Министерство цифрового развития государственного управления, информационных технологий и связи Республики Татарстан

государственное автономное профессиональное образовательное учреждение

«Международный центр компетенций - Казанский техникум

информационных технологий и связи»

|  |  |
| --- | --- |
| Дипломный проект выполнен и защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Председатель ГЭК  \_\_\_\_\_\_\_\_\_\_\_\_ А.Ф.Хасьянов  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2024 г | ДОПУСТИТЬ К ЗАЩИТЕ  Заместитель директора по учебной работе  \_\_\_\_\_\_\_\_\_\_\_ О. С. Тимофеева |

ДИПЛОМНЫЙ ПРОЕКТ

Проектирование и разработка симуляции «Игра жизнь» с использованием ранее разработанных общих модулей и инструментария

|  |  |  |
| --- | --- | --- |
| Исполнитель,  студент группы 423 СИС | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись, дата* | Р.А. Гарипов |
| Руководитель  директор института цифровых технологий и экономики ФГБОУ ВО «КГЭУ» | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись, дата* | Э.И. Беляев |

Казань, 2024г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc169087507)

[1 АНАЛИТИЧЕСКАЯ ЧАСТЬ 7](#_Toc169087508)

[1.1 Описание и анализ предметной области 7](#_Toc169087509)

[1.2 Обзор аналогов 8](#_Toc169087510)

[2 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ 10](#_Toc169087511)

[2.1 Обоснование выбора языка программирования 10](#_Toc169087512)

[2.2 Обоснование выбора фреймворка 12](#_Toc169087513)

[2.3 Обоснование выбора СУБД 14](#_Toc169087514)

[2.4 Обоснование выбора среды разработки 15](#_Toc169087515)

[2.5 Описание используемых библиотек 16](#_Toc169087516)

[2.5.1 Обоснование выбора библиотеки SkiaSharp 17](#_Toc169087517)

[2.5.2 Обоснование выбора библиотеки Actipro 18](#_Toc169087518)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 19](#_Toc169087519)

[3.1 Диаграмма Use Case 19](#_Toc169087520)

[3.2 ER-диаграмма базы данных 20](#_Toc169087521)

[4 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА 22](#_Toc169087522)

[4.1 Создание базы данных 22](#_Toc169087523)

[4.2 Структура проекта симуляции 25](#_Toc169087524)

[4.3 Разработка доменной модели симуляции Игра Жизнь 27](#_Toc169087525)

[5 Тестирование программного продукта 31](#_Toc169087526)

[5.1 Понятие и виды тестирования 31](#_Toc169087527)

[5.2 Методы тестирования 32](#_Toc169087528)

[5.3 Тестирование приложения с использованием тест-кейсов 33](#_Toc169087529)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 38](#_Toc169087530)

[6.1 Запуск симуляции с использованием Лаунчера 38](#_Toc169087531)

[6.2 Окно запущенной симуляции 40](#_Toc169087532)

[6.3 Запуск и настройка симуляции 46](#_Toc169087533)

[ЗАКЛЮЧЕНИЕ 51](#_Toc169087534)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 53](#_Toc169087535)

[ПРИЛОЖЕНИЕ А 55](#_Toc169087536)

[ПРИЛОЖЕНИЕ Б 57](#_Toc169087537)

[ПРИЛОЖЕНИЕ В 58](#_Toc169087538)

[ПРИЛОЖЕНИЕ Г 60](#_Toc169087539)

государственное автономное профессиональное образовательное учреждение

«Международный центр компетенций –

Казанский техникум информационных технологий и связи»

|  |
| --- |
| УТВЕРЖДАЮ  Заместитель директора  по учебной работе  \_\_\_\_\_\_\_\_\_\_\_\_\_ О.С. Тимофеева  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

ЗАДАНИЕ

на дипломный проект

Выпускнику группы 423 СИС Гарипов Руслан Айратович по специальности 09.02.07 Информационные системы и программирование, квалификация: Специалист по информационным системам

1. Тема дипломного проекта: «Проектирование и разработка симуляции «Игра жизнь» с использованием ранее разработанного инструментария», утверждена приказом директора техникума от 18.03.24 № 85 - с
2. Срок сдачи студентом законченного ДП: 13.06.24
3. Исходные данные к ДП: техническое задание.
4. Краткое содержание ДП или перечень подлежащих разработке вопросов:
   1. Краткое содержание пояснительной записки ДП: аналитическая часть, проектирование симуляции, разработка симуляции, тестирование симуляции разработка руководства пользователя, заключение, список использованных источников, приложения.
   2. Требования к программному продукту: кроссплатформенность/ поддержка десктопных ОС Windows, локализация, беспрерывный доступ к системе, отказоустойчивость, надёжность и безопасность. Иметь интуитивно понятный интерфейс.
5. Перечень графических материалов/Приложения: техническое задание, словарь данных, скрип базы данных и программный код.

План-график выполнения дипломного проекта:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | Наименование этапов ДП | Срок выполнения этапов | | Подпись руководителя, консультантов |
| План | Факт |
| 1 | Получение задания | 25.03.24- 30.03.24 |  |  |
| 2 | Анализ предметной области и сбор данных | 25.03.24- 30.03.24 |  |  |
| 3 | Проектирование системы | 06.04.24- 10.04.24 |  |  |
| 4 | Разработка и создание компонентов | 12.04.24- 15.05.24 |  |  |
| 5 | Тестирование и отладка | 15.05.24- 18.05.24 |  |  |
| 6 | Написание руководства пользователя | 18.05.24- 25.05.24 |  |  |
| 7 | Подготовка и конечное оформление документации | 25.05.24- 29.05.24 |  |  |
| 9 | Внесение исправлений и финальная доработка проекта | 29.06.24- 13.06.24 |  |  |

Рассмотрено на заседании ЦК Программирование

Протокол № 8 от 22.03.24

Председатель ЦК\_ Л.Р. Калинина ь

*подпись инициалы и фамилия*

Дата выдачи задания:

«25» марта 2024г.

Руководитель дипломного проекта:

Директор ИЦТЭ

ФГБОУ ВО «КГЭУ» \_\_\_\_\_\_\_\_\_\_\_\_ Э.И. Беляев ь

*подпись инициалы и фамилия*

Задание принял к исполнению: Р.А. Гарипов ь

*подпись инициалы и фамилия*

# ВВЕДЕНИЕ

Разработка проекта по созданию клеточного автомата «Игра Жизнь» имеет свои истоки в концепции, задуманной английским математиком Джоном Конвеем в 1970 году. Это игра без игроков, в которой человек создает начальное состояние и затем наблюдает за развитием событий.

Игра происходит на плоскости, разделенной на клетки, каждая из которых может быть «живой» или «мертвой». Правила игры определяют, какие клетки будут «жить» в следующем поколении, исходя из их состояния и состояния их соседей. Клеточный автомат «Игра Жизнь» позволяет создавать процессы с полнотой по Тьюрингу, что открывает возможности для реализации различных вычислительных моделей.

Дипломный проект, воплощает в себе идею симуляции «Игра Жизнь», основанной на принципах одноименного клеточного автомата, придуманного Джоном Конвеем.

Основная цель проекта заключается в том, чтобы дать возможность протестировать ранее разработанные модули и архитектуру проекта. Важным аспектом является удобство использования проекта для не знакомых с разработкой и проектом пользователей.

Работа будет осуществляться поэтапно, следуя плану: определение первоначальных задач, которые необходимо реализовать, анализ предметной области симуляции для понимания ее специфики и требований, разработка симуляции, программирование и настройка базы данных для хранения информации, тестирование программного продукта для обеспечения его корректной работы, создание всей необходимой документации для последующего использования и поддержки систем

Ожидаемым результатом дипломного проекта является полноценная информационная система, которая будет полностью соответствовать всем предъявляемым к ней требованиям.

Техническое задание представлено в ПРИЛОЖЕНИЕ А.

# 1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

## Описание и анализ предметной области

Клеточный автомат – дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Основой является пространство из прилегающих друг к другу клеток, образующих решётку.

Представление двумерной модели включает следующие характеристики:

* клетки имеют 2 состояния: жива или мертва;
* клетки без смещений;
* правила изменения состояния зависят только от количества живых соседей из окрестности Мура первого порядка, в окрестность включается 8 окружающих клеток.
* Правила учитывают 4 исхода для каждой клетки: рождение, выживание, смерть, пустота.

Решётка может быть любой размерности, бесконечной или конечной, для решётки с конечными размерами часто предусматривается закольцованность при достижении границ. Для каждой клетки определено множество клеток, называемых окрестностью.

Правила симуляции предусматривает клеточное поле, состоящее из клеток, которые имеют два состояния: живые или мёртвые, расчет состояния клеток производится по поколениям в соответствии с определенными правилами;

Аспектом симуляции является создание и применение шаблонов и структур в «Игре Жизнь», таких как «планеры», «стабильные фигуры», «осцилляторы» и другие.

Основное направление исследования клеточных автоматов — алгоритмическая разрешимость тех или иных задач.

Также рассматриваются вопросы построения начальных состояний, при которых клеточный автомат будет решать заданную задачу.

## 1.2 Обзор аналогов

Для более глубокого понимания и разработки требований к планируемой информационной системе, необходимо изучить уже существующие аналоги с схожими возможностями.

На данный момент существует лишь одно значимое сравнимое решение Golly, которое предлагает подобный функционал и предоставляет возможность пользователям или разработчикам модифицировать и расширять открытый исходный код.

Golly – это кроссплатформенное приложение с открытым исходным кодом для изучения игры Конвея в жизнь и многих других типов клеточных автоматов. Основными авторами являются Эндрю Треворроу и Том Рокицки, а авторами кода являются Крис Роуэтт, Тим Хаттон, Дэйв Грин, Джейсон Саммерс, Макс Вервер, Роберт Мунафо, Брентон Бостик и Донгук Ли (Рисунок 1.1).

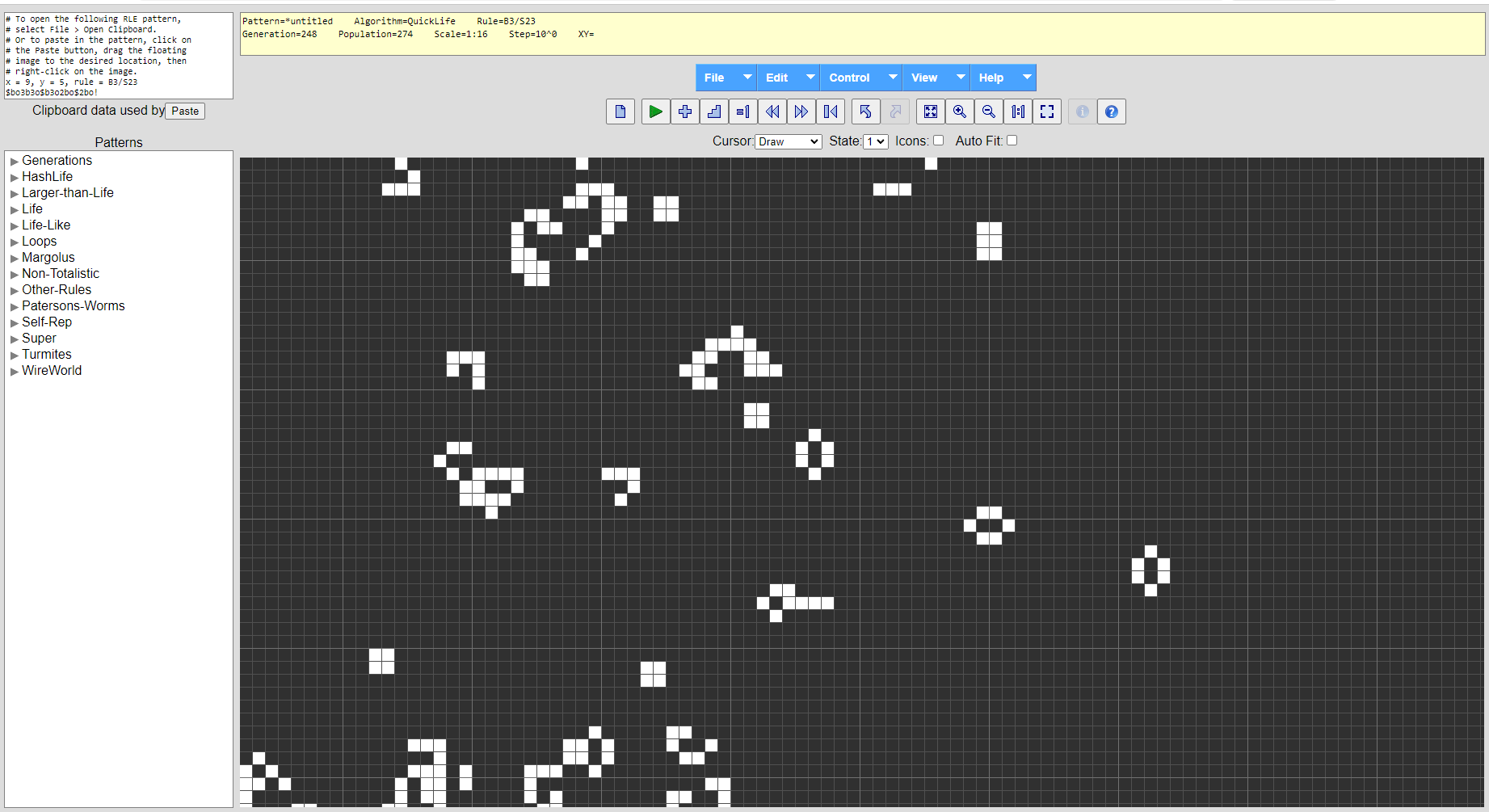


Рисунок 1.1 кроссплатформенное приложение Golly

Кроссплатформенное приложение Golly обладает рядом преимуществ и недостатков, которые необходимо учитывать при его использовании. Среди преимуществ следует выделить возможность неограниченного масштабирования симуляций для создания крупных и сложных проектов.

Стоить отметить высокую производительность приложения и отсутствие подвисаний, что позитивно сказывается на пользовательском опыте.

Кроме того, важно отметить, что многообразие доступных шаблонов и правил симуляции в приложении Golly предоставляет пользователям широкие возможности для настройки симуляций согласно их собственным предпочтениям. Пользователи могут изменять параметры симуляций, применять различные шаблоны и правила, чтобы создавать уникальные и интересные модели и эксперименты. Это способствует креативному процессу и позволяет каждому пользователю настроить симуляцию по своему усмотрению, расширяя возможности приложения и делая опыт использования более персонализированным и увлекательным (Рисунок 1.2).

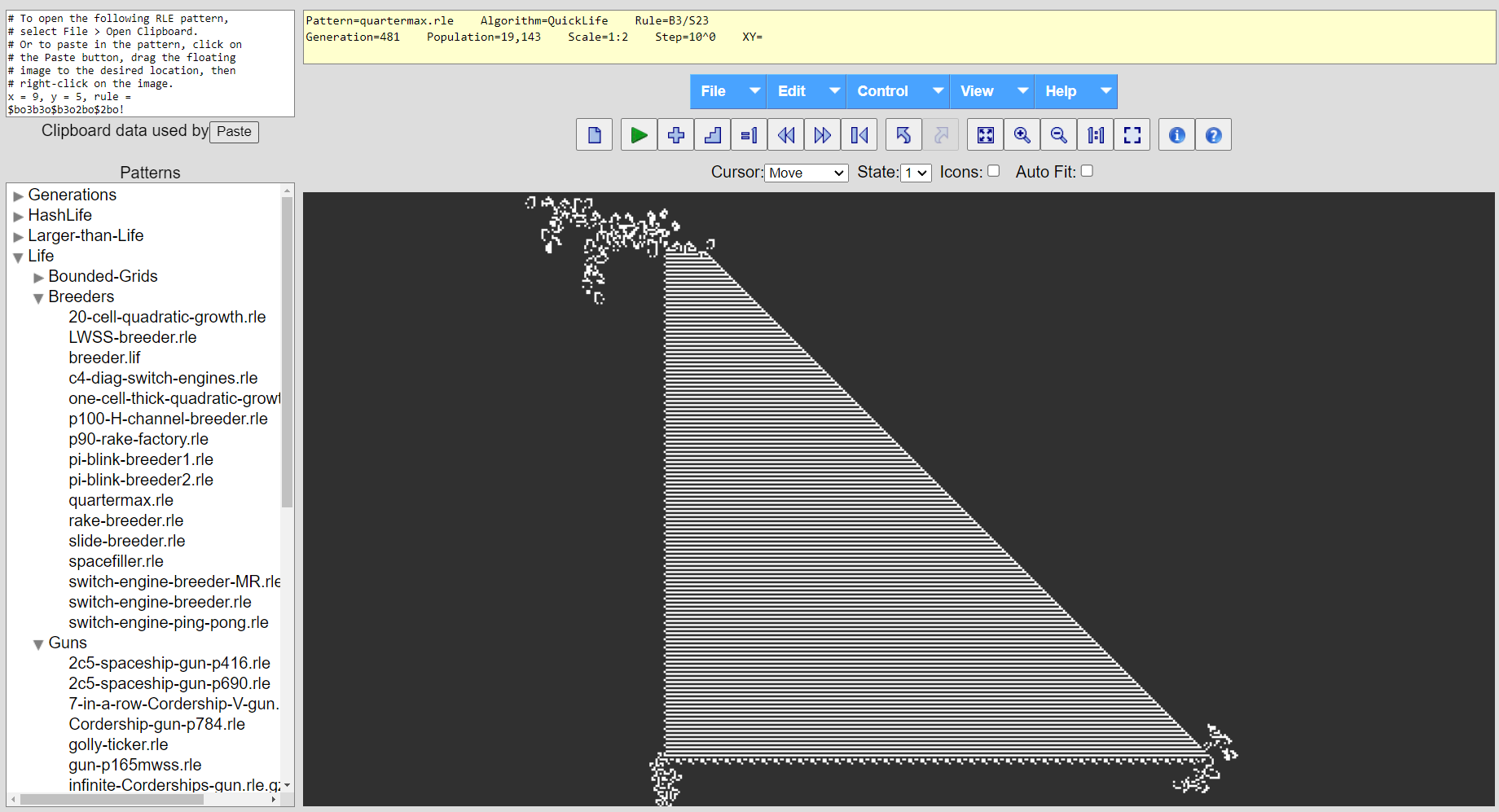


Рисунок 1.2 Выбор готового шаблона и последующий запуск симуляции

Среди недостатков стоит выделить отсутствие локализации приложения на разные языки, что может ограничить доступность для пользователей из различных стран. Основным недостатком можно считать сложность и запутанность исходного кода, написанного на языке C++, что может создать дополнительные трудности для разработчиков, желающих расширить или модифицировать функционал приложения на ранних этапах проектирования и реализации.

# 2 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ

## 2.1 Обоснование выбора языка программирования

C# – современный, ориентированный на объекты и безопасный язык программирования, предназначенный для создания надежных приложений в среде .NET. Он унаследовал синтаксис и некоторые концепции от языков семейства C, таких как C, C++, Java и JavaScript. [1]

C# представляет собой язык программирования, который сочетает в себе простоту и мощность. Его удобный синтаксис делает процесс написания кода крайне интуитивным, что упрощает разработку для программистов всех уровней опыта.

Строгая типизация C# помогает выявлять ошибки на этапе компиляции, что делает код более надежным и устойчивым к возможным ошибкам выполнения программы. Это способствует созданию качественных приложений, которые работают более эффективно.

Кроме того, C# поддерживает современные парадигмы программирования, такие как объектно-ориентированное программирование, функциональное программирование и асинхронное программирование. Это дает разработчикам большую свободу выбора подхода к решению задач и дает возможность создавать более гибкие и масштабируемые приложения.

С интеграцией с платформой Windows C# становится идеальным решением для разработки приложений для данной операционной системы. Это обеспечивает более тесное взаимодействие с функциональностью Windows API и упрощает создание графических приложений для пользователей Windows.

C# предлагает разработчикам все необходимое для создания масштабируемых, надежных и производительных приложений, что делает его одним из наиболее популярных и востребованных языков программирования в современном мире разработки программного обеспечения.

Также стоит упомянуть большое сообщество разработчиков, которые всегда могут ответить на интересующий вопрос и подсказать необходимые источники для более корректной реализации приложения.

Надежность и устойчивость приложений на C# обеспечиваются рядом фундаментальных механизмов и особенностей языка:

* система сборки мусора, важный компонент, который автоматически освобождает память, занимаемую объектами, на которые больше нет ссылок, а также данная система предотвращает утечки памяти и позволяет разработчикам не беспокоиться о ручном освобождении ресурсов.
* Типы данных Nullable позволяет более явно определить, когда переменная может быть пустой, что способствует более безопасной работе с данными.
* Обработка исключений мощная система обработки исключений, которая позволяет обнаруживать и восстанавливаться от ошибок в программе, а также структурированный подход к обработке исключений упрощает понимание и отладку кода, а также обеспечивает более плавное исполнение программы.
* Лямбда-выражения один из стандартных блоков LINQ, они не только удобны для создания анонимных функций, но и поддерживают функциональное программирование, что делает код более гибким и выразительным.
* LINQ (Language Integrated Query) является мощным инструментом, позволяющий писать запросы к данным на языке C# непосредственно в коде, что делает их более понятными и легко поддерживаемыми.
* Асинхронное программирование предоставляет возможность приоритетного выполнения определенных задач без ожидания сигнала об их завершении и вызванной его отсутствие блокировки других работающих программ.

Производительность является одним из важнейших критериев при разработке приложений. Хорошо спроектированный язык, такой как C#, обеспечивает эффективное использование ресурсов компьютера и оптимизированное выполнение кода, что может значительно повысить производительность приложения.

Асинхронные операции становятся все более важными в современном программировании, особенно при работе с сетевыми запросами, вводом-выводом (I/O) и асинхронными событиями. Языки программирования, такие как C#, предоставляют удобные средства для реализации асинхронных паттернов, что позволяет эффективно использовать ресурсы системы и повышать отзывчивость приложений.

Поддержка многопоточности также является ключевым аспектом при разработке масштабируемых и отзывчивых систем. Возможность выполнять несколько потоков параллельно позволяет эффективно использовать многоядерные процессоры и улучшает отклик приложения на запросы пользователей.

Языки программирования, такие как C#, предоставляют средства для управления потоками выполнения и синхронизации доступа к общим ресурсам, что делает разработку многопоточных приложений более удобной и безопасной.

Дополнительно, C# обладает широким спектром возможностей, таких как LINQ для работы с данными, асинхронные операции, итераторы для пользовательских коллекций, управление версиями для обеспечения совместимости программ и библиотек, а также интеграцию с другими технологиями Microsoft.

## 2.2 Обоснование выбора фреймворка

Avalonia – это кроссплатформенный фреймворк для разработки графических пользовательских интерфейсов (GUI) на языке программирования C#. Он обеспечивает возможность создания современных и стильных пользовательских интерфейсов, которые могут быть запущены на различных операционных системах, включая Windows, Linux и macOS. [2]

Ключевым аспектом для выбора платформы интерфейса Avalonia, является его производительность в сравнении с WPF и Window Forms.

Основные особенности Avalonia делают его мощным и гибким инструментом для разработки кроссплатформенных графических пользовательских интерфейсов:

* Avalonia поддерживает использование XAML для описания пользовательского интерфейса, что упрощает создание и структурирование интерфейсных элементов.
* Фреймворк Avalonia позволяет разработчикам создавать стили и шаблоны, которые могут быть легко применены ко всем элементам интерфейса, обеспечивая единообразный дизайн.
* Avalonia предлагает расширенную систему привязки данных, позволяющую легко связывать данные и элементы интерфейса, что делает разработку приложений более эффективной.
* С помощью библиотеки SkiaSharp, Avalonia обеспечивает возможность работы с векторной графикой, что позволяет создавать красивые и адаптивные интерфейсы.
* Уникальной чертой Avalonia является его независимость от платформы.
* Благодаря гибкости фреймворка и использованию современных технологий, Avalonia позволяет создавать красивые и функциональные приложения.

Высокая производительность Avalonia обеспечивает эффективное использование системных ресурсов, что позволяет создавать приложения, способные обрабатывать большие объемы данных или сложную логику с минимальными задержками и перегрузками.

Это достигается благодаря оптимизированной архитектуре Avalonia, которая учитывает особенности работы на различных операционных системах. Платформа использует современные технологии и методы программирования, такие как асинхронное выполнение и многопоточность, для максимального использования процессора и памяти.

Благодаря этим подходам, приложения, разработанные на Avalonia, демонстрируют высокую отзывчивость и быстродействие даже при работе с большими объемами данных. Это особенно важно для современных приложений, которые часто требуют обработки и отображения больших наборов информации или выполнения сложных вычислительных задач.

Кроме того, Avalonia предоставляет разработчикам инструменты и ресурсы для оптимизации производительности своих приложений. Это включает в себя возможности для профилирования кода, оптимизации запросов к данным, кэширования результатов и другие методы, направленные на улучшение скорости и эффективности работы приложений.

Таким образом, высокая производительность Avalonia делает эту платформу привлекательным выбором для разработчиков, стремящихся создать быстрые, отзывчивые и масштабируемые приложения для различных операционных систем.

## 2.3 Обоснование выбора СУБД

SQLite — это компактная встраиваемая система управления базами данных, которая используется для создания локальных баз данных в приложениях. SQLite отличается тем, что не требует отдельного сервера для своей работы, а база данных обрабатывается непосредственно приложением, что делает ее удобной для встраивания в различные приложения и устройства.

SQLite [3] является кроссплатформенным решением, предоставляя возможность работать на различных операционных системах без необходимости дополнительной настройки. Это делает его идеальным выбором для разработки мобильных приложений, веб-приложений и настольных программ.

Кроме того, SQLite отличается высокой производительностью и эффективностью использования ресурсов, что позволяет использовать его даже на устройствах с ограниченными возможностями, также это особенно отличает его от других подобных СУБД.

Основные преимущества использования SQLite:

* легко интегрируется в приложения благодаря своей простоте;
* обеспечивает высокую производительность при работе с небольшими и средними объемами данных, быстро выполняет запросы к базе данных, что делает его отличным выбором для задач, требующих быстрого доступа к информации.
* Требует минимальных ресурсов для работы, что делает его идеальным для использования на мобильных устройствах, веб-приложениях и других средах, где ограничены ресурсы.
* Обеспечивает надежное хранение данных, защиту от сбоев и целостность данных, имеет механизмы для обеспечения транзакционной безопасности и восстановления после сбоев.
* Поддерживается на многих платформах, что обеспечивает переносимость ваших данных и приложений на различные устройства и операционные системы.

Благодаря своей простоте и легковесности, SQLite широко используется в мобильных приложениях, настольных приложениях, встроенных системах, браузерах и других областях, где требуется локальное хранение и управление данными.

## 2.4 Обоснование выбора среды разработки

JetBrains Rider [4] представляет собой интегрированную среду разработки (IDE), предназначенную для работы с платформой .NET. Данный продукт был создан компанией JetBrains.

JetBrains Rider предлагает разработчикам полный набор инструментов, необходимых для создания, отладки и тестирования приложений на платформе .NET.

Кроме того, JetBrains Rider обладает продвинутыми средствами автодополнения кода, рефакторинга, навигации и поиска, что значительно ускоряет процесс разработки и повышает его эффективность.

Ключевые достоинств JetBrains Rider:

* предоставляет широкий спектр инструментов для разработки на .NET, включая поддержку C#;
* использует мощные инструменты анализа кода и автоматического завершения, что упрощает процесс написания кода и повышает производительность разработчика.
* Поддерживает работу на различных операционных системах, включая Windows, macOS и Linux, что позволяет разработчикам выбирать наиболее удобное окружение.
* Включает в себя функциональность ReSharper, расширяя возможности IDE и обеспечивая единое, удобное рабочее окружение для разработчиков.
* Предоставляет мощные инструменты для отладки и профилирования приложений, что помогает обнаруживать и исправлять ошибки более эффективно.
* Обеспечивает интеграцию с инструментами сборки, такими как MSBuild и dotnet CLI, и предоставляет удобные средства управления проектами.
* Данная среда разработки имеет интеграцию с различными системами контроля версий, такими как Git, SVN и Mercurial, упрощая работу с кодом в командной среде.

Эти возможности делают JetBrains Rider мощным инструментом для разработки на платформе .NET, обеспечивая удобство, производительность и качество кода.

## 2.5 Описание используемых библиотек

Правильный выбор библиотек помогает сделать разработку более эффективной и безопасной, что в конечном итоге ведет к созданию качественного программного продукта. Библиотеки «SkiaSharp» и «Actipro», обладающие широким спектром функциональности и инструментов, значительно упрощают процесс разработки.

Именно благодаря этим библиотекам, выбор которых был обусловлен их эффективностью и многофункциональностью, удалось достичь высоких результатов в рамках дипломного проекта.

SkiaSharp — это 2D-графическая система с открытым исходным кодом, которая широко применяется в продуктах Google. Она разработана для .NET и C# и предоставляет широкий набор инструментов для работы с графикой.

Actipro — это обширная коллекция элементов управления пользовательским интерфейсом, предназначенная для создания привлекательных настольных приложений WPF. Она включает в себя множество компонентов, таких как редакторы, панели инструментов, вкладки, диалоговые окна и многое другое, что позволяет разработчикам быстро и эффективно создавать пользовательские интерфейсы.

Использование подобных библиотек позволяет повысить производительность, ускорить разработку и сделать код более надежным.

### 2.5.1 Обоснование выбора библиотеки SkiaSharp

SkiaSharp предоставляет комплексный 2D API, который можно использовать в моделях мобильных, серверных и настольных компьютеров для рендеринга изображений. SkiaSharp доступен в виде удобного пакета NuGet.

Ключевые особенности библиотеки SkiaSharp включают в себя:

* поддержку платформ, таких как Android, iOS, macOS, Windows, Linux и другие;
* библиотеки обладает простым и интуитивно понятным интерфейсом, что упрощает разработку графических приложений;
* обеспечивает высокую производительность, что позволяет создавать плавные и отзывчивые интерфейсы;
* поддерживает рендеринг в различных форматах изображений, включая PNG, JPEG, SVG;
* предоставляет разнообразные инструменты для рисования, масштабирования, трансформации и обработки графики.

Основными достоинствами данной библиотеки является широкая сфера использования в разных IT-компаниях, которая обеспечивает обширное сообщество, так же ключевое достоинство «SkiaSharp» является производительность и открытый исходный код, который позволяется удобно и гибко использовать данную библиотеку.

Благодаря активному сообществу пользователей и разработчиков, «SkiaSharp» постоянно совершенствуется и обновляется, что помогает создавать современные и эстетичные приложения с впечатляющим пользовательским интерфейсом.

### 2.5.2 Обоснование выбора библиотеки Actipro

Actipro — это мощный инструмент для разработки настольных приложений WPF, предлагающий широкий спектр современных элементов управления пользовательским интерфейсом. Эта библиотека позволяет разработчикам быстро и эффективно создавать удобные и функциональные интерфейсы.

Одним из главных преимуществ использования элементов управления Actipro является значительное ускорение процесса создания интерфейса приложения.

Элементы управления Actipro отличаются богатым набором функциональных возможностей и стильным дизайном, что позволяет создавать привлекательные и современные пользовательские интерфейсы

Кроме того, Actipro обладает активным сообществом пользователей и разработчиков, которые готовы помочь и поддержать в случае возникновения вопросов или проблем. Сообщество предоставляет ресурсы, документацию, примеры кода и форумы для обсуждения, что делает работу с Actipro еще более продуктивной и приятной.

В целом, Actipro — это надежный и эффективный инструмент для разработки настольных приложений WPF, который поможет вам создать профессиональный и современный пользовательский интерфейс.

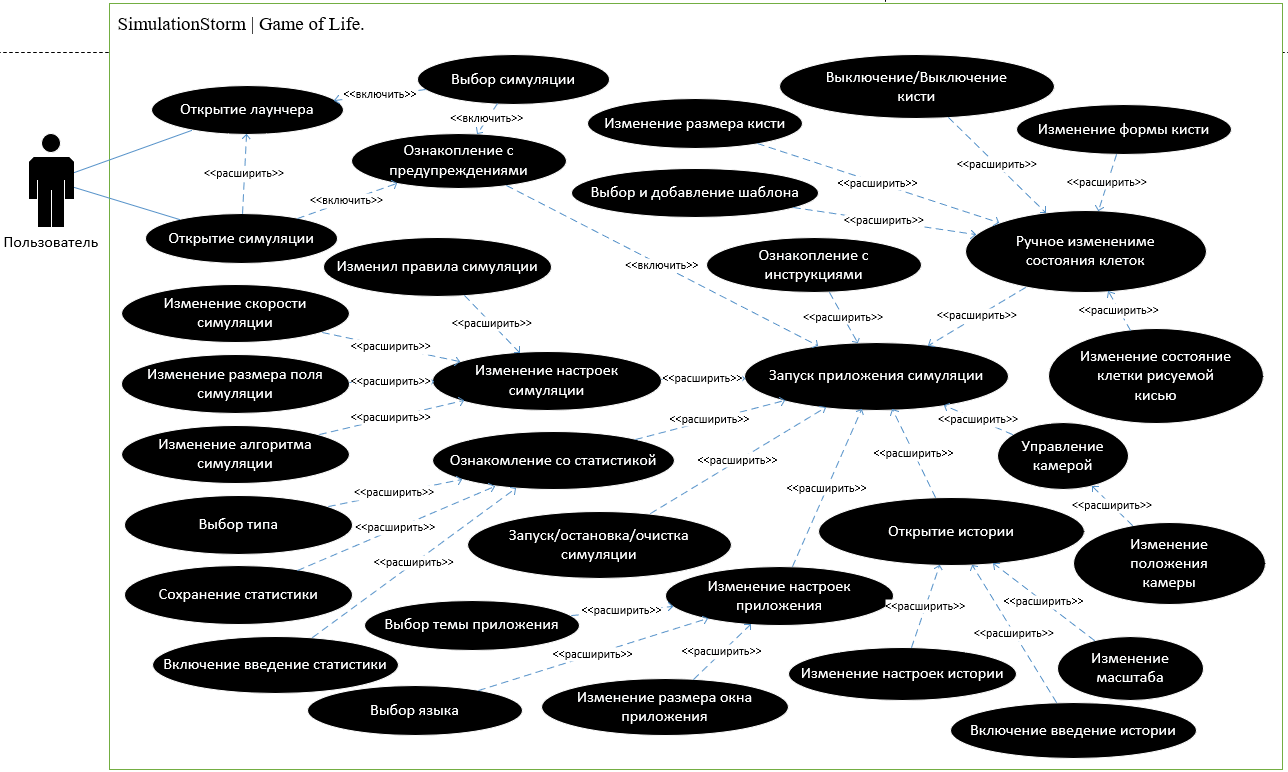
# 3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

## 3.1 Диаграмма Use Case

Use case (прецедент использования) – это описание взаимодействия между системой и ее актерами для достижения конкретной цели [5]. Он представляет собой сценарий или последовательность действий, которые выполняются для достижения определенного результата.

Прецеденты использования описывают функциональность системы с точки зрения конечного пользователя или других внешних участников системы. Они помогают определить требования к системе и понять, как она будет использоваться в реальных условиях.

Use case описывает, как система взаимодействует с пользователями, какие действия они выполняют и какие результаты они ожидают. Ниже представлена диаграмма Use Case (Рисунок 3.1).

 Рисунок 3.1 Диаграмма вариантов использования

Use Case помогает определить функциональные требования системы и является основой для проектирования и разработки системы, обеспечивая понимание взаимодействия между системой и пользователями для достижения конкретных целей.

## 3.2 ER-диаграмма базы данных

ER-диаграмма (Entity-Relationship diagram) — это графическое представление сущностей и их взаимосвязей в базе данных. Она широко используется при проектировании баз данных для описания структуры данных и их взаимосвязей.

На ER-диаграмме сущности представлены в виде прямоугольников, а связи между ними — линиями с различными связями. Каждая сущность имеет свои атрибуты, которые отображаются внутри прямоугольника сущности.

ER-диаграммы играют важную роль в создании и анализе баз данных, помогая визуализировать и уточнять логику связей между сущностями и проектировать базы данных более систематически (Рисунок 3.2).

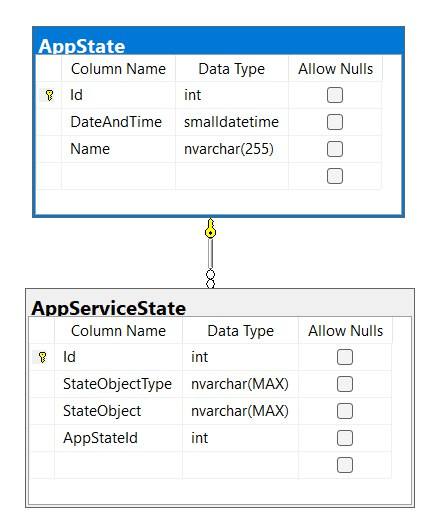


Рисунок 3.2 ER-диаграмма базы данных

База данных в данном проекте не является центральным элементом, однако ее роль заключается в сохранении настроек приложения, статистики и истории симуляций.

Когда данные о статистике сохраняются в базе данных, это позволяет пользователям проводить более глубокий анализ собранных данных, выявлять закономерности, делать выводы и принимать обоснованные решения.

Повторное использование статистики также играет важную роль в оптимизации процессов и принятии решений. Анализ ранее собранных данных позволяет извлечь уроки из прошлых ошибок, оптимизировать исследования и методы работы.

Структура ER-диаграммы:

* таблица AppState содержит информацию о названии сохранения и времени их записи в базу данных, данные атрибуты служат для более корректного определения сохранения, а также для удобства пользователя;
* таблица AppServiceState содержит сохранение какого-либо сервиса приложения, данная таблица имеет ссылку на сохранение, которому принадлежит.

Также стоит упомянуть основные атрибуты таблицы «AppServiceState»:

* атрибут StateObjectType хранит в себе тип объекта сохранения, данные храниться сериализованно;
* атрибут StateObject хранит в себе объект, содержащий информацию о состоянии сервиса, данные также хранятся сериализованно.

В целом, база данных является неотъемлемой частью приложения, обеспечивая хранение, управление и доступ к данным, необходимым для его функционирования. Ее эффективное использование способствует стабильной работе приложения и обеспечению безопасности и целостности данных.

# 4 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

## 4.1 Создание базы данных

Таблицы базы данных были реализованы, опираясь на ER-диаграмму. Для более корректного проектирования и создания базы данных был сформирован словарь данных, словарь данных предоставлен в ПРИЛОЖЕНИЕ Б. Типы данных и их ограничения выбираются, опираясь на словарь данных.

При создании базы данных был применен подход «Code First».

«Code First» — данный подход в разработке баз данных означает, что база данных создается на основе модели данных, описанной в коде приложения, код модели данных предоставлен в Приложении Г. Данный подход позволяет сосредоточиться на разработке приложения, не беспокоясь о деталях создания самой базы данных.

В контексте Entity Framework (EF) в среде .NET, подход Code First позволяет определить сущности и их отношения в виде классов сущностей (entity classes). Entity Framework затем автоматически создает соответствующую базу данных на основе этой модели, скрип базы данных предоставлен в Приложении В.

В Entity Framework подход «Code First» поддерживает механизм миграций, который помогает обновлять структуру базы данных в соответствии с изменениями модели данных без потери существующих данных, также данный подход позволяет быстро создавать прототипы и тестировать новые функции без необходимости предварительного создания схемы базы данных.

Также модель данных в коде делает код более чистым и читаемым, а также обеспечивает единообразие между бизнес-логикой и структурой данных.

Использование подхода «Code First» с помощью Entity Framework в среде .NET помог значительно облегчить процесс разработки приложений, сделал проект и взаимодействие с базой данных более гибким, прозрачным и удобным. Код создания базы данных с подходом «Code First» предоставлен в ПРИЛОЖЕНИЕ Г.

После создания и реализации модели данных автоматически была создана диаграмма базы данных и построены корректные связи между таблицами (Рисунок 4.1).

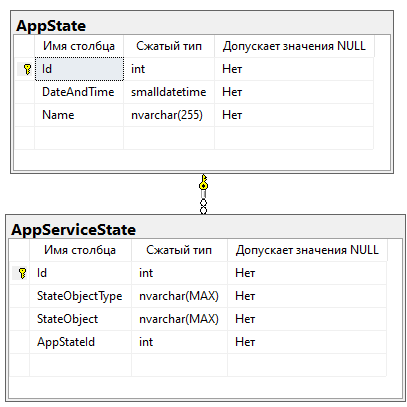
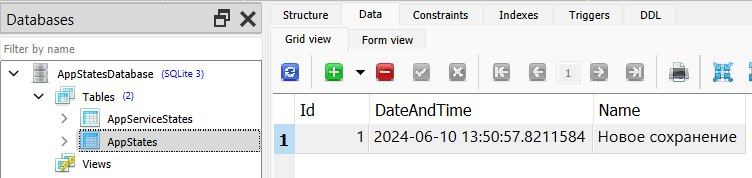


Рисунок 4.1 Диаграмма базы данных

Скрип базы данных предоставлен в ПРИЛОЖЕНИЕ В.

Тестирование базы данных производилось после реализации основного функционала приложения и симуляции, для более корректного заполнения данными и последующего анализа данных. Заполнение данных в базу данных позволит провести тестирование и убедиться в корректности записываемых данных.

При сохранении первоначально заполняется таблица «AppState», которая хранить наименование и время сохранения (Рисунок 4.2).

Рисунок 4.2 Заполненная таблица «AppState» базы данных

После сохранения симуляции, либо приложения и заполнения таблицы «AppState», заполняется вторая таблицы базы данных «AppServiceState», которая хранит все необходимые данные для будущего восстановления сохранения (Рисунок 4.3).[6]

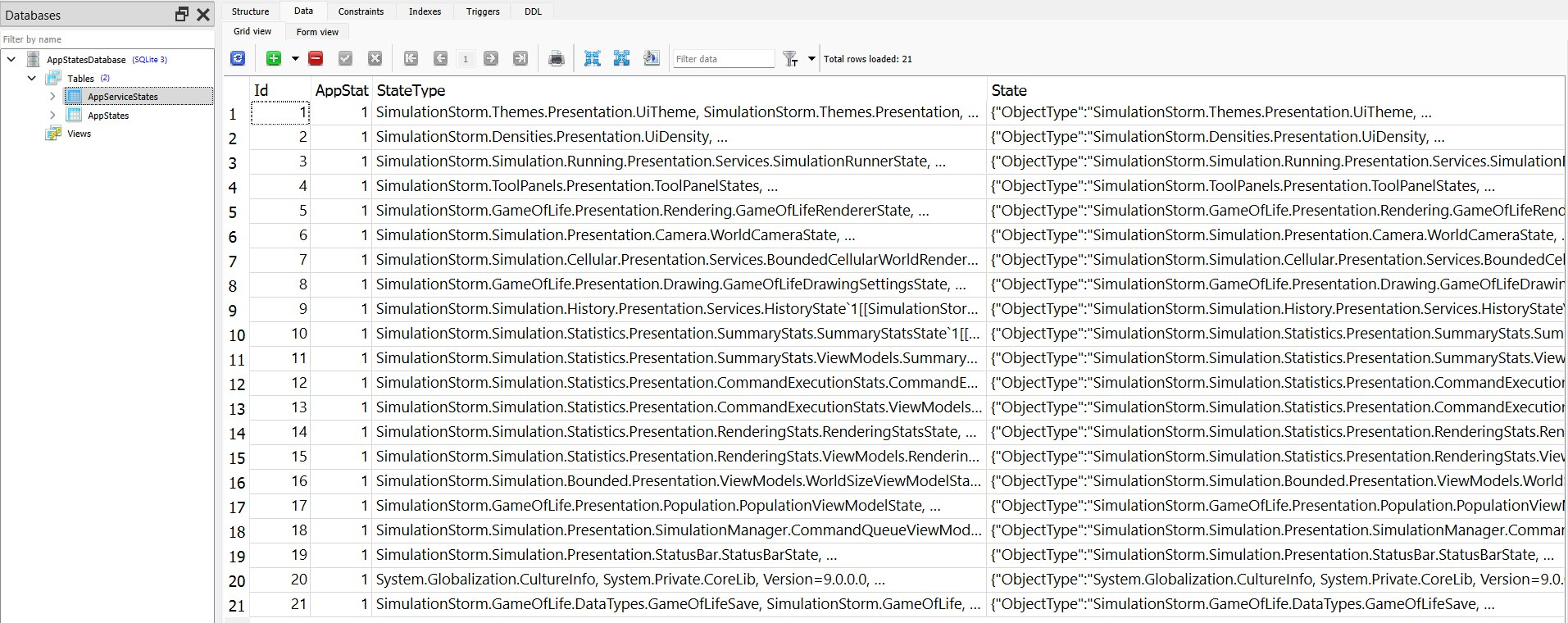


Рисунок 4.3 Заполненная таблица «AppServiceState» базы данных

Данные в таблице «AppServiceState» сериализованны, что позволило хранить различные настройки и состояния сервисов приложения в двух полях базы данных. Это удобно для хранения сложных структур данных, таких как объекты со множеством атрибутов или вложенных объектов, которые неудобно хранить в отдельных столбцах.

Сериализация данных в полях таблицы «AppServiceState», позволило сэкономить пространство в базе данных и упростить доступ к настройкам и состояниям сервисов и других объектов.

Однако, при использовании сериализованных данных потребовалось дополнительно обратить внимание на возможные проблемы с производительностью при извлечении и обновлении данных.

## 4.2 Структура проекта симуляции

Структура проекта представляет собой фиксированное упорядоченное множество объектов и отношений между ними, классифицированных по заданному основанию.

Симуляция «Игра Жизнь», как и большинство модулей инструментария, состоит из трёх основных проектов (Рисунок 4.4):

* проект доменной модели симуляции;
* проект уровня представления;
* проект уровня пользовательского интерфейса.

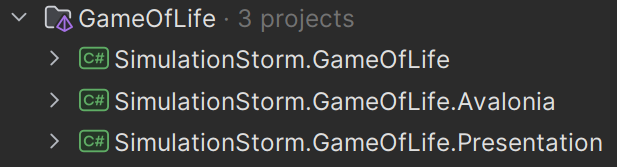


Рисунок 4.4 Состав проектов симуляции «Игра Жизнь»

Первый проект, составляющий симуляцию «Игра Жизнь» — это доменная модель симуляции.

Данный проект содержит компоненты, непосредственно реализующие Игру Жизнь, такие как правило, шаблон, алгоритм, состояние клетки (Рисунок 4.5).

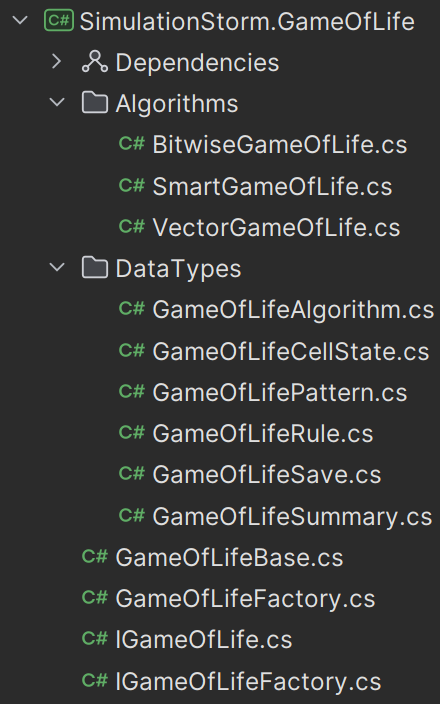


Рисунок 4.5 Проект доменной модели

Второй проект — это проект уровня представления, содержащий независимые от конкретного фреймворка пользовательского интерфейса MVVM-компоненты [7], которые адаптируют доменную модель симуляцию к использованию пользовательским интерфейсом (Рисунок 4.6).

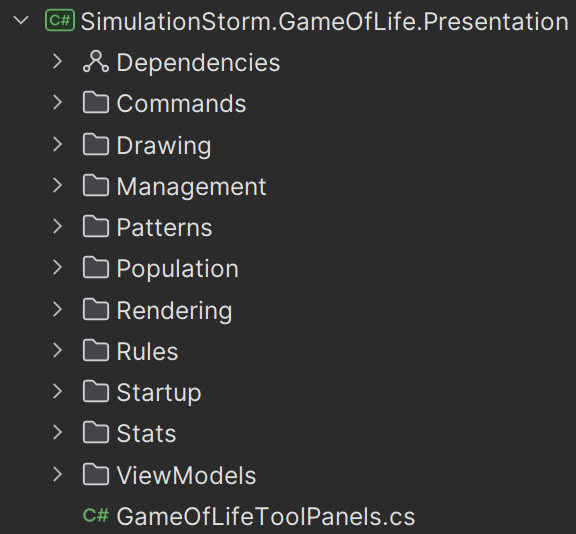


Рисунок 4.6 Проект уровня представления

Третий проект — это проект уровня пользовательского интерфейса. Данный проект использует MVVM-компоненты из проекта уровня представления и реализует пользовательский интерфейс с применением фреймворка Avalonia UI (Рисунок 4.7).

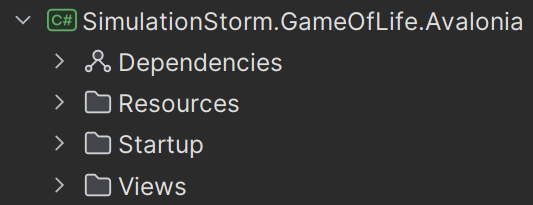


Рисунок 4.7 Проект уровня пользовательского интерфейса

Данная структура проекта «Игра жизнь» обеспечивает эффективное использование ресурсов, а также позволяет создать архитектуру, способствующую достижению высокой читаемости и модульности проекта.

## 4.3 Разработка доменной модели симуляции «Игра Жизнь»

Разработку доменной модели разделена на две части:

* разработка типов данных;
* разработка алгоритмов.

Один из основных типов данных — это перечисление, которое содержит все возможные состояния клеток Игры Жизнь: мёртвая (Dead) и живая (Alive) (Рисунок 4.8).

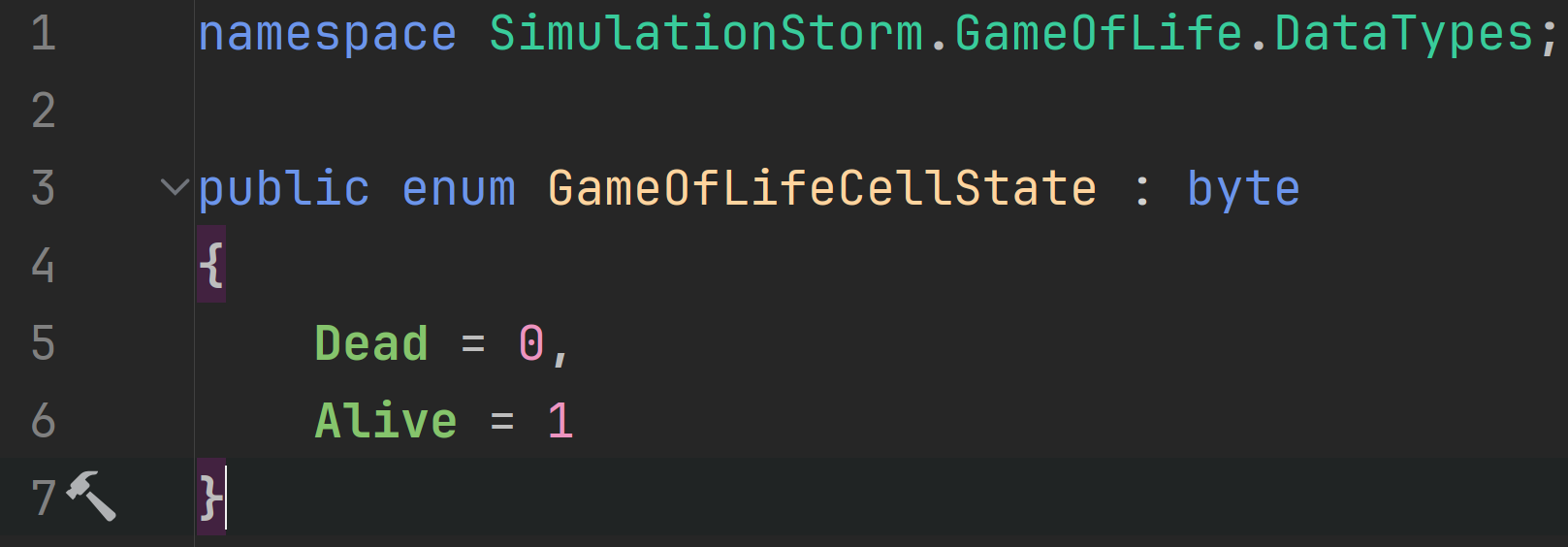


Рисунок 4.8 Перечисление, описывающее возможные состояния клеток

Следующий тип данных — это правило рождения и выживания клеток. Данный тип данных представляет собой класс, который содержит два множества: количество живых соседей, при которых клетка рождается, и количество, при которых живая клетка остаётся живой. Также, данный класс предоставляет ряд методов для валидации создаваемого правила, удобного создания новых правил на основе текущего правила и преобразования правила в строковое представление (Рисунок 4.9).



Рисунок 4.9 Правило рождения и выживания клеток

Для представления шаблонов мёртвых и живых клеток используется одноимённый тип данных. Данный класс содержит размер шаблона и координаты живых клеток в пределах размера, а также предоставляет статический метод для удобного создания экземпляров шаблонов из их строкового представления (Рисунок 4.10).



Рисунок 4.10 Шаблон из мёртвых и живых клеток

Для хранения состояния симуляции при ведении истории, используется тип данных «Сохранение». Данный класс хранит все свойства симуляции «Игра Жизнь», для возможности восстановления этого состояния в будущем (Рисунок 4.11).

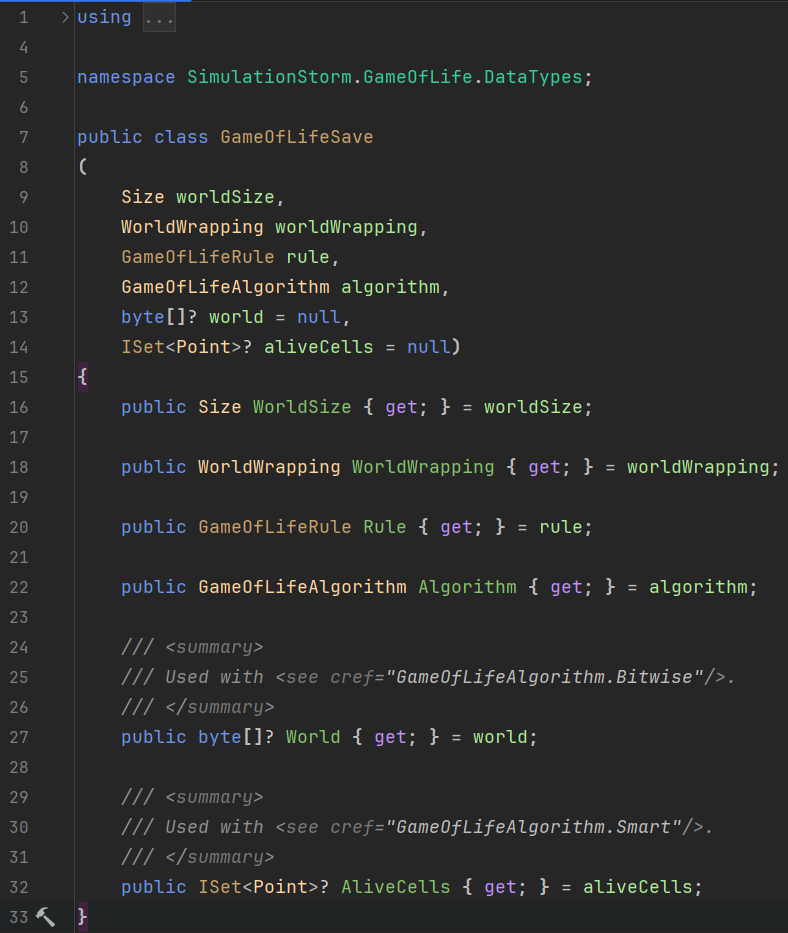


Рисунок 4.11 Сохранение Игры Жизнь

Последний из типов данных — это итоги по текущему состоянию «Игры Жизнь». Данный класс хранит количество мёртвых и живых клеток. Этот тип используется при ведении статистики симуляции (Рисунок 4.12).

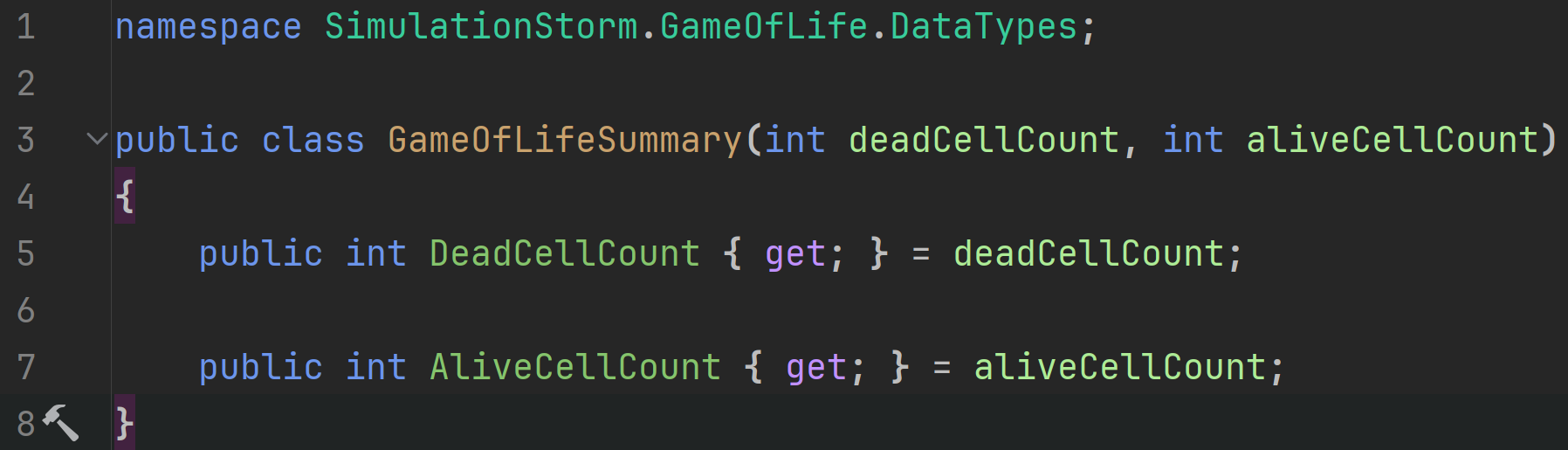


Рисунок 4.12 Итоги по текущему состоянию Игры Жизнь

«Игра Жизнь» может быть реализована с использованием различных алгоритмов. Алгоритм представляет собой механизм, вычисляющий следующее состояние клетки на основе текущего состояния клетки и количества живых соседей этой клетки.

Для всех алгоритмов имеется абстрактный класс, содержащий общие свойства и методы (Рисунок 4.13).

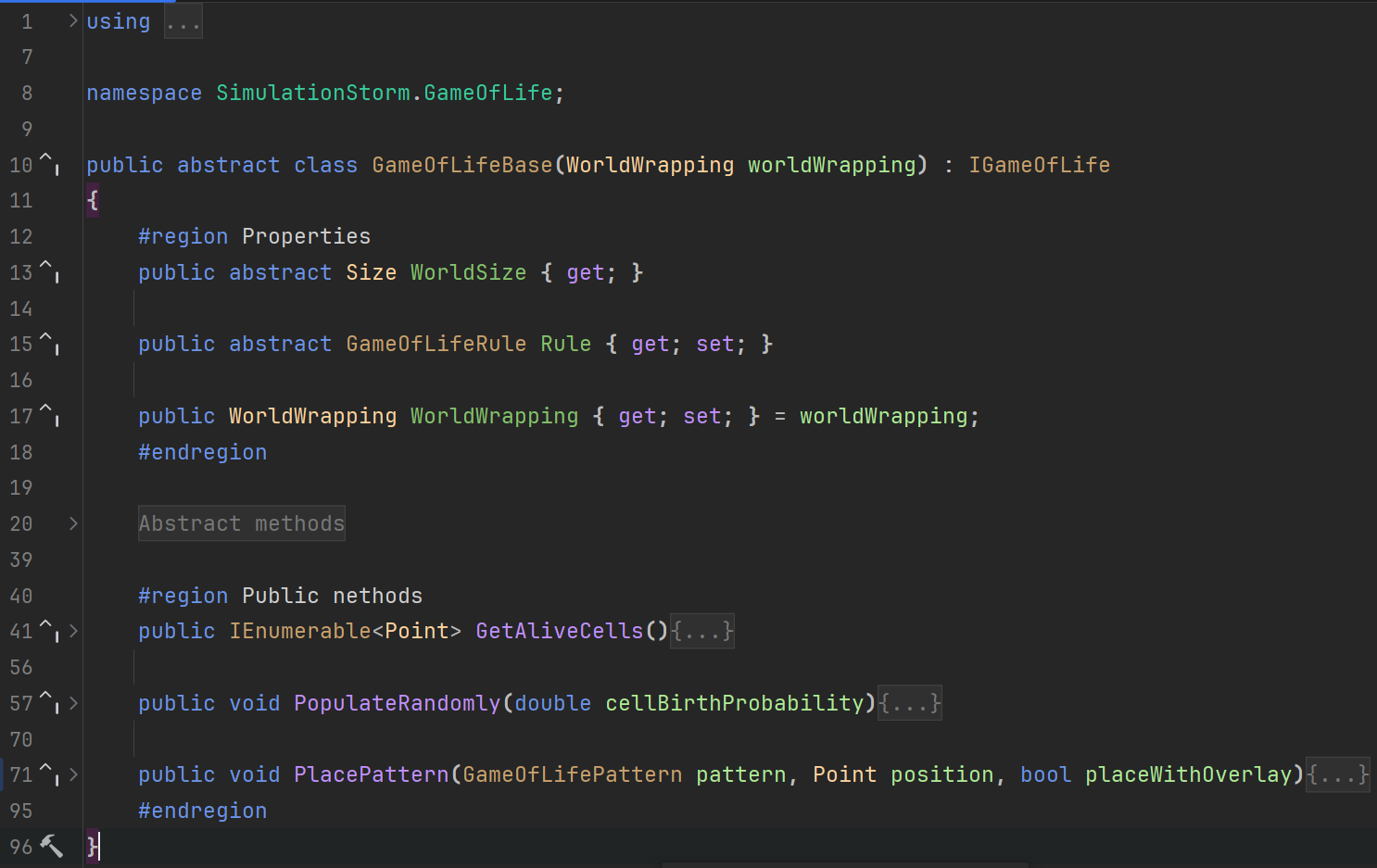


Рисунок 4.13 Абстрактный класс алгоритма Игры Жизнь

На данный момент реализовано два алгоритма: универсальный и умный.

Универсальный алгоритм за проход просчитывает каждую клетку поля симуляции, выполняя восемь вычислений за одну итерацию цикла. Клетки поля представляются в виде массива байтов, где мёртвая клетка — это ноль, живая клетка – единица. Такой массовый просчёт становится возможным благодаря применению операций с указателями (Рисунок 4.14).



Рисунок 4.14 Универсальный алгоритм Игры Жизнь

Умный алгоритм просчитывает только живые клетки поля. В данном алгоритме для представления поля используется множество, которое содержит координаты живых клеток. Этот алгоритм следует использовать в том случае, если размер поля большой, а живых клеток не много (Рисунок 4.15).



Рисунок 4.15 Умный алгоритм Игры Жизнь

Использование данных алгоритмов дает пользователю полный контроль над симуляцией и помогает выбрать требуемый алгоритм исходя из предпочтений и требований пользователя.

# 5 Тестирование программного продукта

## 5.1 Понятие и виды тестирования

Тестирование программного обеспечения — это процесс проверки программного продукта на соответствие его спецификациям, обнаружения ошибок и оценки его качества.

Существует множество различных видов тестирования, каждое из которых направлено на проверку определенных аспектов качества программного продукта.

Наиболее распространенными этапами тестирования является:

* проверка функциональности программы с целью убедиться, что программа работает согласно спецификации;
* оценка стабильности и производительности приложения под различными нагрузками;
* проверка взаимодействия между различными компонентами программы;
* тестирование отдельных модулей или компонентов программы;
* проверка программы на уязвимости и обеспечение защиты от киберугроз;
* проверка работы программы на различных платформах, ОС и браузерах;
* определение скорости работы программы и ее способности к масштабированию;
* оценка удобства использования и внешнего вида пользовательского интерфейса;
* проверка способности программы восстановиться после аварийной ситуации.

Отслеживание и устранение выявленных дефектов позволяют повысить надежность, эффективность и общее качество приложения, обеспечивая пользователей надежным и безопасным инструментом для решения их задач.

## 5.2 Методы тестирования

Основным методом, применяемым в данном проекте, является функциональное тестирование.

Функциональное тестирование — это вид тестирования программного обеспечения, который проверяет, соответствует ли программа предъявленным функциональным требованиям.

В процессе функционального тестирования будут проверяться различные сценарии использования приложения, включая запуск симуляции, изменения правил симуляции, изменения различных настроек приложения, сохранение и вывод статистики и истории протекания симуляции.

Для предотвращения возможных ошибок было применено модульное тестирование.

Модульное тестирование — это вид тестирования, который сосредотачивается на проверке отдельных модулей или компонентов программы для убеждения в их правильной работе. Данный вид тестирования помогает выявить и исправить ошибки на раннем этапе разработки.

В процессе разработки программного обеспечения широко применялось интеграционное тестирование.

Интеграционное тестирование — это процесс проверки взаимодействия между различными компонентами или модулями программного продукта для убеждения в правильной работоспособности системы в целом.

Данный вид тестирования необходим для выявления возможных ошибок и несоответствий, которые могут возникнуть при взаимодействии различных частей приложения, таких как модули, классы или сервисы. В процессе интеграционного тестирования также оценивается работоспособность различных элементов приложения как единого целого, что позволяет выявить проблемы, связанные с их взаимодействием.

Проведение интеграционного тестирования после завершения модульного тестирования позволяет своевременно выявлять и исправлять потенциальные ошибки взаимодействия между компонентами приложения.

Кроме того, успешное интеграционное тестирование способствует улучшению стабильности, надежности и производительности программного продукта.

Для обеспечения высокой стабильности и надежности программного продукта важным методом тестирования является нагрузочное тестирование.

Нагрузочное тестирование — это процесс проверки работы системы при условиях экстремально высокой нагрузки, для оценки ее производительности, устойчивости и способности к масштабированию, что помогает выявить потенциальные узкие места и проблемы, которые могут возникнуть в процессе реальной эксплуатации.

Последующий анализ результатов позволяет корректно и своевременно устранить выявленные уязвимости, оптимизировать производительность приложения и обеспечить бесперебойную работу в условиях высоких нагрузок.

## 5.3 Тестирование приложения с использованием тест-кейсов

Для тестирования функциональности реализованного приложения были использованы тест-кейсы.

Тест-кейс — это шаги, предусловия, ожидаемые результаты и другие детали, необходимые для проведения конкретного тестирования части программного обеспечения или функциональности.[8]

В процессе тестирования были использованы тест-кейсы для проверки основных функций приложения и симуляции. Каждый тест-кейс включает в себя последовательность шагов, которые необходимо выполнить, и ожидаемый результат.

В процессе проведения тестирования, каждый отдельный тест-кейс подвергался тщательной ручной проверке, что было необходимо для обеспечения полного соответствия между фактическим поведением программного продукта в ходе его эксплуатации и теми результатами, которые были изначально запланированы и ожидались.

Тестовый сценарий №1 служит для проверки работоспособности механизма установки выбранного пользователем шаблона мёртвых и живых клеток на поле симуляции (Таблица 5.1).

Таблица 5.1 – Тест кейс №1

|  |  |
| --- | --- |
| Test Case № | 1 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции расположения шаблона на поле |
| Резюме испытания | Проверка того, что при выборе шаблона в панели рисования и последующем нажатии по клетке мира, выбранный шаблон будет размещён на поле |
| Шаги тестирования | 1. открыть панель рисования;  2. выбрать произвольный шаблон;  3. кликнуть по клетке поля. |
| Данные тестирования | - |
| Ожидаемый результат | Выбранный шаблон должен быть размещён на поле |
| Фактический результат | Выбранный шаблон был размещён на поле |
| Предпосылки | – |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Механизм установки шаблонов мертвых и живых клеток на поле симуляции является важным компонентом пользовательского опыта при работе с программой. Данный функционал предоставляет пользователю возможность влиять на начальную конфигурацию клеток в процессе симуляции клеточного автомата. Выбор и установка шаблонов имеют значительное влияние на последующее развитие игрового процесса. Они позволяют создавать интересные комбинации клеток и их поведения, а также оказывают воздействие на сложность и динамику происходящих в симуляции событий.

Таким образом, механизм установки шаблонов клеток играет ключевую роль в формировании уникального игрового опыта и предоставляет пользователю возможность экспериментировать с различными вариантами стартовых условий.

Тестовый сценарий №2 проверяет работоспособность механизма замыкания поля симуляции (Таблица 5.2).

Таблица 5.2 – Тест кейс №2

|  |  |
| --- | --- |
| Test Case № | 2 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции замыкания поля |
| Резюме испытания | Проверка того, что при установке замкнутости поля по горизонтали в панели симуляции, шаблоны, движущиеся слева направо, появятся с правой стороны поля |
| Шаги тестирования | 1. открыть панель симуляции;  2. в разделе настроек замыкания поля, установить замкнутость по горизонтали;  3. расположить на левом краю поля шаблон «космический корабль»;  4. запустить продвижение симуляции;  5. спустя несколько шагов, «космический корабль» должен появиться с правой стороны поля. |
| Данные тестирования | - |
| Ожидаемый результат | Шаблон «космический корабль» должен появиться с правой стороны поля |
| Фактический результат | Шаблон «космический корабль» появился с правой стороны поля |
| Предпосылки | Размер поля должны быть больше размера шаблона «космический корабль» |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Когда поле замкнуто, например, по горизонтали, это означает, что, когда объект достигает одного края поля в замкнутой среде, он сразу же появляется на противоположном краю, создавая иллюзию непрерывности и продолжения движения. Такой механизм обеспечивает более плавное и непрерывное движение шаблонов на поле симуляции и помогает сохранить целостность игрового процесса в замкнутой среде.

Тестовый сценарий №3 служит для проверки работоспособности механизма заселения поля живыми клетками с заданной вероятностью (Таблица 5.3).

Таблица 5.3 – Тест кейс №3

|  |  |
| --- | --- |
| Test Case № | 3 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции случайного заселения поля |
| Резюме испытания | Проверка того, что при выполнении команды «заселить поле» в панели симуляции, поле действительно будет заселено живыми клетками с заданной вероятностью |
| Шаги тестирования | 1. открыть панель симуляции;  2. перейти в раздел заселения;  3. задать вероятность появления живой клетки;  4. кликнуть по кнопке «заселить поле»;  5. процент живых клеток на круговой диаграмме статистики симуляции должен стать равен заданной вероятности |
| Данные тестирования | вероятность рождения клетки – 50% |
| Ожидаемый результат | Поле должно быть заселено живыми клетками с заданной вероятностью |
| Фактический результат | Поле было заселено живыми клетками с указанной вероятностью |
| Предпосылки | 1. поле симуляции должно быть пустым;  2. должна быть открыта круговая диаграмма статистики симуляции |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Эта функция играет важную роль в исследовании симуляции и поиске новых шаблонов. Путем применения различных правил и параметров на больших масштабах, она позволяет отслеживать изменения в поведении симуляции и обнаруживать интересные закономерности и взаимосвязи между элементами симуляции.

Тестовый сценарий №4 проверяет функцию изменения размера поля симуляции в соответствие заданному размеру (Таблица 5.4).

Таблица 5.4 – Тест кейс 4

|  |  |
| --- | --- |
| Test Case № | 4 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции изменения размера поля |
| Резюме испытания | Проверка того, что при задании размеров поля и выполнении команды изменения размера поля, размер поля действительно будет изменён |
| Шаги тестирования | 1. открыть панель симуляции;  2. перейти в раздел настройки размера поля;  3. задать новый размер поля;  4. кликнуть по кнопке «изменить размер поля». |
| Данные тестирования | новый размер поля – 1000 x 1000 клеток |
| Ожидаемый результат | Размер поля должен быть изменён на заданный |
| Фактический результат | Размер поля был изменён в соответствии с заданным |
| Предпосылки | – |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Данная функция — ключевой инструмент клеточных симуляций, который обеспечивает возможность выбора оптимального размера поля для проведения экспериментов. Путем корректного определения размеров поля и параметров симуляции данная функция способствует созданию условий для точного моделирования поведения клеток и взаимодействия между ними.

Выбор правильного размера поля играет важную роль в обеспечении достоверности результатов симуляции и их пригодности для дальнейшего анализа.

Благодаря этой функции исследователи могут проводить эксперименты в контролируемых условиях, что позволяет более точно и надежно изучать процессы, происходящие в клеточной среде.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 6.1 Запуск симуляции с использованием Лаунчера

При запуске приложения пользователя встречает основное окно, в котором предоставляется выбор (Рисунок 6.1): языка интерфейса, темы приложения, плотность интерфейса и выбор симуляции, которая будет запущена.

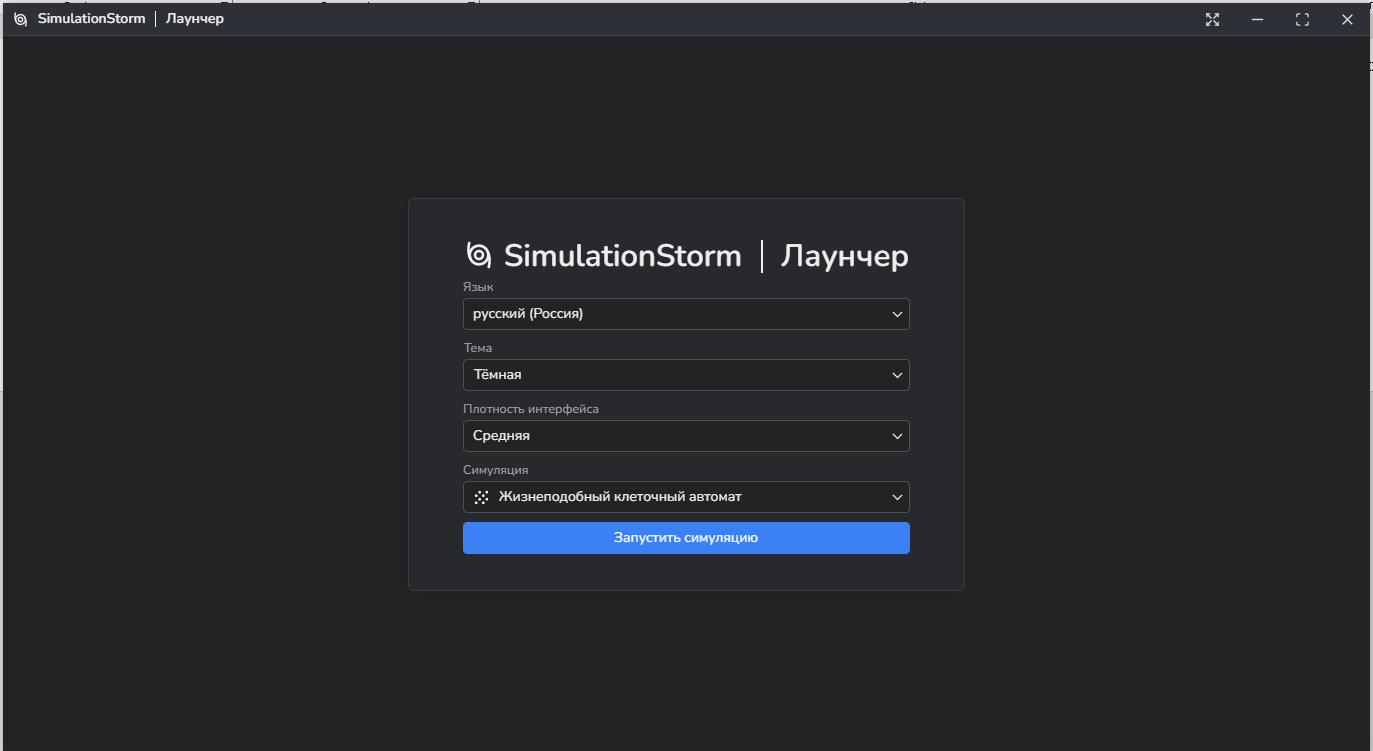


Рисунок 6.1 Лаунчер приложения

Выбор симуляции ориентирован на будущих разработчиков, заинтересованных в проекте. Разработчики могут легко создавать свои собственные симуляции, используя предоставленный инструментарий, без лишних затруднений. Это означает, что разработчики могут быстро начать работу над своими проектами, не тратя много времени на изучение сложных инструментов или процессов.

Предоставленный инструментарий обеспечивает простой и интуитивно понятный интерфейс, который позволяет разработчикам сосредоточиться на создании самой симуляции, а не на том, как использовать инструменты для её разработки.

Такой подход способствует быстрой и эффективной работе, стимулируя творческий процесс и ускоряя достижение желаемых результатов.

При разработке приложения и симуляции были приняты меры для защиты пользователей от возможных негативных последствий быстрых изменений цветов и ярких изображений. После запуска приложения появляется предупреждение, которое можно закрыть, нажав на кнопку «Space» (Рисунок 6.2).



Рисунок 6.2 Сообщение с предупреждение

Предупреждение, которое появляется перед пользователем, выполняет две важные функции. Во-первых, оно информирует пользователя о возможных последствиях его выбора, будь то выбор определенной симуляции или параметров. Это помогает предотвратить случайные ошибки, которые могут привести к нежелательным результатам.

Во-вторых, предупреждение служит удобным инструментом для пользователя. Если он обнаружил, что сделал ошибку в своем выборе, или просто хочет изменить свой выбор, ему достаточно нажать на кнопку «Esc», чтобы вернуться к предыдущему окну лаунчера. Это позволяет пользователю контролировать процесс запуска симуляции и вносить необходимые корректировки.

Такие меры обеспечивают удобство использования приложения, предоставляя пользователям контроль над процессом запуска симуляции.

## 6.2 Окно запущенной симуляции

После согласия с предупреждением пользователя встречает основное окно симуляции, где предоставляется игровое поле и разнообразные настройки приложения и симуляции (Рисунок 6.3).

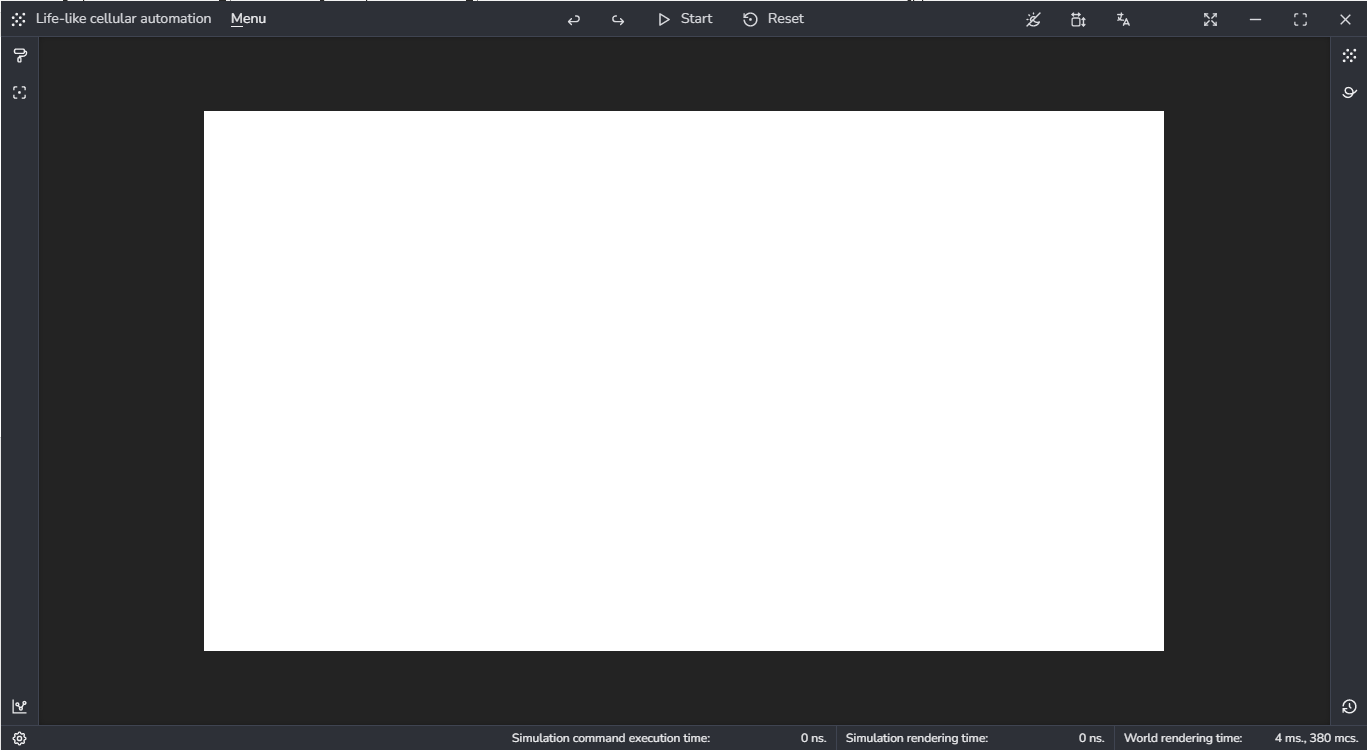


Рисунок 6.3 Окно симуляции

В верхней левой части приложения представлен логотип с названием приложения (Рисунок 6.4).

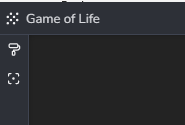


Рисунок 6.4 Логотип с названием

Пользователь имеет возможность удобно и быстро узнать какая симуляция запущена, взглянув на название и логотип приложения.

В верхней средней части окна предоставлены основные кнопки управления симуляций (Рисунок 6.5).



Рисунок 6.5 Основные кнопки управления симуляцией

Запуск симуляции осуществляется после нажатия на кнопку «Start», отмеченной зеленым цветов (Рисунок 6.6).



Рисунок 6.6 Кнопка запуска симуляции

Возврат на одну итерацию назад, либо вперед осуществляется двумя кнопками, располагающимися с левой стороны от кнопки запуска (Рисунок 6.7).



Рисунок 6.7 Кнопки переходов по итерациям назад, либо вперёд

Так же имеется возможность очистить поле симуляции нажатием кнопки «Reset» (Рисунок 6.8).



Рисунок 6.8 Кнопка очистки поля симуляции

Кнопки управления размерами окна приложения предоставлены же в верхней правой части приложения (Рисунок 6.9).



Рисунок 6.9 Кнопки управления окном приложением

Все кнопки, которые были упомянуты ранее, предоставляют пользователю удобную и интуитивно понятную возможность управления как самим приложением, так и симуляцией.

Основные настройки симуляции предоставлены с правой стороны (Рисунок 6.10).



Рисунок 6.10 Основные настройки симуляции

Для удобной и комфортной работы в приложении в нижней части приложения демонстрируется статистика приложения, а также кнопка истории симуляции (Рисунок 6.11).

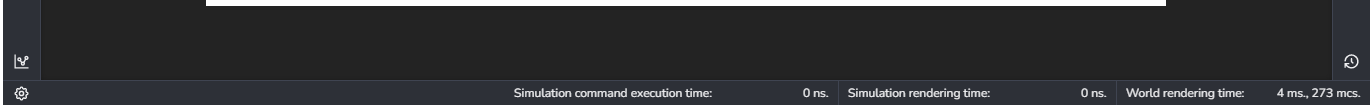


Рисунок 6.11 Нижняя панель со статистикой и с кнопкой истории

В статистике отображается основные параметры, такие как: время отрисовки симуляции, время отрисовки поля симуляции, время, потраченное на выполнение команд. Данная статистика поможет разработчикам более точно определить в каком именно моменты происходят зависания симуляции и предположить с чем это может быть связано (Рисунок 6.12).



Рисунок 6.12 Основная статистика приложения

Так же стоит упомянуть время отрисовки мира, она располагается в нижней правой части панели и демонстрирует точное время в миллисекундах (Рисунок 6.13).



Рисунок 6.13 Время отрисовки мира

Данное время предоставляет комфортные условия для тестирования созданного проекта, позволяя при этом отслеживать ресурсы и время, затраченные на отрисовку всего мира.

Настройка поля и работа с камерой в приложении осуществляется в левой панели (Рисунок 6.14).

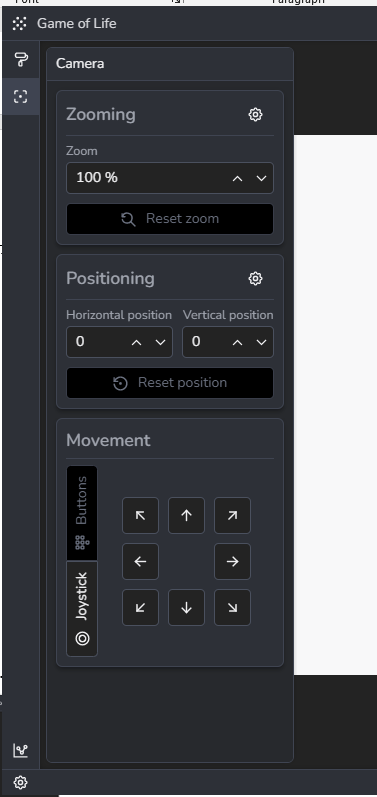


Рисунок 6.14 Левая панель приложения с настройками камеры и поля

Модули интерфейса, отвечающие за управление камерой, играют важную роль в обеспечении комфортного и точного взаимодействия с ней. Данные элементы помогают пользователям легко контролировать позиционирование камеры и настраивать ее параметры с необходимой точностью.

Для более комфортного управления имеется возможность использовать кнопки, либо использовать джойстик (Рисунок 6.15).



Рисунок 6.15 Джойстик, предоставляющий возможность управлять камерой

Так же имеется возможность выставить самостоятельно приближение камеры в процентах, сбросить данное значение кнопкой «Reset zoom», а также управлять камерой используя числа по горизонтали и вертикали (Рисунок 6.16).

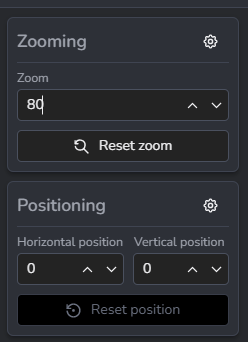


Рисунок 6.16 Управление приближением и передвижением камеры

Грамотное использование данных модулей интерфейса способствует более эффективному взаимодействию с камерой и повышает пользовательский опыт при использовании приложения.

Поле симуляции расположено в средней части приложения (Рисунок 6.17).

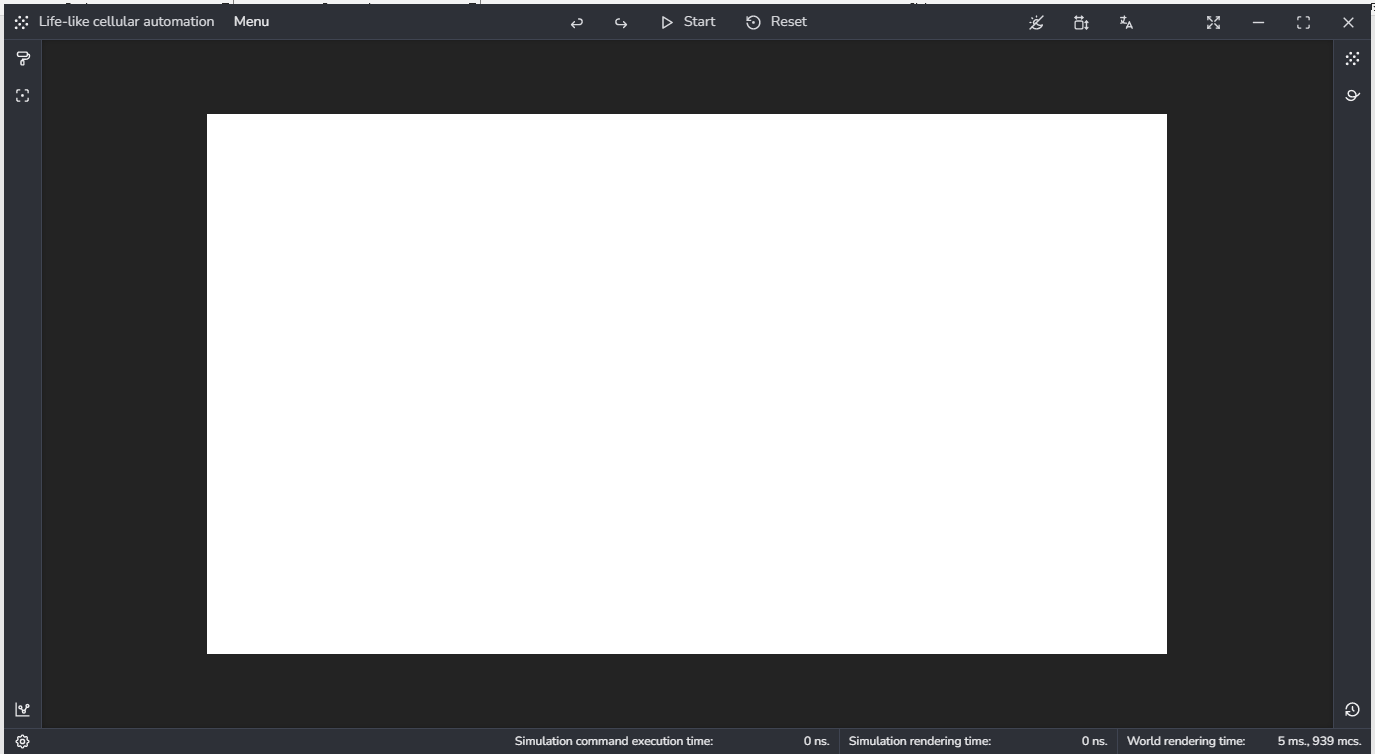


Рисунок 6.17 Поле симуляции

Так же стоит упомянуть настройки приложения: язык, тему и плотность интерфейса (Рисунок 6.18).

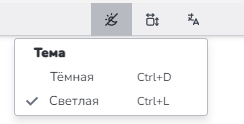


Рисунок 6.18 Настройки стиля приложения

Данные настройки позволяют сделать приложение дружелюбным под каждое устройство с разными размерами экрана и разными вкусами пользователей.

## 6.3 Запуск и настройка симуляции

Перед запуском симуляции стоит выбрать необходимый шаблон, либо нарисовать собственные фигуры, включив режим рисования, данные возможности представлены в меню рисования в правой части приложения (Рисунок 6.19).

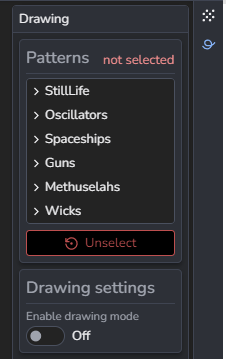


Рисунок 6.19 Меню шаблонов и включения рисования на поле симуляции

Выбор подходящего шаблона или создание уникальных фигур с помощью функции рисования позволяет настраивать симуляцию в соответствии с потребностями пользователя. Это дает возможность создавать индивидуальные сценарии, адаптированные под конкретные условия или задачи.

Использование меню рисования обеспечивает удобный и интуитивно понятный способ настройки элементов симуляции, что способствует более эффективному и гибкому взаимодействию с приложением.

После выбора необходимого шаблона, либо отрисованной вручную фигур, так же имеется возможность изменить настройки симуляции, которые так же находятся с левой стороны приложения (Рисунок 6.20).

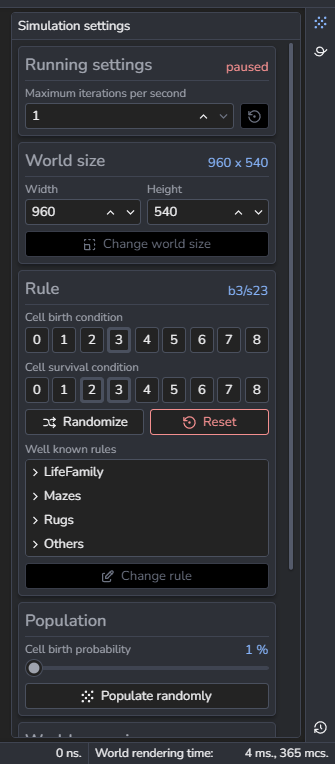


Рисунок 6.20 Меню настроек симуляции

Настройки симуляции предоставляют широкие возможности для изменения и настройки симуляции. Они позволяют пользователю глубоко влиять на ход и параметры симуляции, делая её более персонализированной и соответствующей индивидуальным предпочтениям.

Благодаря настройкам, пользователь может управлять всеми аспектами симуляции, включая начальные условия, поведение объектов, взаимодействия и многое другое. Это дает возможность создавать уникальные сценарии и адаптировать симуляцию под конкретные интересы каждого пользователя.

В меню настроек симуляции имеется возможность изменить правила запускаемой, либо запущенной симуляции (Рисунок 6.21).

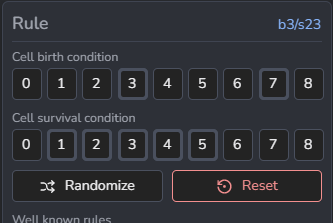


Рисунок 6.21 Настройки правил симуляции

Так же для более комфортного и гибкого изучения запущенной симуляции имеется возможность настроить скорость отрисовки симуляции (Рисунок 6.22).

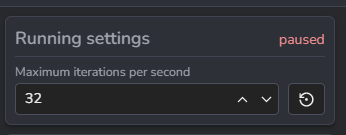


Рисунок 6.22 Настройка скорости отрисовки симуляции

После настройки скорость и выбора правил симуляции следует выбрать размеры поля симуляции (Рисунок 6.23).

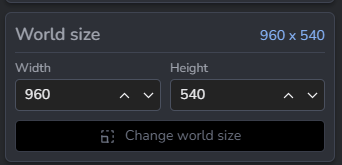


Рисунок 6.23 Настройка размеров поля симуляции

Данная настройка позволяет быстро и удобно изменить размеры поля симуляции, вписав свои значения либо используя кнопки уменьшения, либо увеличения значения поля.

Для пользователей, которые не хотят самостоятельно изучить разные правила симуляции и изменять её с целью изучения, имеется список ранее разработанных шаблонов правил, которые были поделены на разные группы (Рисунок 6.24).

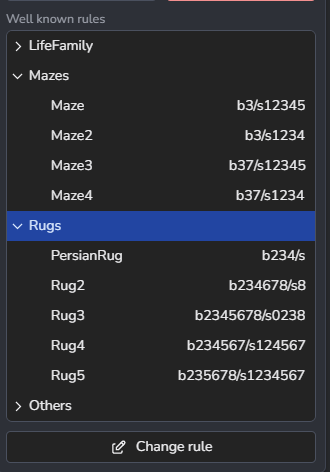


Рисунок 6.24 Шаблоны правил

Пользователь может заполнить поле в случайном порядке, не рисуя произвольные фигуры и не выбирая шаблоны фигур нажав на кнопку «Populate rendomly» (Рисунок 6.25).

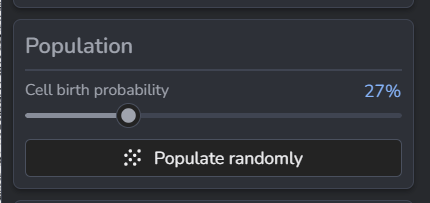


Рисунок 6.25 Заполнения поля симуляции в случайном порядке

Так же имеется возможность изменить плотность заполнения поля симуляции используя ползунок, выбранную плотность можно узнать в правой части модуля в процентном соотношении.

Имеется возможность зациклить поле симуляции по горизонтали либо по вертикали, при выборе данных настроек клетки имеют возможности перейти на противоположную сторону при запуске симуляции (Рисунок 6.26).

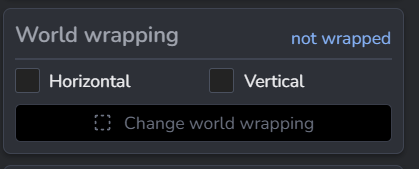


Рисунок 6.26 Настройка цикличности поля симуляции

Важной настройкой симуляции является выбор алгоритма отрисовки, которая сильно влияет на время отрисовки той или иной фигуры (Рисунок 6.27).

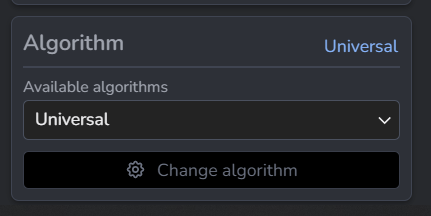


Рисунок 6.27 Настройка выбора алгоритма симуляции

В данной настройке симуляции имеется возможность выбора двух видов алгоритма: Universal и Smart.

Алгоритм Universal предназначен для обработки большого объёма данных и больше подходит для огромных полей симуляции, в данном алгоритме подсчет состояния клеток производится на основании живых, не затрагивая не живые клетки.

Алгоритм Smart предназначен для быстрой обработки малых объёмов данных и наиболее сильно подходит в случае, если поле симуляции имеется маленькие размеры, подсчет состояния клеток в данном случае будет производится на основании всех клеток поля и будут приняты в расчет как живые, так и не живые клетки.

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта было разработано демонстрационное симуляция «Игра Жизнь». Данный проект был спроектирован с целью продемонстрировать возможности ранее разработанного инструментария, а также популяризовать клеточные автоматы.

Основной функционал приложения:

* + пользователь имеет возможность выбирать шаблон из ранее созданного списка и добавить его на поле симуляции;
* пользователь имеет возможность изменять правила симуляции, либо выбирать определенные правила из предложенного списка, что позволяет настраивать симуляцию по своему усмотрению;
* приложение предоставляет пользователю возможность ознакомиться со статистикой выполняемой симуляции, что помогает отслеживать результаты и ход выполнения симуляции;
* пользователь имеет возможность просматривать историю симуляции, что позволяет следить за ходом протекания симуляции;
* с помощью кнопок и джойстика пользователь может управлять камерой, что помогает отслеживать протекания то или иной области поля симуляции;
* пользователь имеет возможность запускать, останавливать или очищать поле симуляции по своему усмотрению;
* пользователь имеет возможность настраивать различные параметры приложения, такие как плотность интерфейса и тему приложения;
* пользователь может легко переключаться между различными языками интерфейса приложения для удобства использования;
* имеется возможность изменения алгоритма расчета следующего хода симуляции, для более корректного и точного тестирования и анализа симуляции;
* имеется возможность сохранения статистики и истории симуляции.

Процесс создания демонстрационного проекта «Игра жизнь» включал в себя глубокий анализ предметной области и аналогов, что позволило создать приложение, которые целиком смог показаться возможности реализованного инструментария и привлечь ранее не заинтересованных пользователей в сферу разработки и изучения клеточных автоматов.

Симуляции подобные симуляции «Игра Жизнь» может быть полезно в различных областях науки и техники. Например, он может использоваться для моделирования биологических, физических и социальных процессов.

Примеры использования симуляции:

* в биологии может быть использована для моделирования роста и развития биологических организмов, популяций, экосистем и других биологических процессов;
* В физике может быть использована для моделирования физических процессов, таких как распространение волн, диффузия, теплопередача и другие.
* В социологии и экономике может быть использована для моделирования социальных процессов, таких как миграция населения, экономическое развитие, изменение социальных норм и ценностей и другие.
* В информатике и компьютерных науках может быть использована для обучения алгоритмам и структурам данных, а также для исследования сложности вычислений и параллельных вычислений.
* В искусстве и дизайне может быть использована для создания визуальных эффектов и анимации, а также для генерации новых форм и структур.
* В образовании может быть использована для обучения основам математики, информатики и других наук, а также для развития критического мышления и творческого подхода к решению задач.

Подобные симуляции имеют широкий спектр возможностей применения и могут быть полезны в различных областях науки, техники и искусства.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Возможности .Net [Электронный ресурс]. Режим доступа: https://checkroi.ru/blog/, свободный.
2. Инструкция по работе с платформой Avaloni [Электронный ресурс]. Режим доступа: <https://docs.avaloniaui.net/docs/welcome>, свободный.
3. Справочник по SQL [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/SQL>, свободный.
4. Изучение Rider [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/ru-ru/academy/#learn, свободный.
5. Инструкция по работе со сценариями использования [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/699522/, свободный.
6. Типы данных в SQL [Электронный ресурс]. Режим доступа: https://metanit.com/sql/sqlite/2.2.php, свободный.
7. Инструкция по работе с шаблоном проектирования MVVM [Электронный ресурс]. Режим доступа: <https://habr.com/ru/articles/338518/>, свободный.
8. Use case [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/699522/, свободный.
9. Общие сведения о Visual Studio | Microsoft Learn [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022, свободный.
10. Общие свесдения о Visual Studio | Microsoft Learn [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022, свободный.
11. GitHub [Электронный ресурс]. Режим доступа: https://github.com/, свободный.
12. Программное-обеспечение [Электронный ресурс]. Режим доступа: https://creately.com/ru/lp, свободный.
13. Программное-обеспечение [Электронный ресурс]. Режим доступа: https://qna.habr.com/q/292075, свободный.
14. Entity Framework Core [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/ef/core/>, свободный.
15. ASP.NET Core [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/aspnet/core/>, свободный.
16. Транзакции в SQL Server [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/transactions-transact-sql/>, свободный.
17. Microsoft SQL Server Management Studio (SSMS) [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/sql/ssms/>, свободный.
18. Хранимые процедуры и функции в SQL Server [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/sql/relational-databases/stored-procedures/overview-of-stored-procedures/>, свободный.
19. Руководство работы с C# [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/, свободный.
20. Разработка на SQLite [Электронный ресурс]. Режим доступа: https://sqlite.org/, свободный.

# ПРИЛОЖЕНИЕ А

Техническое задание.

Данный документ представляет собой техническое задание, которое определяет требования и задачи для проектирования и последующей разработки демонстрационного проекта под названием «Игра Жизнь».

Основная цель этого проекта заключается в демонстрации полного функционала, который был ранее разработан и внедрен в инструментарий.

Требования к функциональности информационной системы:

* проектирование и создание базы данных;
* обеспечение высокой производительности системы;
* гарантированное корректное сохранение всего приложения в базе данных;
* предоставление удобного и интуитивно понятного интерфейса для пользователей;
* все диаграммы, представленные в проекте, корректно отображают данные воспроизводящиеся во время выполнения симуляции;
* симуляция полностью и корректно следует заданным правилам;
* гарантированное корректное сохранение истории и статистики в файловую или оперативную память системы.

Требования к надежности информационной системы:

* система должна обеспечивать непрерывную работу и быть всегда функциональной;
* в случае возникновения неполадок необходимо уведомить пользователя.

Архитектура информационной системы:

* проект уровня доменной модели симуляции;
* проект уровня представления;
* проект уровня пользовательского интерфейса.

Пользовательский интерфейс разрабатываемого приложения базируется на использовании двух ключевых библиотек: «SkiaSharp» и «Actipro».

Первая из них предоставляет инструменты для рендеринга 2D графики, а вторая - набор компонентов для создания удобного и интуитивно понятного интерфейса.

Для работы с данными в приложении используется система управления базами данных SQLite. Эта система позволяет эффективно хранить и обрабатывать данные, обеспечивая при этом высокую степень безопасности и производительности.

Важным этапом разработки любого программного продукта является его тестирование.

Тестирование приложения включает в себя несколько этапов.

В первую очередь следует проводит тестирование всех компонентов приложения на различных стадиях разработки, что позволяет выявить возможные ошибки и недочеты на ранних этапах.

После проверяется работоспособность приложения на разных уровнях нагрузки. Это позволяет убедиться в том, что приложение способно стабильно работать даже при высоких нагрузках.

Данное техническое задание представляет собой основу для проектирования и разработки проекта под названием «Игра жизни». При этом предполагается использование уже существующих общих модулей и инструментария, что позволит ускорить процесс разработки и избежать дублирования усилий.

# ПРИЛОЖЕНИЕ Б

Словарь данных

Таблица 1.1 – сущность «AppStates»

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип данных | Допускается значение NULL | Вид ключа |
| Id | int | NO | PK |
| DateAndTime | smalldatetime | NO |  |
| Name | nvarchar(25) | NO |  |

Таблица 1.2 – сущность «AppServiceState»

|  |  |  |  |
| --- | --- | --- | --- |
| Название поля | Тип данных | Допускается значение NULL | Вид ключа |
| Id | int | NO | PK |
| StateObjectType | nvarchar(MAX) | NO |  |
| StateObject | nvarchar(MAX) | NO |  |
| AppStateId | int | NO | FK |

# ПРИЛОЖЕНИЕ В

Текст скрипта на создание таблиц на языке SQL

USE [master]

GO

/\*\*\*\*\*\* Object: Database [Stuff] Script Date: 10.06.2024 12:24:19 \*\*\*\*\*\*/

CREATE DATABASE [Stuff]

CONTAINMENT = NONE

ON PRIMARY

( NAME = N'Stuff', FILENAME = N'C:\Users\Ильназ\Stuff.mdf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )

LOG ON

( NAME = N'Stuff\_log', FILENAME = N'C:\Users\Ильназ\Stuff\_log.ldf' , SIZE = 8192KB , MAXSIZE = 2048GB , FILEGROWTH = 65536KB )

WITH CATALOG\_COLLATION = DATABASE\_DEFAULT

GO

ALTER DATABASE [Stuff] SET COMPATIBILITY\_LEVEL = 150

GO

IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))

begin

EXEC [Stuff].[dbo].[sp\_fulltext\_database] @action = 'enable'

end

GO

ALTER DATABASE [Stuff] SET ANSI\_NULL\_DEFAULT OFF

GO

ALTER DATABASE [Stuff] SET ANSI\_NULLS OFF

GO

ALTER DATABASE [Stuff] SET ANSI\_PADDING OFF

GO

ALTER DATABASE [Stuff] SET ANSI\_WARNINGS OFF

GO

ALTER DATABASE [Stuff] SET ARITHABORT OFF

GO

ALTER DATABASE [Stuff] SET AUTO\_CLOSE OFF

GO

ALTER DATABASE [Stuff] SET AUTO\_SHRINK OFF

GO

ALTER DATABASE [Stuff] SET AUTO\_UPDATE\_STATISTICS ON

GO

ALTER DATABASE [Stuff] SET CURSOR\_CLOSE\_ON\_COMMIT OFF

GO

ALTER DATABASE [Stuff] SET CURSOR\_DEFAULT GLOBAL

GO

ALTER DATABASE [Stuff] SET CONCAT\_NULL\_YIELDS\_NULL OFF

GO

ALTER DATABASE [Stuff] SET NUMERIC\_ROUNDABORT OFF

GO

ALTER DATABASE [Stuff] SET QUOTED\_IDENTIFIER OFF

GO

ALTER DATABASE [Stuff] SET RECURSIVE\_TRIGGERS OFF

GO

ALTER DATABASE [Stuff] SET DISABLE\_BROKER

GO

ALTER DATABASE [Stuff] SET AUTO\_UPDATE\_STATISTICS\_ASYNC OFF

GO

ALTER DATABASE [Stuff] SET DATE\_CORRELATION\_OPTIMIZATION OFF

GO

ALTER DATABASE [Stuff] SET TRUSTWORTHY OFF

GO

ALTER DATABASE [Stuff] SET ALLOW\_SNAPSHOT\_ISOLATION OFF

GO

ALTER DATABASE [Stuff] SET PARAMETERIZATION SIMPLE

GO

ALTER DATABASE [Stuff] SET READ\_COMMITTED\_SNAPSHOT OFF

GO

ALTER DATABASE [Stuff] SET HONOR\_BROKER\_PRIORITY OFF

GO

ALTER DATABASE [Stuff] SET RECOVERY SIMPLE

GO

ALTER DATABASE [Stuff] SET MULTI\_USER

GO

ALTER DATABASE [Stuff] SET PAGE\_VERIFY CHECKSUM

GO

ALTER DATABASE [Stuff] SET DB\_CHAINING OFF

GO

ALTER DATABASE [Stuff] SET FILESTREAM( NON\_TRANSACTED\_ACCESS = OFF )

GO

ALTER DATABASE [Stuff] SET TARGET\_RECOVERY\_TIME = 60 SECONDS

GO

ALTER DATABASE [Stuff] SET DELAYED\_DURABILITY = DISABLED

GO

ALTER DATABASE [Stuff] SET ACCELERATED\_DATABASE\_RECOVERY = OFF

GO

ALTER DATABASE [Stuff] SET QUERY\_STORE = OFF

GO

USE [Stuff]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AppServiceState] Script Date: 10.06.2024 12:24:19 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AppServiceState](

[Id] [int] NOT NULL,

[StateObjectType] [nvarchar](max) NOT NULL,

[StateObject] [nvarchar](max) NOT NULL,

[AppStateId] [int] NOT NULL,

CONSTRAINT [PK\_AppServiceState] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AppState] Script Date: 10.06.2024 12:24:19 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AppState](

[Id] [int] NOT NULL,

[DateAndTime] [smalldatetime] NOT NULL,

[Name] [nvarchar](255) NOT NULL,

CONSTRAINT [PK\_AppState] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[AppServiceState] WITH CHECK ADD CONSTRAINT [FK\_AppServiceState\_AppState] FOREIGN KEY([AppStateId])

REFERENCES [dbo].[AppState] ([Id])

GO

ALTER TABLE [dbo].[AppServiceState] CHECK CONSTRAINT [FK\_AppServiceState\_AppState]

GO

USE [master]

GO

ALTER DATABASE [Stuff] SET READ\_WRITE

GO

# ПРИЛОЖЕНИЕ Г

Программный код

Создание базы данных, код предоставлен в файле AppStatesDatabaseContext.cs:

using System.IO;

using Microsoft.Data.Sqlite;

using Microsoft.EntityFrameworkCore;

using SimulationStorm.AppStates.Persistence.EntityConfiguration;

namespace SimulationStorm.AppStates.Persistence;

public sealed class AppStatesDatabaseContext : DbContext

{

#region Tables

public DbSet<AppState> AppStates => Set<AppState>();

public DbSet<AppServiceState> AppServiceStates => Set<AppServiceState>();

#endregion

private readonly AppStatesOptions \_options;

public AppStatesDatabaseContext(AppStatesOptions options)

{

\_options = options;

}

public void F() => Database.CloseConnection();

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {Directory.CreateDirectory(\_options.DatabaseDirectoryPath;

var databaseFilePath = Path.Combine(\_options.DatabaseDirectoryPath, \_options.DatabaseFileName);

var connectionStringBuilder = new SqliteConnectionStringBuilder{

DataSource = databaseFilePath,

Pooling = false // This is needed to ensure a connection will be closed after disposing db context. };

var connectionString = connectionStringBuilder.ToString();

optionsBuilder .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking)

.UseLazyLoadingProxies()

.UseSqlite(connectionString);}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new AppStateEntityConfiguration(\_options));

modelBuilder.ApplyConfiguration(new AppServiceStateEntityConfiguration());)}

public class AppStateEntityConfiguration(AppStatesOptions options) : IEntityTypeConfiguration<AppState>

{

public void Configure(EntityTypeBuilder<AppState> builder)

{

builder.ToTable("AppStates");

builder.HasKey(x => x.Id);

builder.HasMany(x => x.ServiceStates);

builder

.Property(x => x.DateAndTime)

.IsRequired();

builder

.Property(x => x.Name)

.HasMaxLength(options.AppStateNameLengthRange.Maximum)

.IsRequired();}}

public class AppServiceStateEntityConfiguration : IEntityTypeConfiguration<AppServiceState>{

public void Configure(EntityTypeBuilder<AppServiceState> builder){

builder.ToTable("AppServiceStates");

builder.HasKey(x => x.Id);

builder

.HasOne(x => x.AppState)

.WithMany(x => x.ServiceStates)

.OnDelete(DeleteBehavior.Cascade);

builder

.Property(x => x.StateType)

.HasConversion(

type => type.AssemblyQualifiedName!,

typeName => Type.GetType(typeName) )

.IsRequired();

builder

.Property(x => x.State)

.HasConversion (

state => ObjectSerializer.SerializeObject(state),

serializedState => ObjectSerializer.DeserializeObject(serializedState)

.IsRequire}}

Менеджер симуляции «Игра Жизнь» уровня представления, код предоставлен в файле GameOfLifeManager.cs:

﻿using System.Collections.Generic;

using System.Threading.Tasks;

using SimulationStorm.GameOfLife.Algorithms;

using SimulationStorm.GameOfLife.DataTypes;

using SimulationStorm.GameOfLife.Presentation.Commands;

using SimulationStorm.Primitives;

using SimulationStorm.Simulation.Bounded.Presentation.Commands;

using SimulationStorm.Simulation.Bounded.Presentation.Services;

using SimulationStorm.Simulation.CellularAutomation;

using SimulationStorm.Simulation.CellularAutomation.Presentation.Commands;

using SimulationStorm.Simulation.CellularAutomation.Presentation.Services;

using SimulationStorm.Simulation.History.Presentation.Commands;

using SimulationStorm.Simulation.History.Presentation.Services;

using SimulationStorm.Simulation.Presentation.SimulationManager;

using SimulationStorm.Simulation.Resetting.Presentation.Commands;

using SimulationStorm.Simulation.Resetting.Presentation.Services;

using SimulationStorm.Simulation.Running.Presentation.Commands;

using SimulationStorm.Simulation.Running.Presentation.Services;

using SimulationStorm.Simulation.Statistics.Presentation.SummaryStats;

using SimulationStorm.Utilities.Benchmarking;

namespace SimulationStorm.GameOfLife.Presentation.Management;

public class GameOfLifeManager :

SimulationManagerBase,

ISaveableSimulationManager<GameOfLifeSave>,

ISummarizableSimulationManager<GameOfLifeSummary>,

IAdvanceableSimulationManager,

IBoundedSimulationManager,

IResettableSimulationManager,

IBoundedCellularAutomationManager<GameOfLifeCellState>

{

#region Properties

public Size WorldSize { get; private set; }

public WorldWrapping WorldWrapping { get; private set; }

public GameOfLifeAlgorithm Algorithm { get; private set; }

public GameOfLifeRule Rule { get; private set; } = null!;

#endregion

#region Fields

private readonly IGameOfLifeFactory \_gameOfLifeFactory;

private IGameOfLife \_gameOfLifeImpl;

#endregion

public GameOfLifeManager

(

IBenchmarkingService benchmarkingService,

IGameOfLifeFactory gameOfLifeFactory,

GameOfLifeManagerOptions options

)

: base(benchmarkingService, options)

{

\_gameOfLifeFactory = gameOfLifeFactory;

\_gameOfLifeImpl = \_gameOfLifeFactory.CreateGameOfLife(options.Algorithm,

options.WorldSize, options.WorldWrapping, options.Rule);

Algorithm = options.Algorithm;

RememberGameOfLifePropertyValues();

}

#region Reading methods

public Task<GameOfLifeSave> SaveAsync() =>

WithReadLockAsync(\_gameOfLifeImpl.Save);

public Task<BenchmarkResult<GameOfLifeSave>> SaveAndMeasureAsync() =>

MeasureWithReadLockAsync(\_gameOfLifeImpl.Save);

public Task<GameOfLifeSummary> SummarizeAsync() =>

WithReadLockAsync(\_gameOfLifeImpl.Summarize);

public Task<BenchmarkResult<GameOfLifeSummary>> SummarizeAndMeasureAsync() =>

MeasureWithReadLockAsync(\_gameOfLifeImpl.Summarize);

public Task<IEnumerable<Point>> GetAliveCellsAsync() =>

WithReadLockAsync(\_gameOfLifeImpl.GetAliveCells);

#endregion

#region Writing methods

public Task ChangeWorldSizeAsync(Size newSize) =>

QueueCommandExecutionAsync(new ChangeWorldSizeCommand(newSize));

public Task ChangeWorldWrappingAsync(WorldWrapping newWrapping) =>

QueueCommandExecutionAsync(new ChangeWorldWrappingCommand(newWrapping));

public Task ChangeRuleAsync(GameOfLifeRule newRule) =>

QueueCommandExecutionAsync(new ChangeRuleCommand(newRule));

public Task ChangeAlgorithmAsync(GameOfLifeAlgorithm newAlgorithm) =>

QueueCommandExecutionAsync(new ChangeAlgorithmCommand(newAlgorithm));

public Task RestoreStateAsync(GameOfLifeSave state, bool isRestoringFromAppState = false) =>

QueueCommandExecutionAsync(new RestoreStateCommand(state, isRestoringFromAppState));

public Task AdvanceAsync() =>

QueueCommandExecutionAsync(new AdvanceCommand());

public Task ResetAsync() =>

QueueCommandExecutionAsync(new ResetCommand());

public Task ChangeCellStatesAsync(IEnumerable<Point> cells, GameOfLifeCellState newState) =>

QueueCommandExecutionAsync(new DrawCommand<GameOfLifeCellState>(cells, newState));

public Task PlacePatternAtPositionsAsync(GameOfLifePattern pattern, IEnumerable<Point> positions, bool placeWithOverlay) =>

QueueCommandExecutionAsync(new PlacePatternCommand(pattern, positions, placeWithOverlay));

public Task PopulateRandomlyAsync(double cellBirthProbability) =>

QueueCommandExecutionAsync(new PopulateRandomlyCommand(cellBirthProbability));

#endregion

protected override void ExecuteCommand(SimulationCommand command)

{

switch (command)

{

case ChangeWorldSizeCommand changeWorldSizeCommand:

{

ExecuteChangeWorldSize(changeWorldSizeCommand);

break;

}

case ChangeAlgorithmCommand changeAlgorithmCommand:

{

ExecuteChangeAlgorithm(changeAlgorithmCommand);

break;

}

case ChangeRuleCommand changeRuleCommand:

{

ExecuteChangeRule(changeRuleCommand);

break;

}

case ChangeWorldWrappingCommand changeWorldWrappingCommand:

{

ExecuteChangeWorldWrapping(changeWorldWrappingCommand);

break;

}

case AdvanceCommand advanceCommand:

{

ExecuteAdvance(advanceCommand);

break;

}

case ResetCommand resetCommand:

{

ExecuteReset(resetCommand);

break;

}

case RestoreStateCommand restoreStateCommand:

{

ExecuteRestoreState(restoreStateCommand);

break;

}

case DrawCommand<GameOfLifeCellState> drawCommand:

{

ExecuteDraw(drawCommand);

break;

}

case PlacePatternCommand placePatternCommand:

{

ExecutePlacePattern(placePatternCommand);

break;

}

case PopulateRandomlyCommand populateRandomlyCommand:

{

ExecutePopulateRandomly(populateRandomlyCommand);

break;

}

}

}

#region Commands execution

private void ExecuteChangeWorldSize(ChangeWorldSizeCommand command)

{

\_gameOfLifeImpl.ChangeWorldSize(command.NewSize);

WorldSize = \_gameOfLifeImpl.WorldSize;

}

private void ExecuteChangeAlgorithm(ChangeAlgorithmCommand command)

{

var newAlgorithm = command.NewAlgorithm;

var newImplementation = \_gameOfLifeFactory.CreateGameOfLife(newAlgorithm,

\_gameOfLifeImpl.WorldSize, \_gameOfLifeImpl.WorldWrapping, \_gameOfLifeImpl.Rule);

foreach (var cell in \_gameOfLifeImpl.GetAliveCells())

newImplementation.SetCellState(cell, GameOfLifeCellState.Alive);

\_gameOfLifeImpl = newImplementation;

Algorithm = newAlgorithm;

}

private void ExecuteChangeRule(ChangeRuleCommand command)

{

\_gameOfLifeImpl.Rule = command.NewRule;

Rule = \_gameOfLifeImpl.Rule;

}

private void ExecuteChangeWorldWrapping(ChangeWorldWrappingCommand command)

{

\_gameOfLifeImpl.WorldWrapping = command.NewWorldWrapping;

WorldWrapping = \_gameOfLifeImpl.WorldWrapping;

}

private void ExecuteAdvance(AdvanceCommand command) =>

\_gameOfLifeImpl.Advance();

private void ExecuteReset(ResetCommand command) =>

\_gameOfLifeImpl.Reset();

private void ExecuteRestoreState(RestoreStateCommand command)

{

var state = (GameOfLifeSave)command.State;

switch (state.Algorithm)

{

case GameOfLifeAlgorithm.Bitwise when Algorithm is not GameOfLifeAlgorithm.Bitwise:

{

ExecuteChangeAlgorithm(new ChangeAlgorithmCommand(GameOfLifeAlgorithm.Bitwise));

break;

}

case GameOfLifeAlgorithm.Smart when Algorithm is not GameOfLifeAlgorithm.Smart:

{

ExecuteChangeAlgorithm(new ChangeAlgorithmCommand(GameOfLifeAlgorithm.Smart));

break;

}

}

\_gameOfLifeImpl.RestoreState(state);

RememberGameOfLifePropertyValues();

}

private void ExecuteDraw(DrawCommand<GameOfLifeCellState> command)

{

var cells = command.Cells;

var newState = command.NewState;

foreach (var cell in cells)

\_gameOfLifeImpl.SetCellState(cell, newState);

}

private void ExecutePlacePattern(PlacePatternCommand command)

{

var pattern = command.Pattern;

var positions = command.Positions;

foreach (var position in positions)

\_gameOfLifeImpl.PlacePattern(pattern, position, command.PlaceWithOverlay);

}

private void ExecutePopulateRandomly(PopulateRandomlyCommand command) =>

\_gameOfLifeImpl.PopulateRandomly(command.CellBirthProbability);

#endregion

private void RememberGameOfLifePropertyValues()

{

WorldSize = \_gameOfLifeImpl.WorldSize;

WorldWrapping = \_gameOfLifeImpl.WorldWrapping;

Rule = \_gameOfLifeImpl.Rule;

}

}

Отрисовщик симуляции «Игра Жизнь», код предоставлен в файле GameOfLifeRenderer.cs:

﻿using System.Linq;

using System.Threading.Tasks;

using SimulationStorm.Graphics;

using SimulationStorm.GameOfLife.Presentation.Management;

using SimulationStorm.Primitives;

using SimulationStorm.Simulation.Cellular.Presentation.Services;

using SimulationStorm.Utilities;

using SimulationStorm.Utilities.Benchmarking;

namespace SimulationStorm.GameOfLife.Presentation.Rendering;

public class GameOfLifeRenderer : BoundedCellularSimulationRendererBase

{

public (Color DeadCellColor, Color AliveCellColor) CellColors

{

get => (\_deadCellColor, \_aliveCellPaint.Color);

set

{

var (deadCellColor, aliveCellColor) = value;

if (deadCellColor == \_deadCellColor && aliveCellColor == \_aliveCellPaint.Color)

return;

\_deadCellColor = deadCellColor;

\_aliveCellPaint.Color = aliveCellColor;

OnPropertyChanged();

if (IsRenderingEnabled)

RequestRerender();

}

}

#region Fields

private readonly GameOfLifeManager \_gameOfLifeManager;

private Color \_deadCellColor;

private readonly IPaint \_aliveCellPaint;

#endregion

public GameOfLifeRenderer

(

IGraphicsFactory graphicsFactory,

IBenchmarkingService benchmarkingService,

IIntervalActionExecutor intervalActionExecutor,

GameOfLifeManager gameOfLifeManager,

GameOfLifeRendererOptions options

)

: base(graphicsFactory, benchmarkingService, intervalActionExecutor, gameOfLifeManager, options)

{

\_gameOfLifeManager = gameOfLifeManager;

\_deadCellColor = options.DeadCellColor;

\_aliveCellPaint = GraphicsFactory.CreatePaint();

\_aliveCellPaint.Color = options.AliveCellColor;

WithDisposables(disposables => disposables.Add(\_aliveCellPaint));

}

protected override async Task RenderAsync(ICanvas canvas)

{

canvas.Clear(\_deadCellColor);

var aliveCells = await \_gameOfLifeManager.GetAliveCellsAsync();

var aliveCellRectangles = aliveCells.Select(cell => new RectF(cell.X, cell.Y, 1, 1));

canvas.DrawRects(aliveCellRectangles, \_aliveCellPaint);

}

}

Основной алгоритм симуляции «Игра жизнь», код предоставлен в файле BitwiseGameOfLife.cs:

﻿using System;

using System.Collections.Generic;

using SimulationStorm.GameOfLife.DataTypes;

using SimulationStorm.Primitives;

using SimulationStorm.Simulation.CellularAutomation;

namespace SimulationStorm.GameOfLife.Algorithms;

public class BitwiseGameOfLife : GameOfLifeBase

{

public const int BatchSize = 8;

#region Properties

public override Size WorldSize => new(\_worldWidth - 2, \_worldHeight - 2);

public sealed override GameOfLifeRule Rule

{

get => \_rule;

set => SetRule(value);

}

#endregion

#region Fields

private int \_worldWidth, \_worldHeight;

private byte[] \_world = null!, // Cells current live states

\_neighbors = null!; // Neighbors of each cell (on respective positions)

private bool \_areSidesReset = true;

private readonly IDictionary<GameOfLifeRule, IReadOnlyList<byte>> \_cellStatesByRuleLookupTable =

new Dictionary<GameOfLifeRule, IReadOnlyList<byte>>();

private GameOfLifeRule \_rule = null!;

private IReadOnlyList<byte> \_cellStatesLookupTable = null!;

#endregion

public BitwiseGameOfLife(Size size, WorldWrapping wrapping, GameOfLifeRule rule) : base(wrapping)

{

ChangeWorldSize(size);

Rule = rule;

}

#region Public methods

public override bool IsValidWorldSize(Size size) => size.Area % BatchSize is 0;

public override GameOfLifeSave Save() =>

new(WorldSize, WorldWrapping, Rule, GameOfLifeAlgorithm.Bitwise, world: (byte[])\_world.Clone());

public override void RestoreState(GameOfLifeSave state)

{

if (state.Algorithm is not GameOfLifeAlgorithm.Bitwise || state.World is null)

throw new ArgumentException($"Save algorithm must be {nameof(GameOfLifeAlgorithm.Bitwise)} " +

$"and {nameof(state.World)} must not be null.", nameof(state));

\_worldWidth = state.WorldSize.Width + 2;

\_worldHeight = state.WorldSize.Height + 2;

\_world = (byte[])state.World.Clone();

\_neighbors = new byte[\_worldWidth \* \_worldHeight];

WorldWrapping = state.WorldWrapping;

Rule = state.Rule;

}

public override GameOfLifeSummary Summarize()

{

var aliveCellCount = 0;

for (var i = \_worldWidth + 1; i < (\_worldHeight - 1) \* \_worldWidth; i++)

aliveCellCount += \_world[i];

int totalCellCount = WorldSize.Area,

deadCellCount = totalCellCount - aliveCellCount;

return new GameOfLifeSummary(deadCellCount, aliveCellCount);

}

public sealed override void ChangeWorldSize(Size newSize)

{

if (newSize.Area is 0 || newSize.Area % BatchSize is not 0)

throw new ArgumentOutOfRangeException(nameof(newSize), newSize,

$"The area of the size must be greater than zero and dividable by {BatchSize}.");

\_worldWidth = newSize.Width + 2;

\_worldHeight = newSize.Height + 2;

\_world = new byte[\_worldWidth \* \_worldHeight];

\_neighbors = new byte[\_worldWidth \* \_worldHeight];

}

public override void Advance()

{

ApplyWorldWrapping();

ClearCellNeighborCounts();

CountCellNeighbors();

ApplyRule();

}

public override void Reset()

{

\_world = new byte[\_worldWidth \* \_worldHeight];

\_neighbors = new byte[\_worldWidth \* \_worldHeight];

}

public override GameOfLifeCellState GetCellState(Point cell) =>

(GameOfLifeCellState)\_world[(cell.Y + 1) \* \_worldWidth + cell.X + 1];

public override void SetCellState(Point cell, GameOfLifeCellState newState) =>

\_world[(cell.Y + 1) \* \_worldWidth + cell.X + 1] = (byte)newState;

#endregion

#region Private methods

private void SetRule(GameOfLifeRule rule)

{

\_rule = rule ?? throw new ArgumentNullException(nameof(rule));

\_cellStatesLookupTable = GetCellStatesLookupTableByRule(rule);

}

#region World wrapping methods

private void ApplyWorldWrapping()

{

switch (WorldWrapping)

{

case WorldWrapping.NoWrap:

if (!\_areSidesReset)

{

ResetSides();

\_areSidesReset = true;

}

break;

case WorldWrapping.Horizontal:

CopyHorizontalSides();

\_areSidesReset = false;

break;

case WorldWrapping.Vertical:

CopyVerticalSides();

\_areSidesReset = false;

break;

case WorldWrapping.Both:

CopyHorizontalSides();

CopyVerticalSides();

\_areSidesReset = false;

break;

}

}

private void CopyHorizontalSides()

{

for (var y = 0; y < \_worldHeight; y++)

{

// Copy right to left

\_world[y \* \_worldWidth] = \_world[y \* \_worldWidth + (\_worldWidth - 2)];

// Copy left to right

\_world[y \* \_worldWidth + (\_worldWidth - 1)] = \_world[y \* \_worldWidth + 1];

}

}

private void CopyVerticalSides()

{

for (var x = 0; x < \_worldWidth; x++)

{

// Copy bottom to top

\_world[x] = \_world[(\_worldHeight - 2) \* \_worldWidth + x];

// Copy top to bottom

\_world[(\_worldHeight - 1) \* \_worldWidth + x] = \_world[\_worldWidth + x];

}

}

private void ResetSides()

{

for (var y = 0; y < \_worldHeight; y++)

{

// Reset left

\_world[y \* \_worldWidth] = 0;

// Reset right

\_world[y \* \_worldWidth + (\_worldWidth - 1)] = 0;

}

for (var x = 0; x < \_worldWidth; x++)

{

// Reset top

\_world[x] = 0;

// Reset bottom

\_world[(\_worldHeight - 1) \* \_worldWidth + x] = 0;

}

}

#endregion

private unsafe void ClearCellNeighborCounts()

{

fixed (byte\* neighborsPtr = \_neighbors)

{

for (var i = 0; i < \_worldWidth \* \_worldHeight; i += BatchSize)

\*(ulong\*)(neighborsPtr + i) = 0;

}

}

private unsafe void CountCellNeighbors()

{

fixed (byte\* worldPtr = \_world, neighborsPtr = \_neighbors)

{

for (var i = \_worldWidth + 1; i < (\_worldHeight - 1) \* \_worldWidth - 1; i += BatchSize)

{

var ptr = (ulong\*)(neighborsPtr + i);

\*ptr += \*(ulong\*)(worldPtr + i - \_worldWidth - 1);

\*ptr += \*(ulong\*)(worldPtr + i - \_worldWidth);

\*ptr += \*(ulong\*)(worldPtr + i - \_worldWidth + 1);

\*ptr += \*(ulong\*)(worldPtr + i - 1);

\*ptr += \*(ulong\*)(worldPtr + i + 1);

\*ptr += \*(ulong\*)(worldPtr + i + \_worldWidth - 1);

\*ptr += \*(ulong\*)(worldPtr + i + \_worldWidth);

\*ptr += \*(ulong\*)(worldPtr + i + \_worldWidth + 1);

}

}

}

private void ApplyRule()

{

var worldWidthMinusOne = \_worldWidth - 1;

for (var i = \_worldWidth + 1; i < (\_worldHeight - 1) \* \_worldWidth - 1; i++)

{

if (i % \_worldWidth == worldWidthMinusOne)

{

i++;

continue;

}

byte cellState = \_world[i],

cellNeighborCount = \_neighbors[i],

newCellState = \_cellStatesLookupTable[cellState << 4 | cellNeighborCount];

\_world[i] = newCellState;

}

}

private IReadOnlyList<byte> GetCellStatesLookupTableByRule(GameOfLifeRule rule)

{

if (\_cellStatesByRuleLookupTable.TryGetValue(rule, out var cellStatesLookupTable))

return cellStatesLookupTable;

cellStatesLookupTable = CreateCellStatesLookupTable(rule);

\_cellStatesByRuleLookupTable[rule] = cellStatesLookupTable;

return cellStatesLookupTable;

}

private static IReadOnlyList<byte> CreateCellStatesLookupTable(GameOfLifeRule rule)

{

var cellStatesLookupTable = new byte[BatchSize \* 4];

for (var i = GameOfLifeRule.MinNeighborCount; i <= GameOfLifeRule.MaxNeighborCount; i++)

{

if (rule.IsBornWhen(i))

cellStatesLookupTable[i] = 1;

if (rule.IsSurviveWhen(i))

cellStatesLookupTable[BatchSize \* 2 + i] = 1;

}

return cellStatesLookupTable;

}

#endregion

}

Алгоритм Smart симуляции «Игра Жизнь», код предоставлен в файле SmartGameOfLife.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using SimulationStorm.GameOfLife.DataTypes;

using SimulationStorm.Primitives;

using SimulationStorm.Simulation.CellularAutomation;

namespace SimulationStorm.GameOfLife.Algorithms;

public class SmartGameOfLife : GameOfLifeBase

{

#region Properties

public override Size WorldSize => new(\_worldWidth, \_worldHeight);

public sealed override GameOfLifeRule Rule

{

get => \_rule;

set => SetRule(value);

}

#endregion

#region Fields

private int \_worldWidth,

\_worldHeight;

private ISet<Point> \_aliveCells = new HashSet<Point>();

private readonly IDictionary<Point, int> \_neighborCountsByCell = new Dictionary<Point, int>();

private GameOfLifeRule \_rule = null!;

#endregion

public SmartGameOfLife(Size size, WorldWrapping wrapping, GameOfLifeRule rule) : base(wrapping)

{

ChangeWorldSize(size);

Rule = rule;

}

#region Public methods

public override bool IsValidWorldSize(Size size) => size.Area >= 1;

public override GameOfLifeSave Save() =>

new(WorldSize, WorldWrapping, Rule, GameOfLifeAlgorithm.Smart, aliveCells: \_aliveCells.ToHashSet());

public override void RestoreState(GameOfLifeSave state)

{

if (state.Algorithm is not GameOfLifeAlgorithm.Smart || state.AliveCells is null)

throw new ArgumentException($"Saving algorithm must be {nameof(GameOfLifeAlgorithm.Smart)} " +

$"and {nameof(state.AliveCells)} must not be null.", nameof(state));

\_worldWidth = state.WorldSize.Width;

\_worldHeight = state.WorldSize.Height;

\_aliveCells = state.AliveCells.ToHashSet();

WorldWrapping = state.WorldWrapping;

Rule = state.Rule;

}

public override GameOfLifeSummary Summarize()

{

int totalCellCount = \_worldWidth \* \_worldHeight,

aliveCellCount = \_aliveCells.Count,

deadCellCount = totalCellCount - aliveCellCount;

return new GameOfLifeSummary(deadCellCount, aliveCellCount);

}

public sealed override void ChangeWorldSize(Size newSize)

{

ArgumentOutOfRangeException.ThrowIfLessThan(newSize.Width, 1, nameof(newSize.Width));

ArgumentOutOfRangeException.ThrowIfLessThan(newSize.Height, 1, nameof(newSize.Height));

\_worldWidth = newSize.Width;

\_worldHeight = newSize.Height;

\_aliveCells.Clear();

}

public override void Advance()

{

CountCellsNeighbors();

ApplyRule();

}

public override void Reset() => \_aliveCells.Clear();

public override GameOfLifeCellState GetCellState(Point cell) =>

\_aliveCells.Contains(cell) ? GameOfLifeCellState.Alive : GameOfLifeCellState.Dead;

public override void SetCellState(Point cell, GameOfLifeCellState newState)

{

if (newState is GameOfLifeCellState.Alive)

\_aliveCells.Add(cell);

else

\_aliveCells.Remove(cell);

}

#endregion

#region Private methods

private void SetRule(GameOfLifeRule rule) => \_rule = rule ?? throw new ArgumentNullException(nameof(rule));

private void CountCellsNeighbors()

{

\_neighborCountsByCell.Clear();

foreach (var cell in \_aliveCells)

CountCellNeighbors(cell);

}

private void ApplyRule()

{

var nextWorld = new HashSet<Point>();

foreach (var (cell, neighborCount) in \_neighborCountsByCell)

{

var wasAliveBefore = \_aliveCells.Contains(cell);

if (\_rule.IsBornWhen(neighborCount) || \_rule.IsSurviveWhen(neighborCount) && wasAliveBefore)

nextWorld.Add(cell);

}

\_aliveCells = nextWorld;

}

private void CountCellNeighbors(Point cell)

{

int x = cell.X,

y = cell.Y;

for (var xOffset = -1; xOffset <= 1; xOffset++)

{

for (var yOffset = -1; yOffset <= 1; yOffset++)

{

if (xOffset == 0 && yOffset == 0)

continue;

int neighborX = x + xOffset,

neighborY = y + yOffset;

if (!ApplyWorldWrappingToCell(ref neighborX, ref neighborY))

continue;

var neighbor = new Point(neighborX, neighborY);

if (!\_neighborCountsByCell.TryAdd(neighbor, 1))

\_neighborCountsByCell[neighbor]++;

}

}

}

private bool ApplyWorldWrappingToCell(ref int x, ref int y)

{

switch (WorldWrapping)

{

case WorldWrapping.NoWrap:

if (x < 0 || x > \_worldWidth - 1 || y < 0 || y > \_worldHeight - 1)

return false;

break;

case WorldWrapping.Horizontal:

x = (x + \_worldWidth) % \_worldWidth;

break;

case WorldWrapping.Vertical:

y = (y + \_worldHeight) % \_worldHeight;

break;

case WorldWrapping.Both:

x = (x + \_worldWidth) % \_worldWidth;

y = (y + \_worldHeight) % \_worldHeight;

break;

}

return true;

}

#endregion

}

Регулировки сервисов уровня представления, код предоставлен в файле PresentationDependencyInjection.cs:

﻿using System;

using System.Collections.Generic;

using System.IO;

using System.Numerics;

using SimulationStorm.AppStates.Persistence;

using SimulationStorm.Collections.StorageControl;

using SimulationStorm.Graphics;

using SimulationStorm.GameOfLife.Algorithms;

using SimulationStorm.GameOfLife.DataTypes;

using SimulationStorm.GameOfLife.Presentation.Drawing;

using SimulationStorm.GameOfLife.Presentation.Management;

using SimulationStorm.GameOfLife.Presentation.Population;

using SimulationStorm.GameOfLife.Presentation.Rendering;

using SimulationStorm.GameOfLife.Presentation.Rules;

using SimulationStorm.GameOfLife.Presentation.Stats;

using SimulationStorm.GameOfLife.Presentation.ViewModels;

using SimulationStorm.Logging;

using SimulationStorm.Primitives;

using SimulationStorm.Simulation.Cellular.Presentation.Services;

using SimulationStorm.Simulation.CellularAutomation;

using SimulationStorm.Simulation.CellularAutomation.Presentation.Models;

using SimulationStorm.Simulation.History.Presentation.Services;

using SimulationStorm.Simulation.History.Presentation.ViewModels;

using SimulationStorm.Simulation.Presentation.Camera;

using SimulationStorm.Simulation.Presentation.StatusBar;

using SimulationStorm.Simulation.Running.Presentation.Services;

using SimulationStorm.Simulation.Statistics.Presentation.Charts;

using SimulationStorm.Simulation.Statistics.Presentation.CommandExecutionStats;

using SimulationStorm.Simulation.Statistics.Presentation.CommandExecutionStats.Charts;

using SimulationStorm.Simulation.Statistics.Presentation.RenderingStats;

using SimulationStorm.Simulation.Statistics.Presentation.RenderingStats.Charts;

using SimulationStorm.Simulation.Statistics.Presentation.SummaryStats;

using SimulationStorm.Simulation.Statistics.Presentation.SummaryStats.ViewModels;

using SimulationStorm.ToolPanels.Presentation;

namespace SimulationStorm.GameOfLife.Presentation.Startup;

public static class PresentationConfiguration

{

public const string ApplicationName = "SimulationStorm | Game of Life";

// Directory name is different because vertical separator symbol is not allowed in a directory name.

public const string ApplicationDirectoryName = "SimulationStorm - Game of Life";

public static readonly string ApplicationWorkingFilesDirectoryPath = Path.Combine

(

Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),

ApplicationDirectoryName

);

public static readonly LoggingOptions LoggingOptions = new()

{

LogTemplate = "[{Level:u3}] {Timestamp:HH:mm:ss} " +

"| Thread: {ThreadId} | {SourceContext} | {Message:lj}{NewLine}{Exception}",

LogFilesDirectoryPath = Path.Combine(ApplicationWorkingFilesDirectoryPath, "Logs"),

LogsFileName = $"Logs on {DateTime.Now:dd-MM-yy HH-mm-ss}.txt"

};

public static readonly AppStatesOptions AppStatesOptions = new()

{

DatabaseDirectoryPath = ApplicationWorkingFilesDirectoryPath,

DatabaseFileName = "AppStatesDatabase.db",

AppStateNameLengthRange = new Range<int>(1, 50)

};

public static readonly WorldCameraOptions WorldCameraOptions = new()

{

ZoomRange = new Range<float>(0.1f, 100f),

ZoomChangeRange = new Range<float>(0.01f, 10f),

DefaultZoomChange = 0.1f,

DefaultZoom = 10f,

TranslationRange = new Range<PointF>(new PointF(-100\_000, -100\_000), new PointF(100\_000, 100\_000)),

TranslationChangeRange = new Range<float>(1, 1\_000),

DefaultTranslationChange = 10f,

DefaultTranslation = Vector2.Zero

};

public static readonly GameOfLifeDrawingOptions DrawingOptions = new()

{

IsDrawingEnabled = false,

BrushRadiusRange = new Range<int>(0, 50),

BrushRadius = 0,

BrushShape = DrawingShape.Circle,

BrushCellState = GameOfLifeCellState.Alive,

Pattern = null

};

public static readonly SimulationRunnerOptions SimulationRunnerOptions = new()

{

MaxStepsPerSecondRange = new Range<int>(1, 100),

MaxStepsPerSecond = 1

};

public static readonly PopulationOptions PopulationOptions = new()

{

CellBirthProbabilityRange = new Range<double>(0.01, 1),

CellBirthProbability = 0.01

};

public static readonly GameOfLifeRendererOptions GameOfLifeRendererOptions = new()

{

IsRenderingEnabled = true,

RenderingIntervalRange = new Range<int>(0, 1\_000),

RenderingInterval = 0,

DeadCellColor = KnownColors.White,

AliveCellColor = KnownColors.Black

};

public static readonly CellularWorldRendererOptions CellularWorldRendererOptions = new()

{

IsGridLinesVisible = true,

GridLinesColor = new Color(0, 0, 0, 128),

HoveredCellColor = new Color(134, 179, 238), // Actipro's ControlBackgroundBrushEmphasizedAccentPointerOver

PressedCellColor = new Color(88, 147, 166) // Actipro's ControlBackgroundBrushEmphasizedAccentPointerPressed

};

public static readonly GameOfLifeManagerOptions GameOfLifeManagerOptions = new()

{

// The simulation manager command executed event could be handled by:

// 1) ISimulationRenderer, 2) IHistoryManager, 3) IStatisticsManager

CommandExecutedEventHandlerCount = 3,

WorldSizeRange = new Range<Size>(new Size(1, 1), new Size(1\_920, 1\_080) \* 4),

// WorldSize = new Size(VectorLifeAutomation.BatchSize, VectorLifeAutomation.BatchSize),

WorldSize = new Size(1\_920, 1\_080) / 10,

Algorithm = GameOfLifeAlgorithm.Bitwise,

// Algorithm = LifeAutomationAlgorithm.Vector,

WorldWrapping = WorldWrapping.NoWrap,

Rule = PredefinedRules.GameOfLife.Rule

};

public static readonly HistoryOptions HistoryOptions = new()

{

StorageLocation = CollectionStorageLocation.Memory,

CapacityRange = new Range<int>(1, 1\_000),

Capacity = 50,

IsSavingEnabled = false,

SavingIntervalRange = new Range<int>(0, 1\_000),

SavingInterval = 0

};

public static readonly StatusBarOptions StatusBarOptions = new()

{

IsCommandProgressVisible = true,

IsCommandTimeVisible = true,

IsSimulationRenderingProgressVisible = true,

IsSimulationRenderingTimeVisible = true,

IsWorldRenderingTimeVisible = true

};

public static readonly CommandExecutionStatsOptions CommandExecutionStatsOptions = new()

{

IsSavingEnabled = false,

Capacity = 50,

CapacityRange = new Range<int>(1, 1\_000),

ChartTypes = new[]

{

ChartType.Line,

ChartType.Bar,

},

DefaultChartType = ChartType.Line,

ViewModelTypesByChartType = new Dictionary<ChartType, Type>

{

[ChartType.Line] = typeof(CommandExecutionLineChartViewModel),

[ChartType.Bar] = typeof(CommandExecutionBarChartViewModel),

},

CommandNumberAxisNameKey = "Simulation.Stats.CommandNumber",

ExecutionTimeAxisNameKey = "Simulation.Stats.ExecutionTime",

ExecutionTimeAxisTimeUnit = TimeSpan.FromMicroseconds(1)

};

public static readonly RenderingStatsOptions RenderingStatsOptions = new()

{

IsSavingEnabled = false,

Capacity = 50,

CapacityRange = new Range<int>(1, 1\_000),

ChartTypes = new[]

{

ChartType.Line,

ChartType.Bar,

},

DefaultChartType = ChartType.Line,

ViewModelTypesByChartType = new Dictionary<ChartType, Type>

{

[ChartType.Line] = typeof(RenderingLineChartViewModel),

[ChartType.Bar] = typeof(RenderingBarChartViewModel),

},

CommandNumberAxisNameKey = "Simulation.Stats.CommandNumber",

RenderingTimeAxisNameKey = "Simulation.Stats.RenderingTime",

RenderingTimeAxisTimeUnit = TimeSpan.FromMicroseconds(1)

};

public static readonly SummaryStatsOptions SummaryStatsOptions = new()

{

StorageLocation = CollectionStorageLocation.Memory,

CapacityRange = new Range<int>(1, 1\_000),

Capacity = 50,

IsSavingEnabled = false,

SavingIntervalRange = new Range<int>(0, 1\_000),

SavingInterval = 0,

ChartTypes = new[]

{

ChartType.Table,

ChartType.Pie,

ChartType.Bar,

ChartType.Line

},

DefaultChartType = ChartType.Table,

ViewModelTypesByChartType = new Dictionary<ChartType, Type>

{

[ChartType.Table] = typeof(TableChartViewModel),

[ChartType.Pie] = typeof(PieChartViewModel),

[ChartType.Bar] = typeof(BarChartViewModel),

[ChartType.Line] = typeof(LineChartViewModel)

},

CommandNumberAxisNameKey = "Simulation.Stats.CommandNumber",

AliveCellCountAxisNameKey = "GameOfLife.AliveCellCount",

DeadCellsStringKey = "GameOfLife.DeadCells",

AliveCellsStringKey = "GameOfLife.AliveCells"

};

public static readonly ToolPanelOptions ToolPanelOptions = new()

{

ToolPanelPositions = new Dictionary<ToolPanel, ToolPanelPosition>

{

[GameOfLifeToolPanels.SimulationToolPanel] = ToolPanelPosition.TopRight,

[GameOfLifeToolPanels.DrawingToolPanel] = ToolPanelPosition.TopRight,

[GameOfLifeToolPanels.RenderingToolPanel] = ToolPanelPosition.TopLeft,

[GameOfLifeToolPanels.CameraToolPanel] = ToolPanelPosition.TopLeft,

[GameOfLifeToolPanels.HistoryToolPanel] = ToolPanelPosition.BottomRight,

[GameOfLifeToolPanels.StatisticsToolPanel] = ToolPanelPosition.BottomLeft

},

ToolPanelVisibilities = new Dictionary<ToolPanel, bool>

{

[GameOfLifeToolPanels.SimulationToolPanel] = false,

[GameOfLifeToolPanels.DrawingToolPanel] = false,

[GameOfLifeToolPanels.RenderingToolPanel] = false,

[GameOfLifeToolPanels.CameraToolPanel] = false,

[GameOfLifeToolPanels.StatisticsToolPanel] = false,

[GameOfLifeToolPanels.HistoryToolPanel] = false

},

ToolPanelViewModelTypes = new Dictionary<ToolPanel, Type>

{

[GameOfLifeToolPanels.SimulationToolPanel] = typeof(SimulationToolPanelViewModel),

[GameOfLifeToolPanels.DrawingToolPanel] = typeof(DrawingToolPanelViewModel),

[GameOfLifeToolPanels.RenderingToolPanel] = typeof(RenderingToolPanelViewModel),

[GameOfLifeToolPanels.CameraToolPanel] = typeof(CameraSettingsViewModel),

[GameOfLifeToolPanels.StatisticsToolPanel] = typeof(ISummaryStatsViewModel),

[GameOfLifeToolPanels.HistoryToolPanel] = typeof(IHistoryViewModel)

}

};

}

Настройка сервисов уровня представления, код предоставлен в файле PresentationConfiguration.cs:

﻿using Microsoft.Extensions.DependencyInjection;

using SimulationStorm.AppStates.Presentation;

using SimulationStorm.Collections.Lists;

using SimulationStorm.Collections.Universal;

using SimulationStorm.DependencyInjection;

using SimulationStorm.Exceptions;

using SimulationStorm.Exceptions.Logging;

using SimulationStorm.Graphics;

using SimulationStorm.Graphics.Skia;

using SimulationStorm.GameOfLife.Presentation.Drawing;

using SimulationStorm.GameOfLife.Presentation.Management;

using SimulationStorm.GameOfLife.Presentation.Patterns;

using SimulationStorm.GameOfLife.Presentation.Population;

using SimulationStorm.GameOfLife.Presentation.Rendering;

using SimulationStorm.GameOfLife.Presentation.Rules;

using SimulationStorm.GameOfLife.Presentation.ViewModels;

using SimulationStorm.Logging;

using SimulationStorm.Simulation.Bounded.Presentation.Services;

using SimulationStorm.Simulation.Bounded.Presentation.ViewModels;

using SimulationStorm.Simulation.Cellular.Presentation.Services;

using SimulationStorm.Simulation.Cellular.Presentation.ViewModels;

using SimulationStorm.Simulation.CellularAutomation.Presentation.Services;

using SimulationStorm.Simulation.CellularAutomation.Presentation.ViewModels;

using SimulationStorm.Simulation.History.Presentation.Services;

using SimulationStorm.Simulation.History.Presentation.ViewModels;

using SimulationStorm.Simulation.Presentation.Camera;

using SimulationStorm.Simulation.Presentation.SimulationManager;

using SimulationStorm.Simulation.Presentation.SimulationRenderer;

using SimulationStorm.Simulation.Presentation.Viewport;

using SimulationStorm.Simulation.Presentation.WorldRenderer;

using SimulationStorm.Simulation.Resetting.Presentation.Services;

using SimulationStorm.Simulation.Running.Presentation.Services;

using SimulationStorm.Simulation.Running.Presentation.ViewModels;

using SimulationStorm.Simulation.Statistics.Presentation;

using SimulationStorm.Simulation.Statistics.Presentation.Charts;

using SimulationStorm.ToolPanels.Presentation;

using SimulationStorm.Utilities;

using SimulationStorm.Utilities.Benchmarking;

using System;

using LiveChartsCore.Kernel.Sketches;

using SimulationStorm.GameOfLife.DataTypes;

using SimulationStorm.GameOfLife.Presentation.Stats;

using SimulationStorm.Presentation.StartupOperations;

using SimulationStorm.Simulation.Presentation.StatusBar;

using SimulationStorm.Simulation.Statistics.Presentation.CommandExecutionStats;

using SimulationStorm.Simulation.Statistics.Presentation.CommandExecutionStats.Charts;

using SimulationStorm.Simulation.Statistics.Presentation.CommandExecutionStats.ViewModels;

using SimulationStorm.Simulation.Statistics.Presentation.RenderingStats;

using SimulationStorm.Simulation.Statistics.Presentation.RenderingStats.Charts;

using SimulationStorm.Simulation.Statistics.Presentation.RenderingStats.ViewModels;

using SimulationStorm.Simulation.Statistics.Presentation.SummaryStats;

using SimulationStorm.Simulation.Statistics.Presentation.SummaryStats.Charts;

using SimulationStorm.Simulation.Statistics.Presentation.SummaryStats.ViewModels;

namespace SimulationStorm.GameOfLife.Presentation.Startup;

public static class PresentationDependencyInjection

{

public static IServiceCollection AddExceptionHandlingAndLoggingServices(this IServiceCollection services) => services

.AddLoggingServices(PresentationConfiguration.LoggingOptions)

.AddAppDomainExceptionNotifier(AppDomain.CurrentDomain)

.AddUnhandledExceptionLogger()

.AddSingleton<ExceptionHandlerToNotifiersSubscriber>();

public static IServiceCollection AddPresentationServices(this IServiceCollection services) => services

// Independent transient services

.AddTransient<IBenchmarkingService, BenchmarkingService>()

.AddTransient<IIntervalActionExecutor, IntervalActionExecutor>()

//

// Independent singleton services

.AddSingleton<IGraphicsFactory, SkiaGraphicsFactory>()

.AddSingleton<IListFactory, ListFactory>()

.AddSingleton<IUniversalCollectionFactory, UniversalCollectionFactory>()

.AddSingleton<IGameOfLifeFactory, GameOfLifeFactory>()

//

// Simulation manager

.AddSingleton(PresentationConfiguration.GameOfLifeManagerOptions)

.AddSharedSingleton<IBoundedSimulationManagerOptions, GameOfLifeManagerOptions>()

.AddSingleton<GameOfLifeManager>()

.AddSharedSingleton<ISimulationManager, GameOfLifeManager>()

.AddSharedSingleton<IAdvanceableSimulationManager, GameOfLifeManager>()

.AddSharedSingleton<IResettableSimulationManager, GameOfLifeManager>()

.AddSharedSingleton<IBoundedSimulationManager, GameOfLifeManager>()

.AddSharedSingleton<ISaveableSimulationManager<GameOfLifeSave>, GameOfLifeManager>()

.AddSharedSingleton<ISummarizableSimulationManager<GameOfLifeSummary>, GameOfLifeManager>()

.AddSharedSingleton<ICellularAutomationManager<GameOfLifeCellState>, GameOfLifeManager>()

.AddSharedSingleton<IBoundedCellularAutomationManager<GameOfLifeCellState>, GameOfLifeManager>()

.AddAsyncServiceStateManager<GameOfLifeStateManager>()

//

// Simulation runner

.AddSingleton(PresentationConfiguration.SimulationRunnerOptions)

.AddSingleton<ISimulationRunner, SimulationRunner>()

.AddSingleton<SimulationRunnerViewModel>()

.AddServiceStateManager<SimulationRunnerStateManager>()

.AddAppStateRestoringOperation<PauseSimulationOnAppStateRestoringOperation>()

//

// Tool panels infrastructure

.AddSingleton(PresentationConfiguration.ToolPanelOptions)

.AddSingleton<IToolPanelManager, ToolPanelManager>()

.AddSingleton<IToolPanelViewModelFactory, ToolPanelViewModelFactory>()

.AddSingleton<ToolPanelManagerViewModel>()

.AddServiceStateManager<ToolPanelStatesManager>()

// Simulation and world rendering

// Simulation renderer

.AddSingleton(PresentationConfiguration.GameOfLifeRendererOptions)

.AddSharedSingleton<ISimulationRendererOptions, GameOfLifeRendererOptions>()

.AddSingleton<GameOfLifeRenderer>()

.AddSharedSingleton<ISimulationRenderer, GameOfLifeRenderer>()

.AddServiceStateManager<GameOfLifeRendererStateManager>()

.AddStartupOperation<RenderSimulationOnStartupOperation>()

//

.AddSingleton<IWorldViewport, WorldViewport>()

// Camera

.AddSingleton(PresentationConfiguration.WorldCameraOptions)

.AddSingleton<IWorldCamera, WorldCamera>()

.AddSingleton<CameraSettingsViewModel>()

.AddServiceStateManager<WorldCameraStateManager>()

//

// World renderer

.AddSingleton<ICellularWorldRendererOptions>(PresentationConfiguration.CellularWorldRendererOptions)

.AddSingleton<BoundedCellularWorldRenderer>()

.AddSharedSingleton<IWorldRenderer, BoundedCellularWorldRenderer>()

.AddSharedSingleton<IBoundedWorldRenderer, BoundedCellularWorldRenderer>()

.AddSharedSingleton<ICellularWorldRenderer, BoundedCellularWorldRenderer>()

.AddSharedSingleton<IBoundedCellularWorldRenderer, BoundedCellularWorldRenderer>()

.AddServiceStateManager<BoundedCellularWorldRendererStateManager>()

//

.AddSingleton<GameOfLifeWorldViewModel>()

//

// Drawing on world

.AddSingleton(PresentationConfiguration.DrawingOptions)

.AddSharedSingleton<IDrawingOptions<GameOfLifeCellState>, GameOfLifeDrawingOptions>()

.AddSingleton<GameOfLifeDrawingSettings>()

.AddSharedSingleton<IDrawingSettings<GameOfLifeCellState>, GameOfLifeDrawingSettings>()

.AddServiceStateManager<GameOfLifeDrawingSettingsStateManager>()

.AddSingleton<IDrawingSettingsViewModel, DrawingSettingsViewModel>()

.AddSingleton<DrawingToolPanelViewModel>()

//

// History

.AddSingleton(PresentationConfiguration.HistoryOptions)

.AddSingleton<IHistoryManager<GameOfLifeSave>, HistoryManager<GameOfLifeSave>>()

.AddServiceStateManager<HistoryStateManager<GameOfLifeSave>>()

.AddSingleton<IHistoryViewModel, HistoryViewModel<GameOfLifeSave>>()

//

// Statistics

// Summary statistics

.AddSingleton(PresentationConfiguration.SummaryStatsOptions)

.AddSharedSingleton<ISummaryStatsOptions, SummaryStatsOptions>()

.AddSingleton<ISummaryStatsManager<GameOfLifeSummary>, SummaryStatsManager<GameOfLifeSummary>>()

// .AddSingleton<ISummaryStatisticsManager<LifeAutomationSummary>, HistoryAwareSummaryStatisticsManager<LifeAutomationSummary, ILifeAutomationSaving>>()

.AddServiceStateManager<SummaryStatsStateManager<GameOfLifeSummary>>()

.AddSingleton<ISummaryStatsViewModel, SummaryStatsViewModel<GameOfLifeSummary>>()

.AddServiceStateManager<SummaryStatsViewModelStateManager>()

// Summary statistics chart view models

.AddSingleton<SummaryStatsChartViewModelFactory>()

.AddTransient<TableChartViewModel>()

.AddTransient<PieChartViewModel>()

.AddTransient<LineChartViewModel>()

.AddTransient<BarChartViewModel>()

//

// Command execution statistics

.AddSingleton(PresentationConfiguration.CommandExecutionStatsOptions)

.AddSingleton<ICommandExecutionStatsManager, CommandExecutionStatsManager>()

.AddServiceStateManager<CommandExecutionStatsStateManager>()

.AddSingleton<CommandExecutionStatsViewModel>()

.AddServiceStateManager<CommandExecutionStatsViewModelStateManager>()

.AddSingleton<CommandExecutionChartViewModelFactory>()

.AddTransient<CommandExecutionLineChartViewModel>()

.AddTransient<CommandExecutionBarChartViewModel>()

//

// Simulation rendering statistics

.AddSingleton(PresentationConfiguration.RenderingStatsOptions)

.AddSingleton<IRenderingStatsManager, RenderingStatsManager>()

.AddServiceStateManager<RenderingStatsStateManager>()

.AddSingleton<RenderingStatsViewModel>()

.AddServiceStateManager<RenderingStatsViewModelStateManager>()

.AddSingleton<RenderingChartViewModelFactory>()

.AddTransient<RenderingLineChartViewModel>()

.AddTransient<RenderingBarChartViewModel>()

//

// Simulation tool panel

.AddSingleton<SimulationToolPanelViewModel>()

.AddSingleton<IWorldSizeViewModel, WorldSizeViewModel>()

.AddServiceStateManager<WorldSizeViewModelStateManager>()

.AddSingleton<IWorldWrappingViewModel, WorldWrappingViewModel<GameOfLifeCellState>>()

// Population

.AddSingleton(PresentationConfiguration.PopulationOptions)

.AddSingleton<PopulationViewModel>()

.AddServiceStateManager<PopulationViewModelStateManager>()

//

.AddSingleton<AlgorithmViewModel>()

.AddSingleton<PatternsViewModel>()

.AddSingleton<RuleViewModel>()

.AddSingleton<CommandQueueViewModel>()

.AddServiceStateManager<CommandQueueViewModelStateManager>()

//

// Rendering tool panel

.AddSingleton<RenderingToolPanelViewModel>()

.AddSingleton<SimulationRenderingSettingsViewModel>()

.AddSingleton<CellColorsViewModel>()

.AddSingleton<WorldBackgroundColorViewModel>()

.AddSingleton<WorldGridLinesViewModel>()

//

// Status bar

.AddSingleton(PresentationConfiguration.StatusBarOptions)

.AddSingleton<StatusBarViewModel>()

.AddServiceStateManager<StatusBarStateManager>()

//

.AddAppStateManagementServices(PresentationConfiguration.AppStatesOptions)

.AddStartupOperationManager();

}

Настройки отрисовщика симуляции, код предоставлен в файле GameOfLifeDrawingSettings.cs:

﻿using CommunityToolkit.Mvvm.ComponentModel;

using SimulationStorm.GameOfLife.DataTypes;

using SimulationStorm.GameOfLife.Presentation.Patterns;

using SimulationStorm.Simulation.CellularAutomation.Presentation.Services;

namespace SimulationStorm.GameOfLife.Presentation.Drawing;

public partial class GameOfLifeDrawingSettings(GameOfLifeDrawingOptions options) :

DrawingSettings<GameOfLifeCellState>(options)

{

#region Properties

[ObservableProperty] private NamedPattern? \_currentPattern = options.Pattern;

[ObservableProperty] private bool \_placePatternWithOverlay = options.PlacePatternWithOverlay;

#endregion

protected override void OnIsDrawingEnabledChanged()

{

if (IsDrawingEnabled)

CurrentPattern = null;

}

partial void OnCurrentPatternChanged(NamedPattern? value)

{

if (CurrentPattern is not null)

IsDrawingEnabled = false;

}

}

Основные категории шаблонов симуляции, код предоставлен в файле PredefinedPatternCategories.cs:

﻿using System.Collections.Generic;

namespace SimulationStorm.GameOfLife.Presentation.Patterns;

public static class PredefinedPatternCategories

{

private static readonly NamedPatternCategory

StillLifes = new(nameof(StillLifes), new[]

{

PredefinedPatterns.Block,

PredefinedPatterns.Beehive,

PredefinedPatterns.Loaf,

PredefinedPatterns.Tub,

PredefinedPatterns.Boat,

PredefinedPatterns.Carrier

}),

Oscillators = new(nameof(Oscillators), new[]

{

PredefinedPatterns.Blinker,

PredefinedPatterns.FigureEight,

PredefinedPatterns.Octagon,

PredefinedPatterns.Pentadecathlon,

PredefinedPatterns.SparkCoil,

PredefinedPatterns.Wheel

}),

Spaceships = new(nameof(Spaceships), new[]

{

PredefinedPatterns.LightweightSpaceship,

PredefinedPatterns.MediumweightSpaceship,

PredefinedPatterns.HeavyweightSpaceship,

PredefinedPatterns.Glider,

PredefinedPatterns.Loafer,

PredefinedPatterns.Moon

}),

Guns = new(nameof(Guns), new[]

{

PredefinedPatterns.GliderGun

}),

Methuselahs = new(nameof(Methuselahs), new[]

{

PredefinedPatterns.Acorn,

PredefinedPatterns.BHeptomino,

PredefinedPatterns.PHeptomino,

PredefinedPatterns.Diehard,

PredefinedPatterns.GliderByTheDozen,

PredefinedPatterns.Thunderbird

}),

Wicks = new(nameof(Wicks), new[]

{

PredefinedPatterns.Ants,

PredefinedPatterns.BlinkerFuse,

PredefinedPatterns.LightWave

});

public static readonly IEnumerable<NamedPatternCategory> All = new[]

{

StillLifes,

Oscillators,

Spaceships,

Guns,

Methuselahs,

Wicks

};

}

Категории правил симуляции, код предоставлен в файле PredefinedRuleCategories.cs:

﻿using System.Collections.Generic;

namespace SimulationStorm.GameOfLife.Presentation.Rules;

public static class PredefinedRuleCategories

{

public static readonly NamedRuleCategory

LifeFamily = new(nameof(LifeFamily), new[]

{

PredefinedRules.GameOfLife,

PredefinedRules.HighLife,

PredefinedRules.LongLife,

PredefinedRules.LifeWithoutDeath

}),

Mazes = new(nameof(Mazes), new[]

{

PredefinedRules.Maze,

PredefinedRules.Maze2,

PredefinedRules.Maze3,

PredefinedRules.Maze4

}),

Rugs = new(nameof(Rugs), new[]

{

PredefinedRules.PersianRug,

PredefinedRules.Rug2,

PredefinedRules.Rug3,

PredefinedRules.Rug4,

PredefinedRules.Rug5

}),

Others = new(nameof(Others), new[]

{

PredefinedRules.LiveFreeOrDie,

PredefinedRules.Seeds,

PredefinedRules.HTrees,

PredefinedRules.Diamoeba,

PredefinedRules.DayAndNight,

PredefinedRules.Assimilation,

PredefinedRules.Corals,

PredefinedRules.Coagulation,

PredefinedRules.Majority,

PredefinedRules.Annealing

});

public static readonly IEnumerable<NamedRuleCategory> All = new[]

{

LifeFamily,

Mazes,

Rugs,

Others

};

}