови програмування Scala та екосистеми бібліотек Akka. Для зручного керування системою має бути розроблений клієнт з використанням технології Godot [7].

1.4 Вибір технології розробки

Scala – це статично типізована мова програмування загального призначення, яка підтримує як об'єктно-орієнтоване програмування, так і функціональне програмування. Розроблена, з лаконічним синтаксисом. Багато дизайнерських рішень Scala спрямовані на усунення недоліків Java. Вихідний код Scala може бути скомпільований в байт-код Java і запущений на віртуальній машині Java (JVM). Scala забезпечує сумісність з Java, так що на бібліотеки, написані на будь-якій мові, можна посилатися безпосередньо в коді Scala або Java. Як і Java, Scala є об'єктно-орієнтованою і використовує C синтаксис. Починаючи з Scala 3, існує також можливість використовувати правило off-side (відступ) для структурування блоків, і його використання рекомендується. Мартін Одерський сказав, що це виявилося найпродуктивнішою зміною, внесеною в Scala 3 [8].

На відміну від Java, Scala має безліч функцій функціональних мов програмування, таких як Scheme, Standard ML і Haskell, включаючи каррування, незмінність, ліниві обчислення і зіставлення зі зразком. Scala також має вдосконалену систему типів, що підтримує алгебраїчні типи даних, коваріацію та контраваріантність, типи вищого порядку та анонімні типи. Інші функції Scala, відсутні в Java, включають перевантаження операторів, необов'язкові параметри, іменовані параметри та необроблені рядки.

Akka - це безкоштовний набір інструментів з відкритим вихідним кодом і середовище виконання, що спрощує конструювання розподілених додатків на JVM. Akka підтримує декілька моделей програмування для паралельності, але

підкреслює паралельність на основі акторів [9]. Мовні прив'язки існують як для Java, так і для Scala. Akka написана на мові Scala, і, починаючи з версії Scala 2.10, актори в стандартній бібліотеці Scala застаріли на користь Akka.

Ключовими моментами, що відрізняють програми, засновані на акторах Akka, є:

- Паралелізм заснований на повідомленнях і асинхронності: зазвичай не використовуються змінні дані і не використовуються примітиви синхронізації.
- Спосіб взаємодії учасників однаковий незалежно від того, знаходяться вони на одному хості або на різних хостах, взаємодіють безпосередньо або через засоби маршрутизації, працюють в декількох потоках або в багатьох потоках і т. д. Такі деталі можуть бути змінені під час розгортання, що дозволяє горизонтально масштабувати систему.
- Актори розташовані ієрархічно щодо збоїв програми, які розглядаються як події, що підлягають обробці батьком актора (незалежно від того, який актор відправив повідомлення, що викликало збій). На відміну від Erlang, Akka забезпечує батьківський нагляд, що означає, що кожен актор створюється і контролюється своїм батьківським актором.

Akka має модульну структуру з основним модулем, що забезпечує акторів. Інші модулі доступні для додавання таких функцій, як розподілення акторів, підтримка кластерів, Event Sourcing, інтеграція з різними сторонніми системами (наприклад, Apache Camel, ZeroMQ) і навіть підтримка інших моделей паралелізму, таких як ф'ючерси і агенти.

Мікросервіси можуть створювати багато нових проблем, і це не єдиний спосіб створювати інформаційні системи. Традиційна розподілена система може мати меншу складність і добре працювати в багатьох випадках. Наприклад, для невеликого стартапу з однією командою, створення програми, де час виходу на ринок – це все. Akka Cluster можна ефективно використовувати для створення такого розподіленого додатка.

У цьому випадку існує єдиний блок розгортання, побудований на основі єдиної кодової бази (або з використанням традиційного управління бінарними залежностями для модульності), але розгорнутий на багатьох вузлах за допомогою одного кластера. Більш щільне зчеплення – це нормально, оскільки є центральний пункт розгортання та контролю. У деяких випадках вузли можуть мати спеціалізовані ролі під час виконання, що означає, що кластер не є повністю однорідним (наприклад, вузли «фронтенду» та «бекенду» або виділені головні/робочі вузли), але вони запускаються з одного артефакту і це не викликає таких же проблем, які можна отримати від розгортання абсолютно окремих артефактів.

Тісно зв’язні системи протягом багатьох років добре служили індустрії і все ще залишаються хорошим вибором.

Мета Akka Streams полягає в тому, щоб запропонувати інтуїтивно зрозумілий та безпечний спосіб сформулювати налаштування обробки потоків, щоб мати змогу виконувати їх ефективно та з обмеженим використанням ресурсів (без OutOfMemoryErrors). Щоб досягти цього, потоки мають можливість обмежити буферизацію, яку вони використовують, уповільнити продюсерів, якщо споживачі не справляються. Ця функція називається зворотним тиском і лежить в основі ініціативи Reactive Streams, засновником якої є Akka. Akka Streams безперебійно взаємодіє з усіма іншими реалізаціями Reactive Streams (де інтерфейси Reactive Streams визначають інтероперабельність SPI, а такі реалізації, як Akka Streams, пропонують користувальницький API).

API Akka Streams повністю відокремлений від інтерфейсів Reactive Streams. Хоча Akka Streams зосереджується на формулюванні перетворень у потоках даних, сфера дії Reactive Streams полягає у визначенні загального механізму переміщення даних через асинхронну межу без втрат, буферизації або вичерпання ресурсів.

Зв’язок між цими двома полягає в тому, що API Akka Streams орієнтований на кінцевих користувачів, тоді як реалізація Akka Streams використовує інтерфейси Reactive Streams для передачі даних між різними операторами. З цієї

причини не можна знайти жодної подібності між інтерфейсами Reactive Streams та Akka Streams API. Це відповідає очікуванням проекту Reactive Streams, основною метою якого є визначення інтерфейсів, щоб різні реалізації потокової передачі могли взаємодіяти; Реактивні потоки не призначені для опису API кінцевого користувача.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Мікросервісна архітектура

Мікросервісна архітектура – варіант сервіс-орієнтованої архітектури (SOA) – організовує додаток як набір слабо пов'язаних сервісів [10]. Мета полягає в тому, щоб окремі команди розробників могли реалізовувати вимоги незалежно від інших. Слабка зв’язність зменшує всі типи залежностей і пов'язані з цим складності, оскільки розробникам сервісів не потрібно піклуватися про користувачів сервісу доки вони не розгорнуть свої зміни. Таким чином, це дозволяє організаціям, що розробляють програмне забезпечення, швидко зростати і розширюватися, а також простіше використовувати готові сервіси. Вимоги до зв'язку менше. Інтерфейси повинні бути ретельно розроблені і розглядатися як загальнодоступний API. Такі методи, як наявність декількох інтерфейсів в одній і тій же службі або декількох версій однієї і тієї ж служби, дозволяють не порушувати існуючий код користувачів.

Для мікросервісів не існує єдиного визначення. З плином часу в галузі сформувалася єдина думка. Деякі з часто згадуваних визначальних характеристик включають:

- Служби в архітектурі мікросервісів часто являють собою процеси, які взаємодіють по мережі для досягнення мети з використанням не залежних від технології протоколів, таких як HTTP.
- Послуги організовані з урахуванням бізнес-вимог.
- Сервіси можуть бути реалізовані з використанням різних мов програмування, баз даних, апаратних і програмних середовищ, в залежності від того, що підходить найкраще.
- Сервіси невеликі за розміром, підтримують обмін повідомленнями, обмежені контекстами, автономно розробляються, незалежно

розгортаються, децентралізовані, побудовані і випущені з автоматизованими процесами.

Мікросервіси володіють багатьма привабливими властивостями, як-от незалежність мікросервісів дає змогу створювати декілька менших та більш зосереджених команд, які можуть частіше надавати нові функції та швидше реагувати на можливості бізнесу. Реактивні мікросервіси повинні бути ізольованими, автономними та нести єдину відповідальність.

В архітектурі мікросервісів слід розглянути комунікацію всередині служби та між службами.

Загалом не рекомендується використовувати Akka Cluster та обмін повідомленнями між різними службами, оскільки це призведе до занадто тісно-зв'язного коду між службами та труднощів із розгортанням їх незалежно один від одного, що є однією з основних причин використання архітектури мікросервісів.

Вузли однієї системи (сукупно називаються кластером) вимагають меншої зв'язності. Вони мають один і той же код і розгортаються разом, як набір, однією командою або окремою особою. Під час постійного розгортання можуть одночасно працювати дві версії, але розгортання всього набору має єдину точку контролю. З цієї причини комунікація всередині системи може використовувати переваги Akka Cluster, керування збоями та обмін повідомленнями акторів, які зручні у використанні та мають високу продуктивність.

Між різними системами Akka HTTP або Akka gRPC можна використовувати для синхронного (але неблокуючого) зв'язку, а Akka Streams Kafka або інші з'єднувачі Alpakka для інтеграції асинхронного зв'язку. Усі ці механізми зв'язку добре працюють із потоковою передачею повідомлень із наскрізним зворотним тиском, а інструменти синхронного зв'язку також можна використовувати для взаємодії з відповіддю на один запит. Важливо також зазначити, що під час використання цих інструментів обидві сторони комунікації не повинні бути реалізовані за допомогою Akka, а також мова програмування не має значення.

2.2 Архітектура цільової системи

Для розробки цільової системи добре підходить, вище загадана, мікросервісна архітектура, оскільки ціль системи полягає в розподілі великої задачі на менші та їх незалежне опрацювання.

Для координації роботи в кластері потрібна керуюча ланка, яка буде координувати роботу у всьому кластері та служити сервером для клієнтських запитів. В подальшому, керуюча ланка буде носити назву – Мастер (Master). Мастер буде відповідальним за обробку таких клієнтських запитів, як створення нової задачі, статус кластера та його характеристики. Управління вже запущеною задачею.

Обробка клітинного автомату буде здійснюватися окремими членами кластеру – робітниками (workers). Мастер має прямий зв’язок з кожним робітником і відповідальний за поділ клієнтської задачі між робітниками.

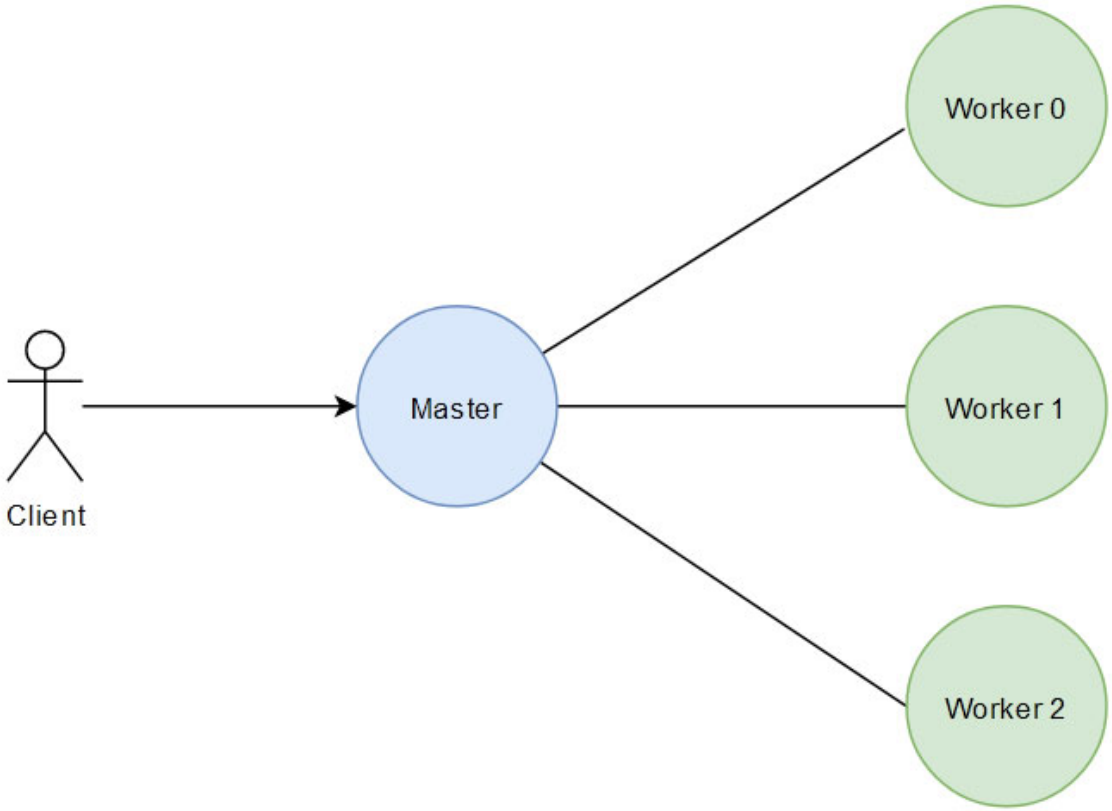

Рисунок 2.2.1 – архітектура системи

Користувач має можливість вказати розмір поля клітинного автомату для подальшого обчислення. Поділ поля як і його обрахунок прозорий для користувача. Якщо розмір заданого поля достатньо малий – його опрацюванням може зайнятись лише один робітник. Цей режим називається Stand-alone, тобто коли всі обчислення проводяться на одному робітнику, а всі інші простоюють. У випадку коли поле достатньо велике для одного робітника, Master має поділити поле між робітниками або повідомити користувачу про неможливість виконання задачі з даними параметрами.

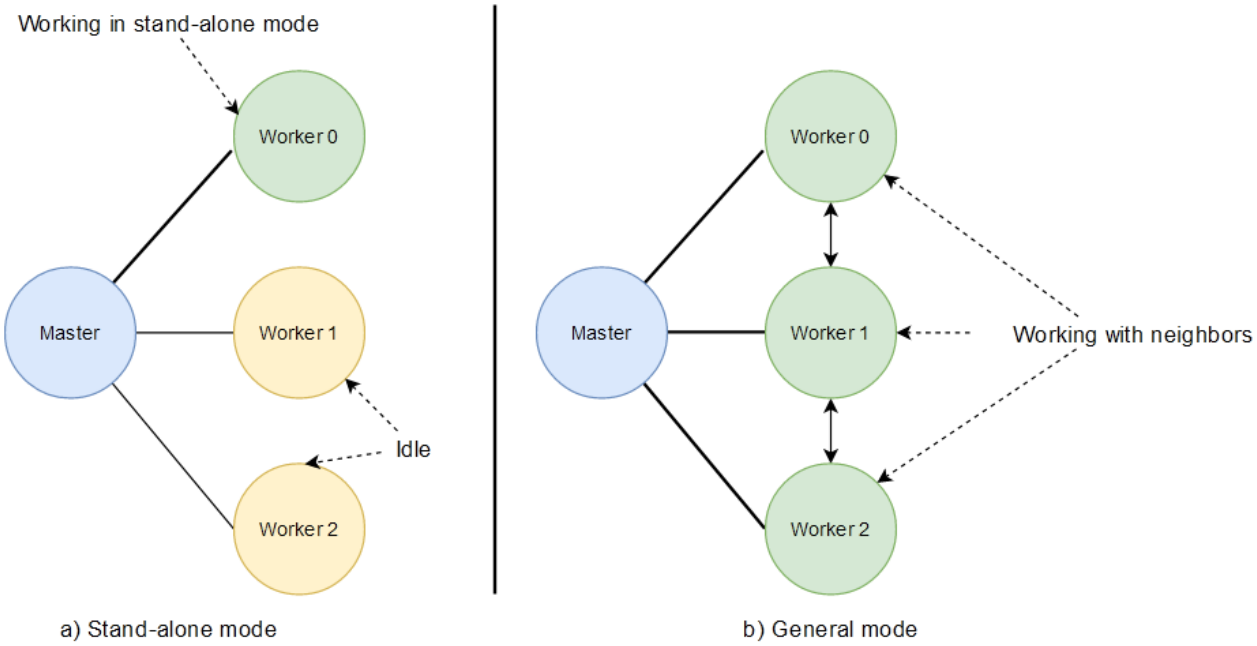

Рисунок 2.2.2 – режими роботи кластеру

Розподіл поля клітинного автомату відбувається на початковому етапі створення задачі. Поле ділиться жадібним алгоритмом по вертикалі. Нижче наведено приклад поділу поля розміром 10 на 20 між двома робітниками. Для демонстрації, ресурсами кожного робітника знехтувано.

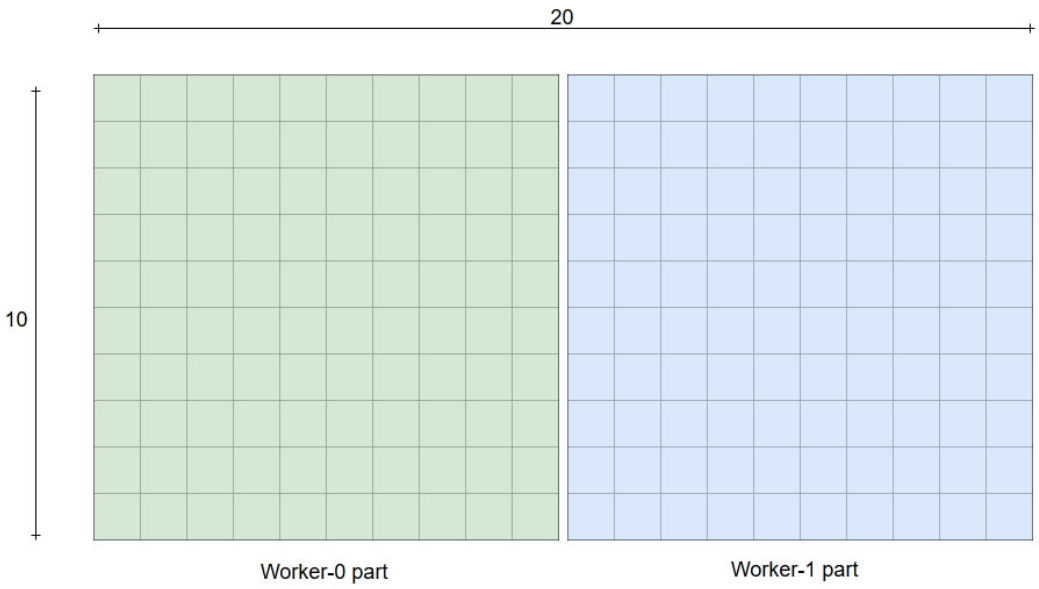

Рисунок 2.2.3 – поділ решітки клітинного автомату

В канонічній версії клітинного автомату Д. Конвея, простір є безкінечним. Тобто кожна сторона замкнута на протилежну. Така геометрична фігура називається тор.

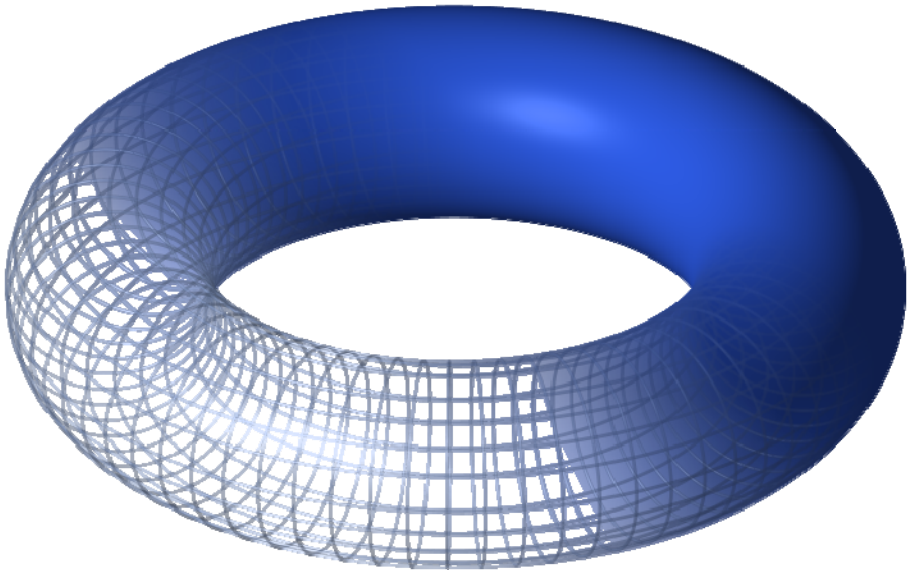

Рисунок 2.2.4 – тор

3 ПРАКТИЧНА ЧАСТИНА

3.1 Середовище та інструменти розробки кластера

Існує декілька середовищ розробки для мови програмування Scala:

- JetBrains IntelliJ IDEA + Scala plugin
- Neovim або Visual Code + Metals plugin
- Scala IDE (застаріле)

Для виконання кваліфікаційної роботи було вибрано IntelliJ IDEA через великий досвід роботи з даним середовищем [11].

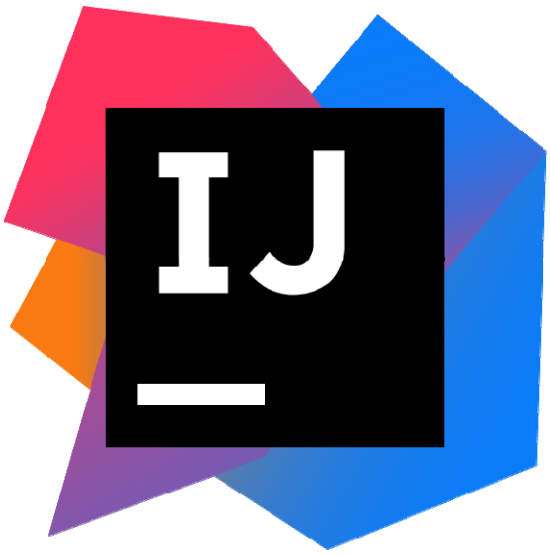

Рисунок 3.1.1 – IntelliJ IDEA

Розробка на Scala передбачає використання додаткового інструментарію окрім середовища розробки. Для компіляції, збірки, запуску та тестування системи використовується інструмент Mill [12]. Для форматування кодової бази згідно налаштованих правил використовується scalafmt. Система контролю версій – Git, віддалений репозиторій – GitHub. Всі перелічені інструменти інтегруються з основним середовищем розробки.

Для кластерної частини використовується одна кодова база. Умовне розділення по пакетам між робітником і мастером присутнє.

3.2 Розробка кластерної частини

Для реалізації роботи кластеру використовується Akka Cluster модуль з екосистеми Akka. Даний модуль дозволяє налаштувати роботу кластера, ролі машин та моніторинг. Для цільової архітектури важливо постійно вести спостереження за станом кожної машини в кластері і реагувати на зміни. Дана поведінка основа реактивних систем. Інша властивість реактивності – асинхронність. Мастер має буди доступний для клієнта на любому етапі життя.

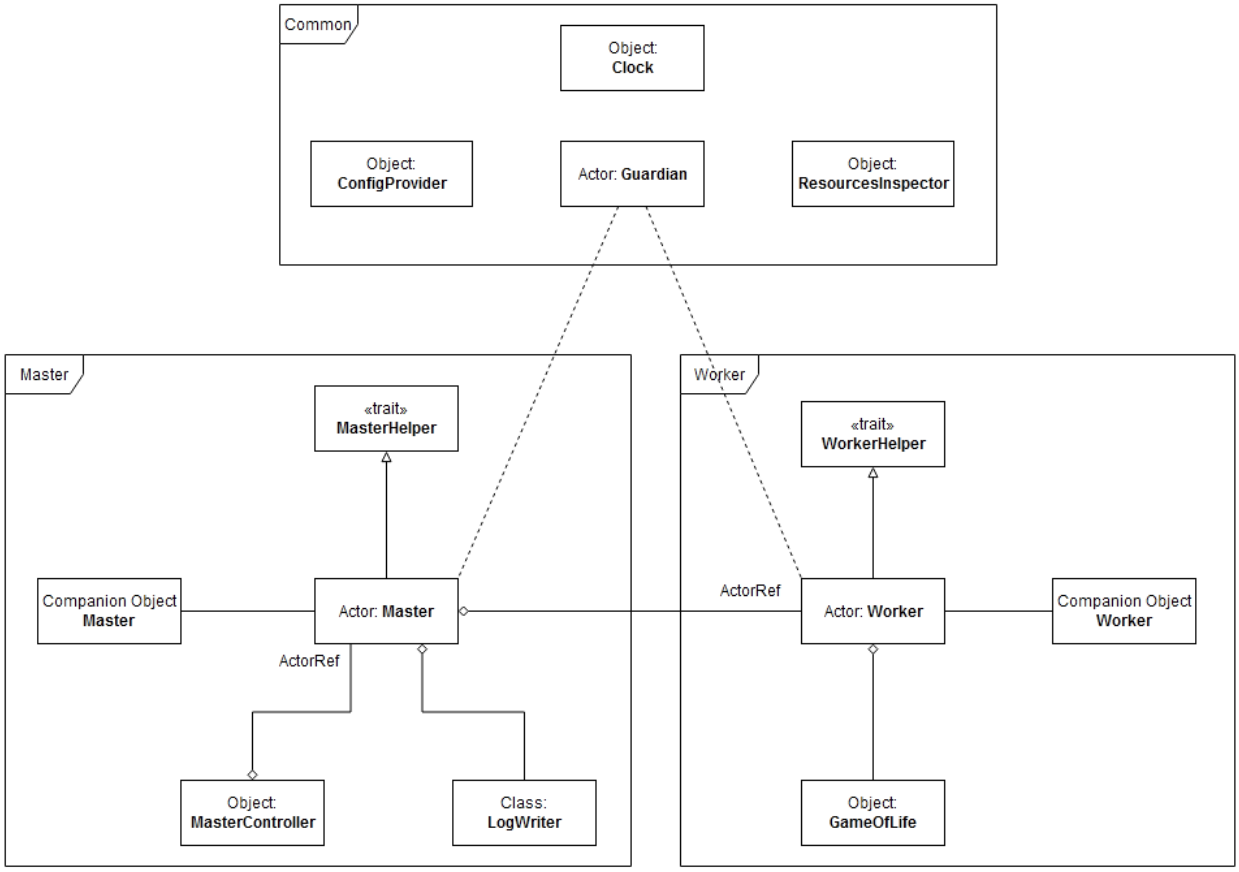

Рисунок 3.2.1 – дизайн кластерної частини

Актори в системі акторів розташовуються ієрархічно. Не залежно від реалізації, в кожній системі акторів має бути корінний актор (root або guardian actor). Далі визначаються актори нащадки. В залежності від вхідних параметрів, корінний актор буде мати, або Master, або Worker актора як нащадка. Починаючи від версії Akka 2.16 – кожний актор має оброблювати помилки самостійно, тобто не звертаючись до батьківського актора.

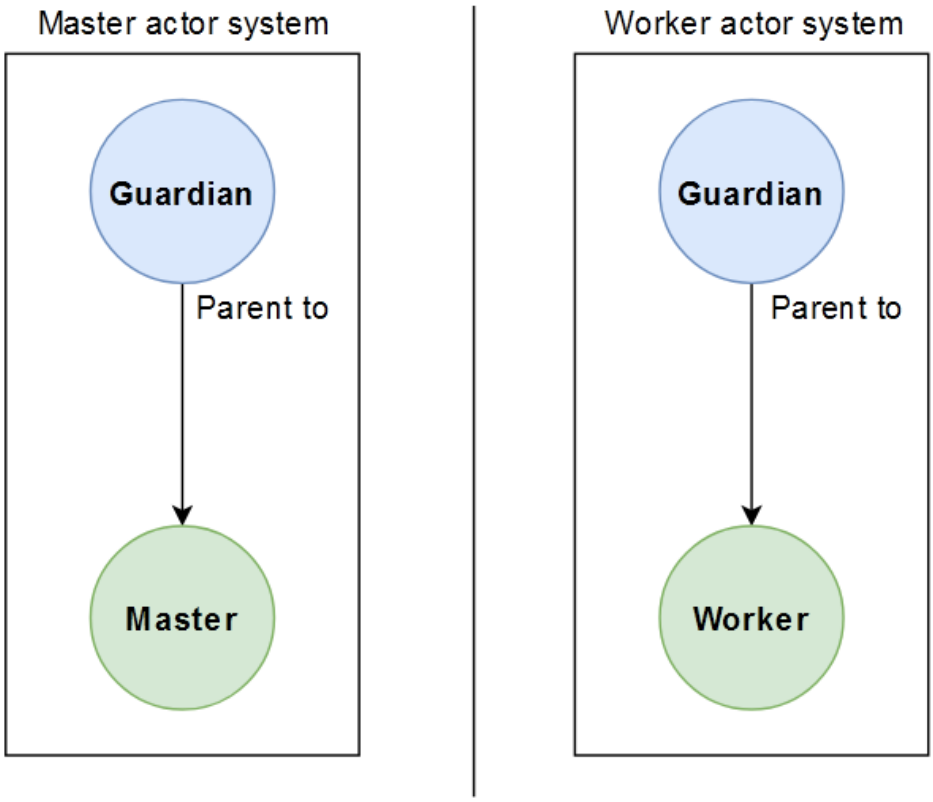

Рисунок 3.2.2 – ієрархія акторів

Для збереження частини решітки клітинного автомату використовується двовимірна матриця з булевими значеннями. Кожне значення позначає стан комірки – жива або мертва. Така реалізація є простою в обрахунку та детермінованою по часу та простору. Знаючи розмір матриці, можна визначати простір необхідний для її збереження і приблизний час обчислення наступної ітерації. Нижче наведено схематичний приклад обчислення матриці розміром 10 на 10 для деякого робітника.

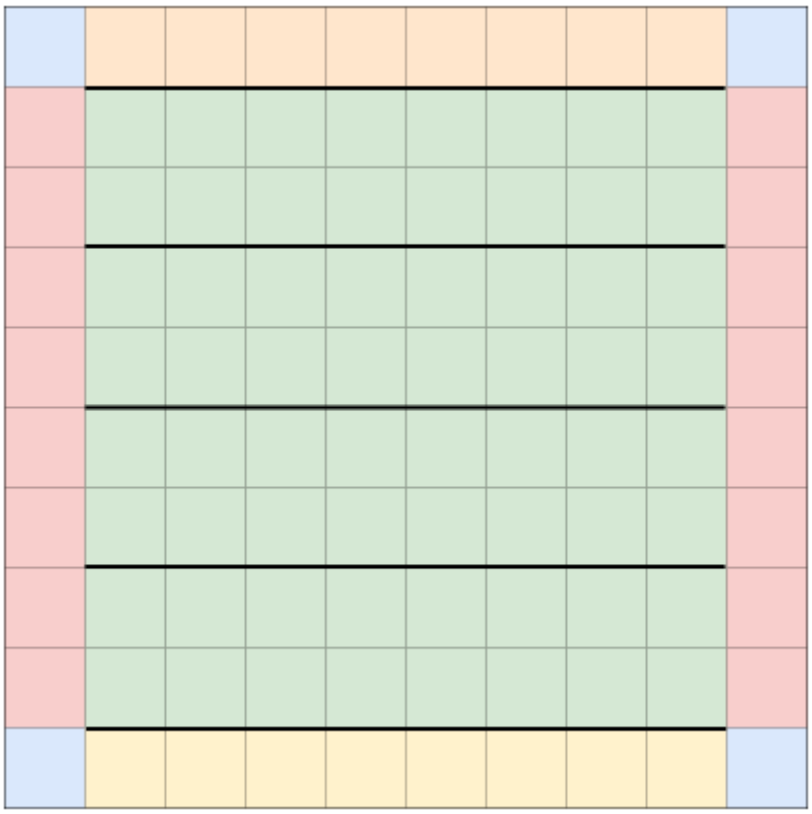

Рисунок 3.2.3 – схематичне пояснення алгоритму обчислення клітинного автомату

Кожний колір на рисунку 3.2.3 позначає, що для обчислення деякої групи клітин застосовуються окремі правила. Найпростіше для обчислення піддається «внутрішня матриця», яка виділена зеленим. Для кожної клітини в даній зоні гарантовано є сусід доступний без нормалізації координат. Також дану групу клітин легко розпаралелити для обчислення (виділено жирними лініями). Інші ж групи клітин вимагають нормалізації координат для деяких їхніх сусідів. Зв’язано це з тим, що решітка має утворювати не просто матрицю, а тор.

Для правої та лівої сторін застосовуються особливі правила обчислення. Якщо кластер працює не в stand-alone режимі, кожний робітник має посилання на свого «правого» та «лівого» сусідів і під час обчислення наступної ітерації, опитує їх про відповідні сторони кожного. Після опитування, робітник має всю інформацію необхідну для повного обчислення своєї ділянки решітки клітинного автомату.

3.3 Розробка клієнта

Для зручного управління кластером було вирішено розробити простий клієнт. Протокол комунікації – HTTP. В якості сервера виступає мастер машина в кластері.

Нажаль для Scala немає стабільних GUI бібліотек, тому було вирішено скористатись стороннім продуктом. Godot це ігровий рушій і фреймворк. Окрім ігрової частини, в Godot наявні модулі для створення GUI. Godot простий в використанні і ідеально підходить для прототипіювання.

Рисунок 3.3.1 – Godot engine

Розробку на Godot можна здійснювати тільки програмним чином або використовуючи візуальний редактор. Другий спосіб більш простий виконанні та вважається канонічним.

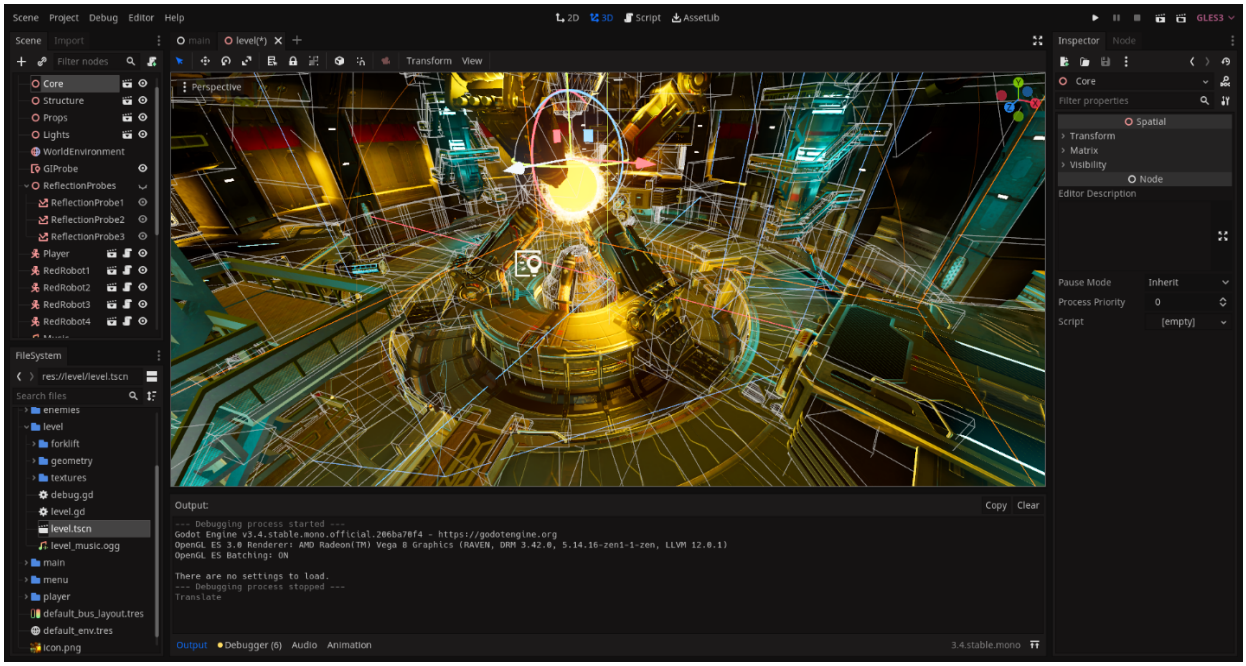

Рисунок 3.3.2 – редактор Godot

В результаті роботи було отримано простий GUI для керування кластером. Реалізовані базові органи управління: вікно з підключенням, кнопки керування кластером, статуси задачі, журнал та вікно створення нової задачі.

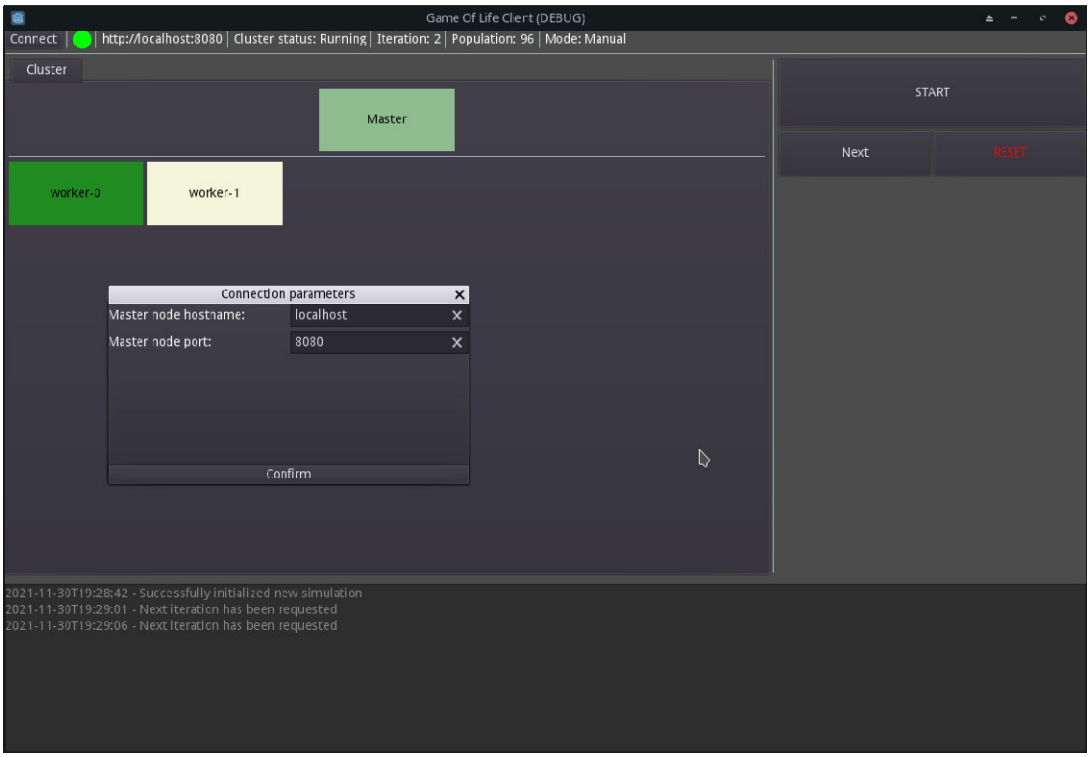

Рисунок 3.3.3 – діалогове вікно підключення

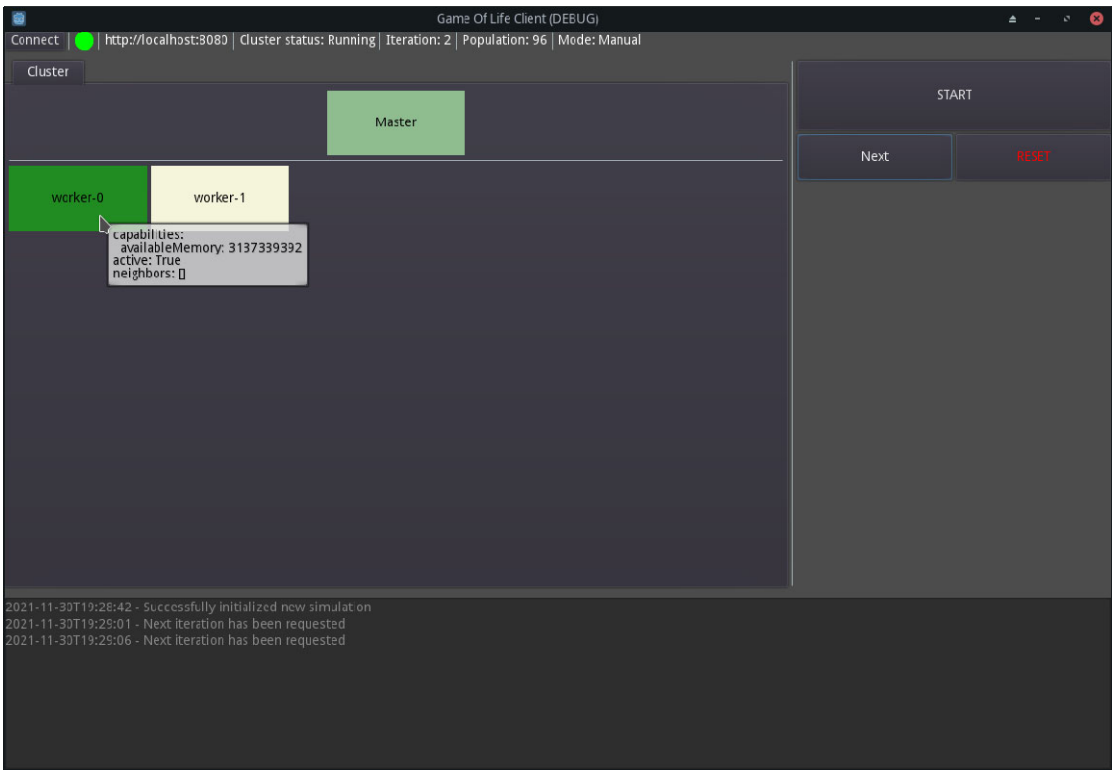

Рисунок 3.3.4 – візуалізація кластеру

Головним елементом клієнта є панель візуалізації кластера. На ній можна побачити статус кластера в цілому та окремо кожного робітника. Зеленим виділені робітники які приймають участь в теперішній симуляції. Жовтим виділені робітники в статусі idle. Якщо навести мишею на прямокутник робітника, можна побачити додаткові метадані: ресурси, статус та сусідів (якщо такі є).

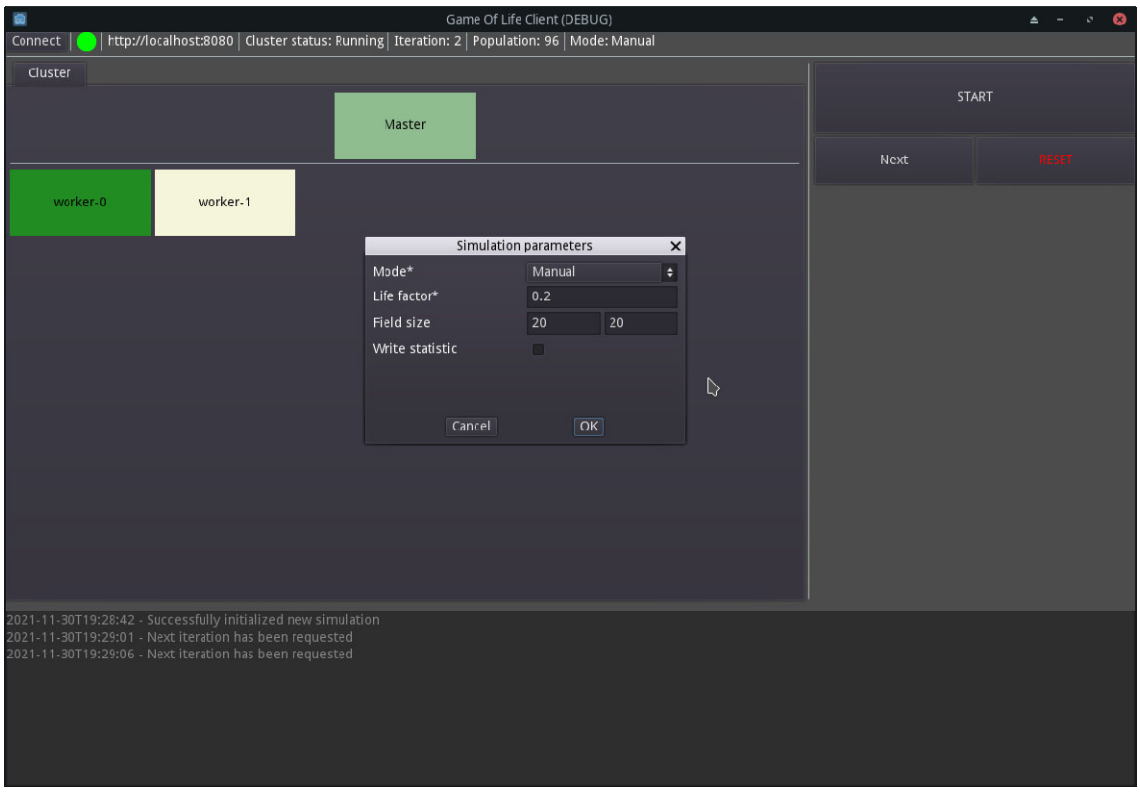

Рисунок 3.3.5 – діалогове вікно налаштувань задачі

В діалоговому вікні налаштувань задачі можна вказати основні параметри задачі: режим, коефіцієнт життя, розмір поля та чи потрібно вести лог файл зі статистикою задачі.

3.4 Unit тестування

Для даної роботи unit тестування є критично важливим. Окремі частити проекту вимагають ретельного тестування на коректність:

1. Розподіл задачі на менші і призначення їх робітникам.
2. Алгоритм обрахунку клітинного автомату.

Всі unit тести реалізовані в загальній кодовій базі. Для реалізації була використана бібліотека scalatest та scalameter для тестів на продуктивність. Для тестування алгоритму обрахунку клітинного автомату було написано тест кейси які перевіряють загальну коректність роботи, коректність замикань тора і врахування сусідських сторін. Шість тестів працюють в контексті наступної ітерації і два тести перевіряють коректність наступних п’яти ітерацій.

Тести продуктивності корисні для ідентифікації регресії в продуктивності алгоритму, а також для тестувань оптимізацій алгоритму.

3.5 Інтеграційне тестування

Для цілей інтеграційного тестування було розроблено окрему функцію кластера – самоперевірку. Дана процедура при її запуску створює заздалегідь підготовлену задачу і ініціює обчислення нової ітерації. Якщо результати співпадають з очікуваними, то вважається, що самоперевірка пройдена. Дана процедура вимагає наявності як мінімум двох робітників в кластері. Також, очевидно, що кластер має бути не зайнятий іншою роботою.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Оскільки продуктом є інформаційна система, яка призначена для запуску на персональних або серверних комп'ютерних системах, то всі вимоги до роботи з даним ПЗ відноситимуться до роботи з ПК.

Умови та безпека праці, їх стан та покращення – самостійна і важлива задача соціальної політики будь-якої сучасної промислово розвинутої держави, яку вирішує така невід'ємна складова БЖД, як охорона праці.

Рівень безпеки будь-яких робіт у суспільному виробництві значною мірою залежить від рівня правового забезпечення цих питань, тобто від якості та повноти викладення відповідних вимог в законах та інших нормативно-правових актах.

Для вирішення існуючих проблем в сфері охорони праці необхідна ефективна взаємодія всіх органів державної влади та громадськості, а також реалізація як на державному, так і на місцевих рівнях відповідних програм, спрямованих на корінне покращення умов і охорони праці.

Згідно Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями від 14.02.2018 [1] існують мінімальні вимоги безпеки під час роботи з екранними пристроями та мінімальні вимоги безпеки до екранних пристроїв. Вимоги безпеки до робочих місць працівників з екранними пристроями:

1. Робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів.

2. Для забезпечення безпеки та захисту здоров'я працівників усе випромінювання від екранних пристроїв має бути зведене до гранично

допустимого рівня (вплив на людину факторів довкілля - шуму, вібрації, забруднювачів, температури тощо, який не спричиняє соматичних або психічних розладів, а також змін стану здоров'я, працездатності, поведінки, що виходять за межі пристосувальних реакцій) з погляду безпеки та охорони здоров'я працівників.

3. Організація робочого місця працівника з екранними пристроями має забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.

4. Освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [2].

5. Мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року № 42 (далі - ДСН 3.3.6.042-99) [3].

6. Робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування.

7. Робоче крісло має бути стійким і дозволяти працівнику з екранними пристроями легко рухатися та займати зручне положення. Сидіння має регулюватися по висоті, спинка сидіння - як по висоті, так і по нахилу. Слід передбачати підніжку для тих, кому це необхідно для зручності.

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

1. Щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.

2. Після закінчення роботи екранні пристрої слід відключати від електричної мережі.

3. У разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі.

4. Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

5. Під час виконання робіт операторського типу, пов'язаних з нервово-емоційним напруженням, у приміщеннях під час роботи з екранними пристроями, на пультах і постах керування технологічними процесами та в інших приміщеннях мають дотримуватися оптимальні умови мікроклімату відповідно до вимог ДСН 3.3.6.042-99 [3].

Таким чином розроблена програмна має працювати на комп'ютерних системах які відповідають вимогам по безпеці роботи з екранними пристроями і охорони праці в цілому згідно чинного законодавства.

Список посилань:

1. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#n14.

2. ДСанПІН 3.3.2.007-98 [Електронний ресурс]. – 1998. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text.

3. ДСН 3.3.6.042-99 [Електронний ресурс]. – 1999. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text.

4.2 Безпека в надзвичайних ситуація

Умови й організацію праці при роботі з візуальними дисплейними терміналами усіх типів вітчизняного та зарубіжного виробництва на основі електронно-променевих трубок, що використовуються в електронно-обчислювальних машинах (ЕОМ*) колективного використання та персональних, визначають Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин.

Мінімальні вимоги безпеки та захисту здоров'я під час роботи, пов'язаної з використанням екранних пристроїв незалежно від їхнього типу та моделі, визначають Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, затверджені наказом Міністерства соціальної політики України від 14 лютого 2018 р. № 207 (далі — НПАОП 0.00-7.15-18).

Державні стандарти (ДСТУ EN) для користувачів не вимагають наявності інструкції з охорони праці під час роботи з комп'ютерною технікою, а передбачають правила користування (настанову з експлуатування) від заводу-виробника.

Виробники забезпечують супроводження електрообладнання інструкціями та інформацією про безпечність, складеними згідно з вимогами закону щодо порядку застосування мов. Зазначені інструкції та інформація, а також будь-яке маркування повинні бути чіткими, зрозумілими та дохідливими (п. 13 Технічного регламенту низьковольтного електричного обладнання, затвердженого постановою Кабінету Міністрів України від 16 грудня 2015 р. № 1067).

Порядок присвоєння кваліфікаційної групи з електробезпеки на виробництві визначає НПАОП 40.1-1.21-98 Правила безпечної експлуатації електроустановок споживачів, затверджений наказом Державного комітету України по нагляду за охороною праці від 9 січня 1998 р. № 4, що поширюється на працівників, які обслуговують діючі електроустановки споживачів. Під обслуговуванням (технічним) розуміють комплекс робіт з підтримки

працездатності обладнання в період його використання, до якого належать роботи з випробування обладнання, пристроїв, огляд обладнання, підтяжка контактних з'єднань (п. 3.1 розділу ІІІ Правил технічної експлуатації електроустановок споживачів, затверджених наказом Міністерства палива та енергетики України від 25 липня 2006 р. № 258 (далі — ПТЕЕС).

Оскільки продуктом є інформаційна система, яка призначена для запуску на персональних або серверних комп'ютерних системах, то всі вимоги до роботи з даним ПЗ відноситимуться до роботи з ПК.

Умови та безпека праці, їх стан та покращення – самостійна і важлива задача соціальної політики будь-якої сучасної промислово розвинутої держави, яку вирішує така невід'ємна складова БЖД, як охорона праці.

Рівень безпеки будь-яких робіт у суспільному виробництві значною мірою залежить від рівня правового забезпечення цих питань, тобто від якості та повноти викладення відповідних вимог в законах та інших нормативно-правових актах.

Для вирішення існуючих проблем в сфері охорони праці необхідна ефективна взаємодія всіх органів державної влади та громадськості, а також реалізація як на державному, так і на місцевих рівнях відповідних програм, спрямованих на корінне покращення умов і охорони праці.

Згідно Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями від 14.02.2018 [1] існують мінімальні вимоги безпеки під час роботи з екранними пристроями та мінімальні вимоги безпеки до екранних пристроїв. Вимоги безпеки до робочих місць працівників з екранними пристроями:

1. Робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів.

2. Для забезпечення безпеки та захисту здоров'я працівників усе випромінювання від екранних пристроїв має бути зведене до гранично допустимого рівня (вплив на людину факторів довкілля - шуму, вібрації,

забруднювачів, температури тощо, який не спричиняє соматичних або психічних розладів, а також змін стану здоров'я, працездатності, поведінки, що виходять за межі пристосувальних реакцій) з погляду безпеки та охорони здоров'я працівників.

3. Організація робочого місця працівника з екранними пристроями має забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.

4. Освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [2].

5. Мікроклімат виробничих приміщень з робочими місцями працівників з екранними пристроями має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року № 42 (далі - ДСН 3.3.6.042-99) [3].

6. Робочий стіл або робоча поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, допускати гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування.

7. Робоче крісло має бути стійким і дозволяти працівнику з екранними пристроями легко рухатися та займати зручне положення. Сидіння має регулюватися по висоті, спинка сидіння - як по висоті, так і по нахилу. Слід передбачати підніжку для тих, кому це необхідно для зручності.

Мінімальні вимоги безпеки під час роботи з екранними пристроями:

1. Щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.

2. Після закінчення роботи екранні пристрої слід відключати від електричної мережі.

3. У разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі.

4. Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

5. Під час виконання робіт операторського типу, пов'язаних з нервово-емоційним напруженням, у приміщеннях під час роботи з екранними пристроями, на пультах і постах керування технологічними процесами та в інших приміщеннях мають дотримуватися оптимальні умови мікроклімату відповідно до вимог ДСН 3.3.6.042-99 [3].

Таким чином розроблена програмна має працювати на комп'ютерних системах які відповідають вимогам по безпеці роботи з екранними пристроями і охорони праці в цілому згідно чинного законодавства.

Список посилань:

1. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#n14.
2. ДСанПІН 3.3.2.007-98 [Електронний ресурс]. – 1998. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text.
3. ДСН 3.3.6.042-99 [Електронний ресурс]. – 1999. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text.

ВИСНОВКИ

Дана кваліфікаційна робота показує як можна реалізувати складну розподілену систему з мінімумом часових витрат. Добре підібрані технології, бібліотеки та фреймворки грають не останню роль в розробці ПЗ.

Реалізована система володіє всіма властивостями реактивних систем та дотримується принципів реактивної архітектури та програмування. В даній роботі використовувались тільки сучасні та популярні технології, тому її практична цінність очевидна. Розроблена система пройшла повний цикл проектування та розробки ПЗ. Було виконане кінцеве тестування та написані тести. Для кінцевої системи був розробленим багатоплатформний клієнт на технології Godot.

Задача яку вирішує розроблена платформа володіє науковою цінністю. Клітинні автомати широко застосовуються та досліджуються в різних сферах науки. Потенційно дана система може обраховувати клітинні автомати любого типу та виду. Наразі система здатна обрахувати безкінечно можливе поле клітинного автомату «Гра життя» Д. Конвея.

Розроблена система легко інтегрується, як на одній фізичній машині, так і в складному датацентрі. При правильному налаштуванні система здатна бути географічно розподіленою (grid обчислення) [13].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Conway's Game of Life [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.
2. The Scala Programming Language [Електронний ресурс] – Режим доступу до ресурсу: https://www.scala-lang.org/.
3. Akka [Електронний ресурс] – Режим доступу до ресурсу: https://akka.io/.
4. Distributed computing [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Distributed_computing.
5. Cellular automaton [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cellular_automaton.
6. John Horton Conway [Електронний ресурс] – Режим доступу до ресурсу: https://mathgenealogy.org/id.php?id=18849.
7. Godot Engine [Електронний ресурс] – Режим доступу до ресурсу: https://godotengine.org/.
8. Martin Odersky [Електронний ресурс] – Режим доступу до ресурсу: https://lampwww.epfl.ch/~odersky/.
9. Actor model [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Actor_model.
10. Microservices architecture style [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.
11. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/?host=intellijidea.net.
12. Mill [Електронний ресурс] – Режим доступу до ресурсу: https://com-lihaoyi.github.io/mill/mill/Intro_to_Mill.html.
13. Grid computing [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Grid_computing.

ДОДАТКИ