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 инструментария для создания, проведения, исследования и визуализации симуляций

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

Казань, 2024 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc169100515)

[1 АНАЛИТИЧЕСКАЯ ЧАСТЬ 8](#_Toc169100516)

[1.1 Анализ предметной области симуляций 8](#_Toc169100517)

[1.2 Анализ предметной области клеточных автоматов 9](#_Toc169100518)

[2 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ 10](#_Toc169100519)

[2.1 Обоснование выбора языка программирования 10](#_Toc169100520)

[2.2 Обоснование выбора фреймворка 11](#_Toc169100521)

[2.3 Обоснование выбора среды разработки 12](#_Toc169100522)

[2.4 Описание используемых библиотек 13](#_Toc169100523)

[2.4.1 Обоснование выбора библиотеки CommunityToolkit.Mvvm 14](#_Toc169100524)

[2.4.2 Обоснование выбора библиотеки Rx.Net 14](#_Toc169100525)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 15](#_Toc169100526)

[3.1 Диаграмма вариантов использования (Use case) 15](#_Toc169100527)

[3.2 Шаблон проектирования архитектуры приложения 16](#_Toc169100528)

[3.3 Файловая структура проекта 18](#_Toc169100529)

[4 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА 20](#_Toc169100530)

[4.1 Разработка модулей симуляций 20](#_Toc169100531)

[4.1.1 Общий модуль для всех симуляций 20](#_Toc169100532)

[4.1.2 Модуль ведения истории симуляции 21](#_Toc169100533)

[4.1.3 Модуль ведения статистики симуляции 22](#_Toc169100534)

[4.1.4 Модуль продвижения симуляции 23](#_Toc169100535)

[4.1.5 Модуль сброса симуляции к начальному состоянию 24](#_Toc169100536)

[4.1.6 Модуль для симуляций с ограниченным миром 25](#_Toc169100537)

[4.1.7 Модуль для клеточных симуляций 26](#_Toc169100538)

[4.1.8 Модуль для клеточных автоматов 27](#_Toc169100539)

[4.2 Разработка лаунчера симуляций 28](#_Toc169100540)

[5 Тестирование программного продукта 30](#_Toc169100541)

[5.1 Общие понятия тестирования 30](#_Toc169100542)

[5.2 Создание и реализация модульных тестов 30](#_Toc169100543)

[5.3 Тест-кейсы 32](#_Toc169100544)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 37](#_Toc169100545)

[6.1 Подготовительный этап 37](#_Toc169100546)

[6.2 Разработка доменной модели симуляции 39](#_Toc169100547)

[6.3 Разработка проекта уровня представления. 40](#_Toc169100548)

[6.4 Разработка проекта уровня пользовательского интерфейса. 43](#_Toc169100549)

[ЗАКЛЮЧЕНИЕ 44](#_Toc169100550)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 46](#_Toc169100551)

[ПРИЛОЖЕНИЕ А 48](#_Toc169100552)

[ПРИЛОЖЕНИЕ Б 49](#_Toc169100553)

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 Хасанову Ильназу Дамировичу по специальности 09.02.07 Информационные системы и программирование, квалификация: Специалист по информационным системам

1. Тема дипломного проекта: «Проектирование и разработка инструментария для создания, проведения, визуализации и исследования симуляций», 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, обоснование выбора средства разработки, проектирование программного продукта, реализация проекта, тестирование программного продукта, руководство пользователя.
   2. Требования к программному продукту: поддержка ОС Windows и Linux, поддержка русского и английского языков, интуитивно понятный интерфейс, отказоустойчивость, надёжность и безопасность

5 Перечень графических материалов/Приложения: техническое задание 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

*подпись инициалы и фамилия*

Дата выдачи задания:

25.03. 2024 г.

Руководитель дипломного проекта:

Директор

ИЦТЭ ФГБОУ ВО «КГЭУ» Э. И. Беляев

*подпись инициалы и фамилия*

Задание принял к исполнению: И. Д. Хасанов в

*подпись инициалы и фамилия*

# ВВЕДЕНИЕ

Симуляции играют ключевую роль в современной науке и технологиях, предоставляя возможность компьютерной реализации различных реальных или вымышленных процессов и явлений. Значимость симуляций проявляется во многих областях, включая науку, инженерию, медицину, экономику, образование и развлечения.

С помощью симуляций, исследователи и специалисты могут создавать виртуальные модели сложных систем, в которых могут проводить различные эксперименты, тестировать гипотезы и анализировать результаты без риска повреждения реальных объектов или опасности для жизни и здоровья. Это позволяет более глубоко понять взаимодействия компонентов системы, выявить потенциальные проблемы и недочеты, а также оптимизировать процессы до их фактической реализации.

В ходе реализации данного проекта были поставлены и успешно решены следующие задачи:

* анализ предметной области симуляций;
* выделение общих свойств и характеристик симуляций;
* проектирование модулей, реализующих общие свойства и характеристики симуляций;
* разработка модулей по результатам проектирования;
* проведение тестирования разработанных модулей.

На данный момент времени разработка компьютерных симуляций различных физических и нефизических процессов или явлений предполагает создание с нуля всех компонентов.

Для того, чтобы избавить разработчиков от необходимости создавать с нуля все компоненты, создан инструментарий, который призван упростить и ускорить процесс разработки компьютерных симуляций, путём предоставления модулей, реализующих типовые повторяющиеся свойства и характеристики симуляций.

Основные требования, предъявленные к проекту:

* кроссплатформенность / поддержка десктопных ОС (Windows, Linux и Mac) и Web;
* локализация / поддержка нескольких языков;
* удобный, интуитивно понятный, адаптивный пользовательский интерфейс;
* наличие отдельной документации и обучающего режима непосредственно в программе;
* лёгкость расширения функционала и возможностей программы в дальнейшей перспективе;
* производительность;
* отказоустойчивость;
* надёжность.

Разработка инструментария в данной области имеет большое значение не только для индивидуальных разработчиков, но и для широкого круга пользователей и отраслей. Универсальный инструментарий позволит существенно увеличить эффективность работы и сэкономить время и ресурсы. Благодаря такому инструментарию и его возможностям, исследователи и разработчики смогут быстрее и эффективнее экспериментировать с различными симуляциями.

Техническое задание представлено в приложении А [1].

# 1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

## Анализ предметной области симуляций

Под симуляцией понимается компьютерная реализация какого-либо реального или придуманного процесса или явления, а под созданием симуляции понимается разработка программы, которая будет реализовывать желаемый процесс.

На данный момент, разработка большинства симуляций выполняется без использования каких-либо специализированных инструментов и представляет собой создание алгоритмов непосредственно самой симуляции и связывание алгоритмов с пользовательским интерфейсом.

Описанный процесс поддаётся упрощению. В основе каждой симуляции лежит ряд одних и тех же повторяющихся модулей, которые могут быть один раз разработаны, а затем использованы в любой создаваемой симуляции.

В ходе анализа предметной области симуляций были выделены следующие повторяющиеся модули:

* общий модуль для всех симуляций;
* модуль ведения истории симуляции;
* модуль ведения статистики симуляции;
* модуль продвижения симуляции;
* модуль сброса симуляции к начальному состоянию;
* модуль для симуляций с ограниченным миром;
* модуль для клеточных симуляций;
* модуль для клеточных автоматов.

Одни из модулей являются необходимыми для большинства симуляций, другие используются в зависимости от уникального типа симуляции. Так, модуль продвижения, модуль ведения истории, модуль ведения статистики, модуль сброса к исходному состоянию используются в большинстве симуляций, а другие, такие как модули для клеточных симуляций являются опциональными.

## 1.2 Анализ предметной области клеточных автоматов

Детального рассмотрения заслуживает модуль для клеточных автоматов.

Клеточные автоматы — это абстрактные модели, состоящие из сетки ячеек, которые могут находиться в различных состояниях, и правил, определяющих, как эти состояния обновляются в дискретном времени на основе состояния соседних ячеек [1].

Стивен Вольфрам в своей книге A New Kind of Science предложил 4 класса, на которые все клеточные автоматы могут быть разделены в зависимости от типа их эволюции.

Классификация клеточных автоматов по типам поведения:

* класс 1, результатом эволюции начальных условий является быстрый переход к гомогенной стабильности. Любые негомогенные конструкции быстро исчезают;
* класс 2, результатом эволюции начальных условий является быстрый переход в неизменяемое негомогенное состояние либо возникновение циклической последовательности. Большинство структур начальных условий быстро исчезает, но некоторые остаются;
* класс 3, результатом эволюции почти всех начальных условий являются псевдо-случайные, хаотические последовательности. Любые стабильные структуры, которые возникают почти сразу же уничтожаются окружающим их шумом. Локальные изменения в начальных условиях оказывают неопределяемое влияние на ход эволюции системы;
* класс 4, результатом эволюции являются структуры, которые взаимодействуют сложным образом с формированием локальных, устойчивых структур. В результате эволюции могут получаться некоторые последовательности Класса 2. Локальные изменения в начальных условиях оказывают неопределяемое влияние на ход эволюции системы.

# 2 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ

## 2.1 Обоснование выбора языка программирования

C# является современным, ориентированным на объекты и безопасным языком программирования, предназначенным для создания надежных приложений в среде .NET. Он унаследовал синтаксис и концепции от языков семейства C, включая C, C++, Java и JavaScript [2].

Основой C# является объектно-ориентированная парадигма с поддержкой компонентной архитектуры, что облегчает создание и использование программных компонентов. C# также предоставляет специальные конструкции, поддерживающие эту концепцию.

Преимущества надежности и безопасности приложений на C# достигаются через различные механизмы, такие как сборка мусора для управления памятью, типы, допускающие значение null для защиты от ошибок, обработка исключений для управления ошибками, лямбда-выражения для поддержки функционального программирования и другие.

Важным аспектом C# является единая система типов, где все типы, включая примитивные, происходят от корневого типа object. Это обеспечивает единообразие операций над данными и поддерживает как ссылочные, так и значимые типы.

Кроме того, C# предоставляет широкий спектр функциональных возможностей. Это включает в себя LINQ для работы с данными, асинхронные операции, итераторы для собственных коллекций, а также управление версиями для обеспечения совместимости с программами и библиотеками, а также интеграцию с другими технологиями Microsoft.

Ключевыми достоинствами С# является такие преимущества как:

* C# получает поддержку от крупной корпорации Microsoft, обеспечивая стабильность, регулярные обновления и обширную документацию;
* C# предоставляет удобные средства для работы с многозадачностью и многопоточностью, что позволяет создавать масштабируемые и эффективные приложения;
* C# является основным языком разработки на платформе .NET, что обеспечивает доступ к обширным библиотекам и инструментам для создания разнообразных приложений, включая веб-сайты, десктопные приложения, мобильные приложения и игры;
* система строгой типизации C# обеспечивает безопасность типов на этапе компиляции, что помогает избежать множества ошибок во время выполнения программы;
* регулярные обновления C# вводят новые возможности и улучшения, делая язык более современным и функциональным;
* с развитием проекта .NET Core и .NET 5+ C# становится кроссплатформенным языком, что позволяет разрабатывать приложения для различных операционных систем, включая Windows, Linux и macOS;

Интеграция C# с другими технологиями Microsoft, такими как SQL Server, Azure и др., делает его предпочтительным выбором для разработки корпоративных приложений.

## 2.2 Обоснование выбора фреймворка

Avalonia — это кроссплатформенная платформа пользовательского интерфейса, основанная на XAML, которая имеет ряд преимуществ и достоинств, которые сильно повлияли на его выбор, как ключевой платформы интерфейса приложения:

* Avalonia обеспечивает возможность создания графических приложений, которые могут работать на различных операционных системах, включая Windows, macOS и Linux, что расширяет аудиторию потенциальных пользователей;
* **мощная графическая библиотека** предоставляет обширные возможности для создания современного и привлекательного пользовательского интерфейса с использованием различных элементов управления, анимаций и стилей;
* использование C# позволяет разработчикам писать код более эффективно и продуктивно благодаря его современным функциональным возможностям, типизации и интеграции с другими технологиями Microsoft;
* Avalonia предоставляет множество возможностей для расширения функциональности приложений с помощью сторонних библиотек и плагинов, что способствует созданию высокопроизводительных и инновационных приложений;
* Avalonia предоставляет поддержку популярной архитектурной модели MVVM (Model-View-ViewModel), что способствует разделению логики приложения и его пользовательского интерфейса;

Существует активное сообщество разработчиков, которые активно поддерживают и развивают Avalonia, предоставляя обновления, исправления ошибок и документацию, что обеспечивает надежность и стабильность платформы [3].

## 2.3 Обоснование выбора среды разработки

JetBrains Rider — это интегрированная среда разработки (IDE) для платформы .NET, разработанная компанией JetBrains. [4]

JetBrains Rider представляет собой полнофункциональную интегрированную среду разработки, которая объединяет в себе множество инструментов, необходимых для эффективной работы над проектами на платформе .NET.

Мощный отладчик облегчает процесс исправления ошибок, предоставляя обширные возможности, такие как точки останова и просмотр значений переменных.

Кроме того, Rider обеспечивает поддержку различных языков программирования, что делает его универсальным инструментом для разработки многоязычных проектов. Улучшенные функции рефакторинга помогают улучшить структуру кода, снизить его дублирование и повысить читаемость.

К тому же, возможность расширения функциональности через плагины делает Rider еще более гибким и адаптируемым к индивидуальным потребностям разработчиков.

Так же стоит отметить ключевые достоинства JetBrains Rider:

* интеграция с другими инструментами разработки JetBrains;
* высокая производительность и стабильность;
* широкие возможности настройки и персонализации;
* поддержка различных платформ, включая Windows, macOS и Linux;
* регулярные обновления и поддержка со стороны JetBrains.

Регулярные обновления является одним из самых важных из ключевых достоинств JetBrains Rider, это позволяет внедрять новые возможности, исправлять ошибки и повышать производительность языка и средств разработки.

## 2.4 Описание используемых библиотек

Для реализации основных функций программы, в проекте были установлены следующие библиотеки:

* CommunityToolkit.Mvvm, библиотека, предоставляющая инструменты для удобного применения шаблона проектирования MVVM;
* Rx.Net, библиотека, предоставляющая возможности реактивного программирования для .NET. Rx позволяет создавать события и обработчики, которые будут реагировать на них. Таким образом, система состоит из последовательности событий, которые будут сообщать об изменении состояния и должным образом реагировать на них;
* DotNext, библиотека, предназначенная для облегчения работы с асинхронными операциями и другими аспектами параллельного программирования. Предоставляет различные инструменты и классы для управления потоками, блокировки и другие средства синхронизации.

Перечисленные библиотеки значительно упрощают разработку приложений с пользовательским интерфейсом.

### 2.4.1 Обоснование выбора библиотеки CommunityToolkit.Mvvm

CommunityToolkit.Mvvm — это часть библиотеки Windows Community Toolkit, предназначенная для разработчиков, использующих платформу Universal Windows Platform (UWP) и приложения на базе .NET. Библиотека предоставляет инструменты и шаблоны для реализации шаблона проектирования MVVM (Model-View-ViewModel). Также, библиотека помогает упростить разработку приложений, разделяя логику приложения от его пользовательского интерфейса и обеспечивая более простое тестирование и поддержку кода. Набор средств MVVM поддерживается и публикуется корпорацией Майкрософт, а также частью .NET Foundation. Ещё одной причиной выбора данной библиотеки является то, эта библиотека используется несколькими сторонними приложениями, встроенными в Windows. Этот пакет направлен на то, чтобы обеспечить максимальную гибкость, поэтому разработчики могут выбирать, какие компоненты следует использовать. Все типы слабо связаны, поэтому необходимо включить только то, что вы используете [5].

### 2.4.2 Обоснование выбора библиотеки Rx.Net

Rx.Net — это библиотека, предоставляющая возможности реактивного программирования для .NET. Библиотека позволяет разработчикам создавать асинхронные и событийно-ориентированные приложения, обрабатывая потоки данных с помощью наблюдаемых последовательностей (Observables) и операторов, что упрощает управление асинхронными операциями и обработку событий. Rx.Net способствует созданию реактивных, отзывчивых и легко масштабируемых приложений, повышая их производительность и читаемость кода. Библиотека реактивных расширений расширяет шаблон наблюдателя и включает состав асинхронных и событийных программ. Также, библиотека позволяет обрабатывать потоки асинхронных событий с такими же простыми и компонуемыми операциями, которые используются для коллекций элементов данных, таких как массивы [6].

# 3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

## 3.1 Диаграмма вариантов использования (Use case)

Диаграмма вариантов использования (Use case) — это детальное описание взаимодействия между системой и ее актерами с целью достижения определенного результата. Он представляет собой сценарий или последовательность действий, которые выполняются для достижения конкретной цели или выполнения определенной функции (Рисунок 3.1).

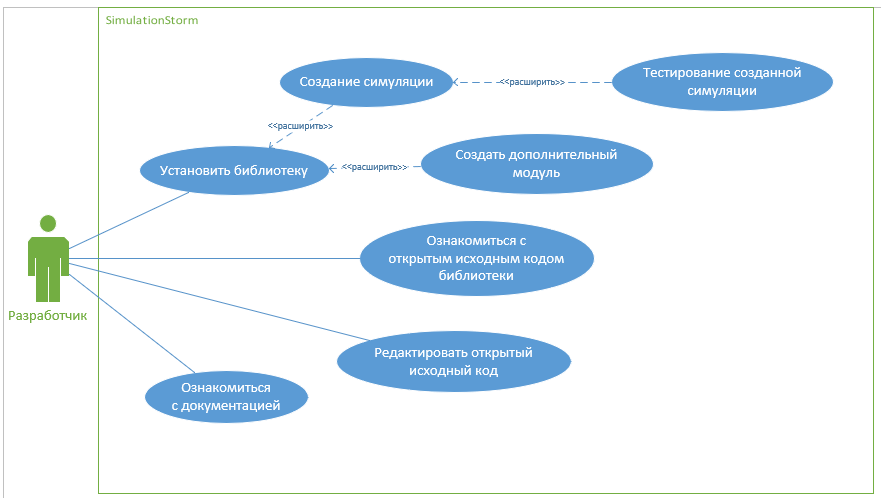


Рисунок 3.1 Диаграмма вариантов использования

Диаграмма вариантов использования служит для определения функциональных требований системы и становится основой для проектирования и разработки приложения.

Основное назначение диаграммы вариантов использования – это описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

Анализ вариантов использования помогает более точно определить, как система будет взаимодействовать с пользователями и обеспечить функциональность, отвечающую их потребностям.

## 3.2 Шаблон проектирования архитектуры приложения

Основным шаблоном проектирования архитектуры приложения был выбран шаблон MVVM [7].

Model-View-ViewModel (MVVM) — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model (Рисунок 3.2).

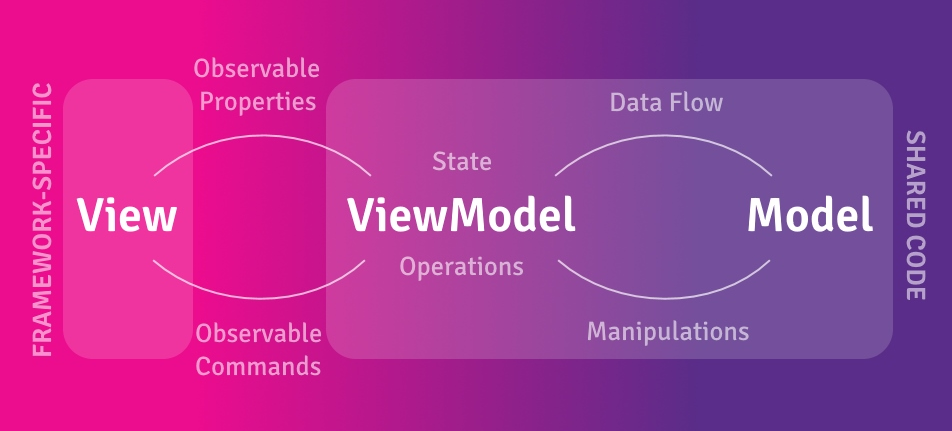


Рисунок 3.2 шаблон проектирования архитектуры приложения

Шаблон Model-View-ViewModel (MVVM) предоставляет эффективный подход к разработке кроссплатформенных приложений .NET, который способствует созданию более переносимого и удобного в обслуживании кода. MVVM разделяет приложение на три основных компонента: Model (модель данных), View (представление) и ViewModel (модель представления), что позволяет четко структурировать код и разделить логику отображения от бизнес-логики.

Использование MVVM значительно упрощает тестирование приложений, поскольку позволяет легко создавать и выполнять автоматизированные тесты для каждого компонента независимо друг от друга. Благодаря этому улучшаются возможности обнаружения и исправления ошибок, а также обеспечивается более высокая стабильность и надежность приложения.

Кроме того, использование шаблона MVVM позволяет увеличить объем кода, который можно переиспользовать на разных платформах, так как бизнес-логика и логика отображения отделены друг от друга.

Данный подход увеличивает эффективность разработки кроссплатформенных приложений и снижает объем дублирующегося кода, что в свою очередь способствует повышению производительности и упрощает поддержку приложения.

Привязка данных — это ключевая технология, которая позволяет приложению MVVM Avalonia UI отделять представления от моделей представлений (Рисунок 3.3).

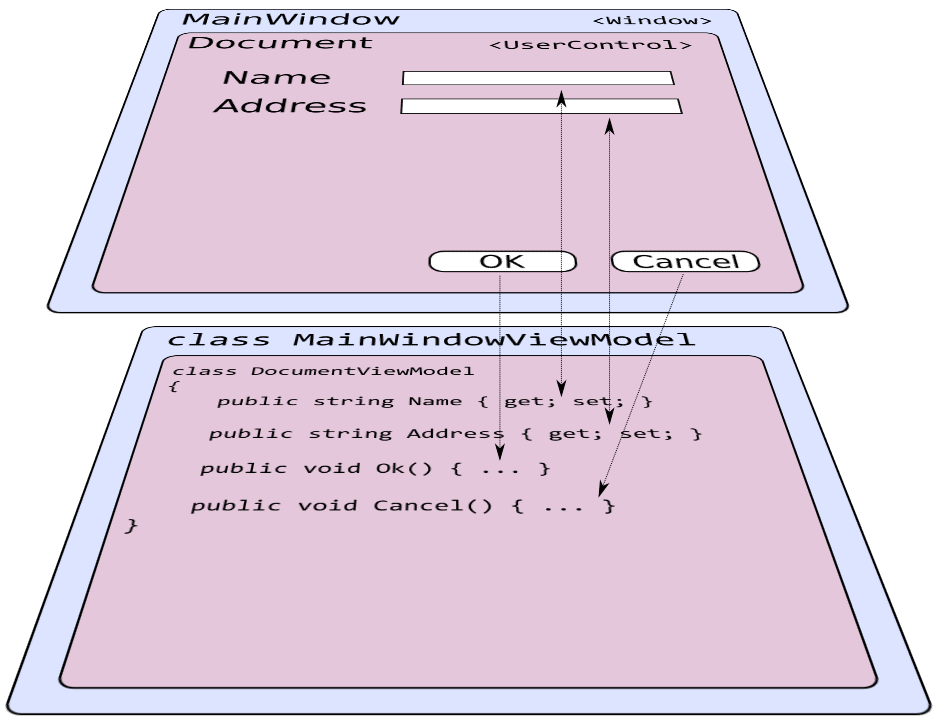


Рисунок 3.3 Привязка данных в архитектуре MVVM

Благодаря привязке данных приложение способно отделять представления от моделей представлений, что улучшает модульность и управляемость кода. Это означает, что изменения в модели данных автоматически отражаются в пользовательском интерфейсе без необходимости вручную обновлять каждый элемент.

Таким образом, привязка данных в сочетании с шаблоном MVVM в Avalonia UI помогает повысить разделение концернов в приложении, упрощает его разработку и обслуживание, а также улучшает пользовательский опыт за счет более эффективного управления данными и их отображением.

## 3.3 Файловая структура проекта

В ходе разработки проекта, для более удобочитаемости в процессе реализации была построена диаграмма состава проекта, которая описывает основные модули и общую файловую структуру проекта (Рисунок 3.4).

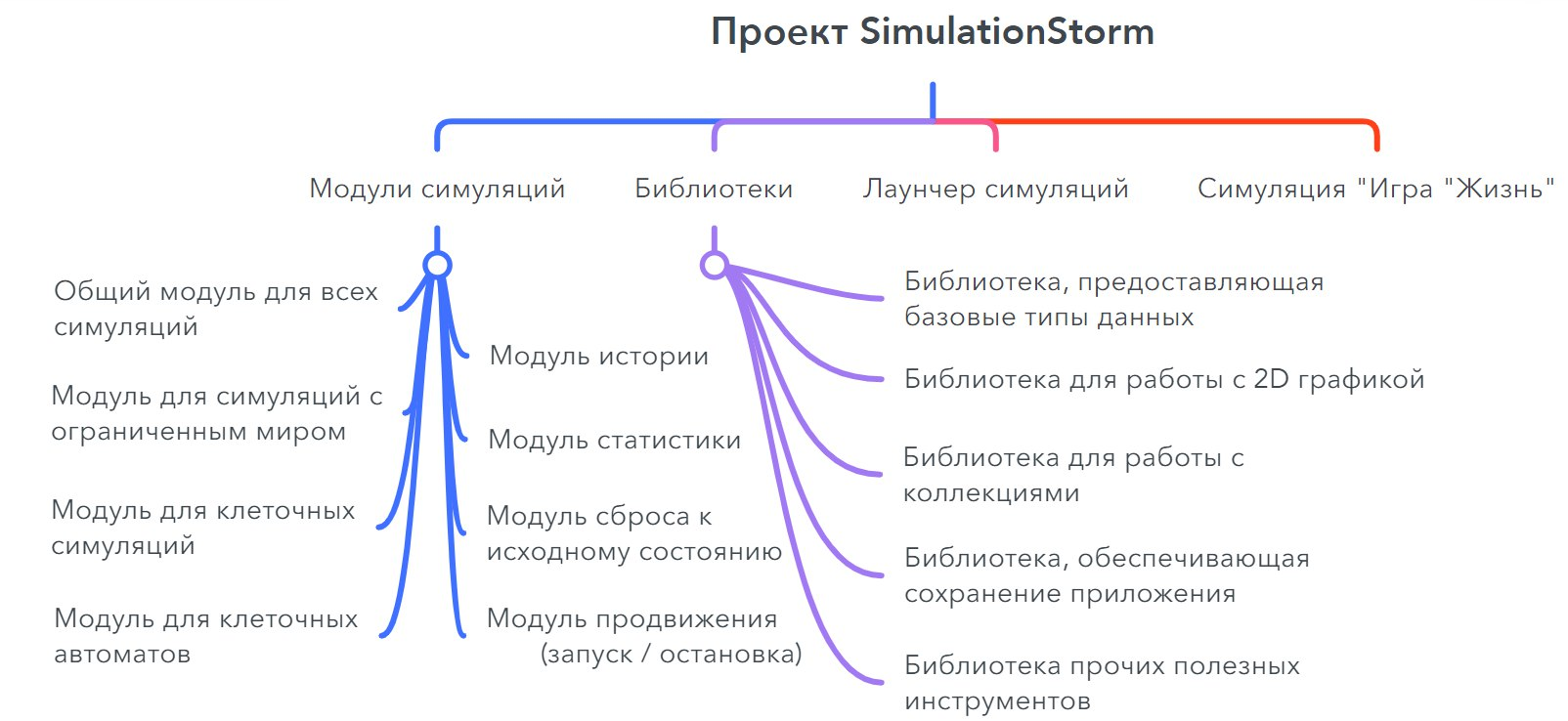


Рисунок 3.4 Диаграмма состава проекта

Стоит отметить, что почти каждый модуль проекта, представленный на диаграмме, подразделяется на подмодули.

Основные классы были организованы в разные каталоги проекта (Рисунок 3.5).

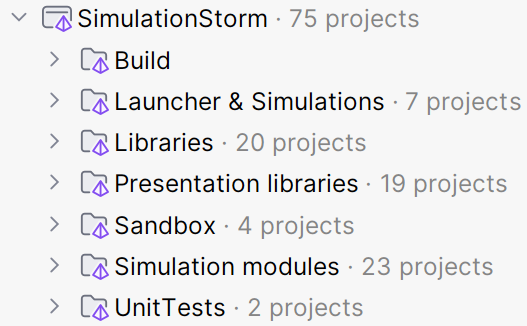


Рисунок 3.5 Структура всего проекта

Разделение модулей и классов в разные папки обеспечивает более структурированный и удобочитаемый проект.

Стоит выделить основные модули проекта, находящие в разных каталогах. Одним из основных для работы с графикой в проекте, является каталог «Graphics» (Рисунок 3.6).

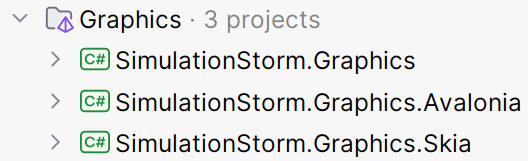


Рисунок 3.6 Библиотека для работы с графикой

Так же стоит упомянуть каталог содержащий Лаунчер симуляций (Рисунок 3.7).

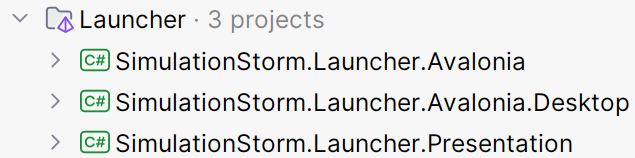


Рисунок 3.7 Каталог содержащий Лаунчер

Все составные модули сгруппированы в папки решения для удобства работы.

В процессе разработки был создан специальный каталог для проведения модульного тестирования проекта, с целью облегчения использования тестов (Рисунок 3.8).

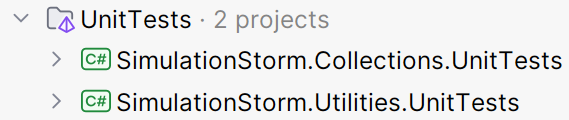


Рисунок 3.8 Проекты модульного тестирования

Также, стоит выделить проект, который предоставляет набор инструментов для работы с коллекциями (Рисунок 3.9).

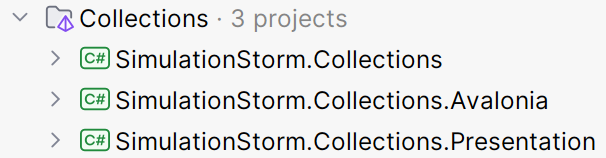


Рисунок 3.9 Инструменты для работы с коллекциями

Данные инструмент дают возможность гибко и удобно использовать в разработке разные типы, коллекции, а также преобразовывать одни виды в другие.

# 4 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

## 4.1 Разработка модулей симуляций

### 4.1.1 Общий модуль для всех симуляций

Общий модуль для всех симуляций называется Common. Данный модуль предоставляет самые необходимые компоненты, которые являются общими абсолютно для всех симуляций (Рисунок 4.1).

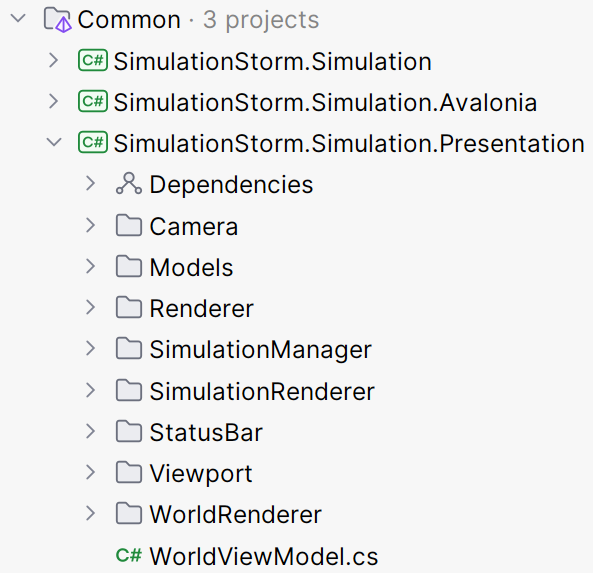


Рисунок 4.1 Общий модуль для всех симуляций

Ключевым компонентом данного модуля являются менеджер симуляций, код которого приведён в приложении и сервисы отрисовки.

Менеджер симуляций предназначен для управления командами или действиями симуляции. Основной функцией данного сервиса является получение действия, организация действия в очередь, а затем выполнение всех действий в очереди в том порядке, в котором действия были добавлены. Выполнение действий происходит в отдельном потоке, причём при окончании выполнения действия, вызывающая сторона получит об этом уведомление.

Сервисы отрисовки служат для визуализации симуляции и применения трансформаций камеры к визуализируемому полю.

### 4.1.2 Модуль ведения истории симуляции

Главной функцией данного модуля является сохранение состояния симуляции при её изменении, а также отображение и возможность возврата к ранее сохраненной истории симуляции (Рисунок 4.2).

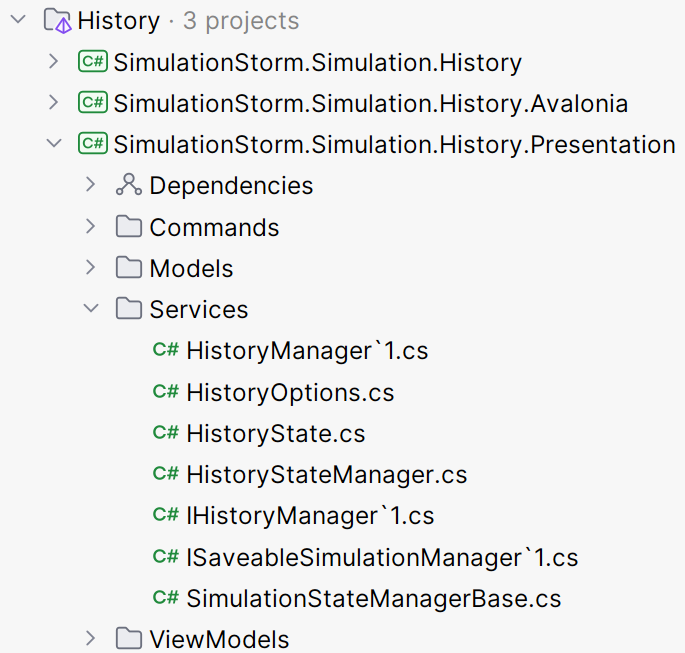


Рисунок 4.2 Модуль ведения истории симуляции

Также, имеется возможность задания интервала, с которым будет выполняться сохранение, возможность задания ёмкости хранилища сохранений и возможность изменения устройства физического расположения сохранений симуляции: оперативная память или файловая система.

Для того, чтобы использовать этот модуль, менеджер симуляции должен реализовать интерфейс ISaveable, тем самым предоставляя метод для выполнения сохранения текущего состояния симуляции, и метод для восстановления состояния симуляции из ранее выполненного сохранения.

Сбор истории работает таким образом, что менеджер истории наблюдает изменение состояния симуляции и при каждом изменении симуляции обращается к менеджеру симуляции, выполняет метод сохранения симуляции, а затем записывает выполненное сохранение в коллекцию записей истории симуляции.

### 4.1.3 Модуль ведения статистики симуляции

Данный модуль реализует ведение статистики состояний симуляции и отображение записей в виде различных диаграмм (Рисунок 4.3).

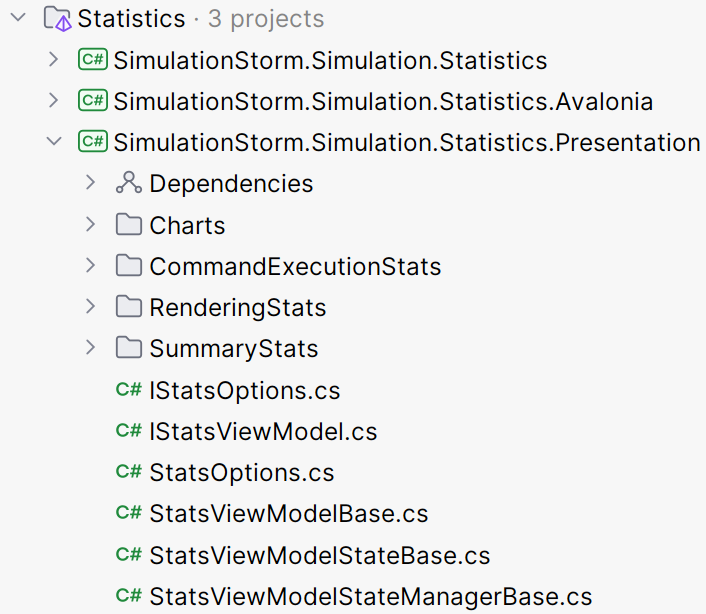


Рисунок 4.3 Модуль ведения статистики симуляции

Работа модуля статистики основывается на том факте, что менеджер симуляции реализует интерфейс ISummarizable. Данный интерфейс предоставляет единственный метод для генерации итогов по текущему состоянию симуляции.

Хранилище статистики, также, как и хранилище записей истории, предоставляет возможность задания интервала выполнения сохранения статистики, а также возможность изменения ёмкости хранилища записей статистики.

Для того, чтобы при выполнении сохранения всего приложения текущая статистика симуляции была сохранена, существует менеджер состояния статистики.

Помимо сказанного, данный модуль позволяет вести статистику по времени выполнения шагов симуляции и по времени визуализации симуляции. Для замера времени выполнения подобных действий используется сервис BenchmarkingService.

### 4.1.4 Модуль продвижения симуляции

Данный модуль предоставляет возможность автоматического продвижения симуляции с регулируемой скоростью, с возможностью запуска и остановки продвижения (Рисунок 4.4).

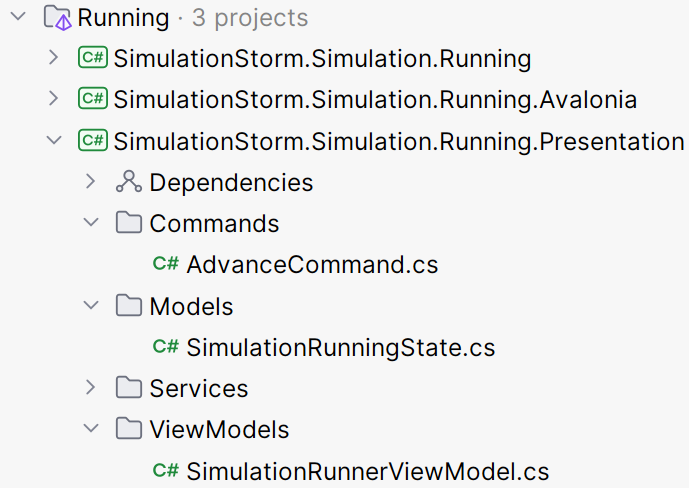


Рисунок 4.4 Модуль продвижения симуляции

Для того, чтобы использовать модуль продвижения, симуляция должна реализовывать интерфейс IAdvanceableSimulation. Данный интерфейс предоставляет единственный метод Advance, который выполняет продвижение симуляции до следующего состояния на основании текущего.

Управление продвижением симуляции выполняется с помощью сервиса SimulationRunner (Рисунок 4.5).

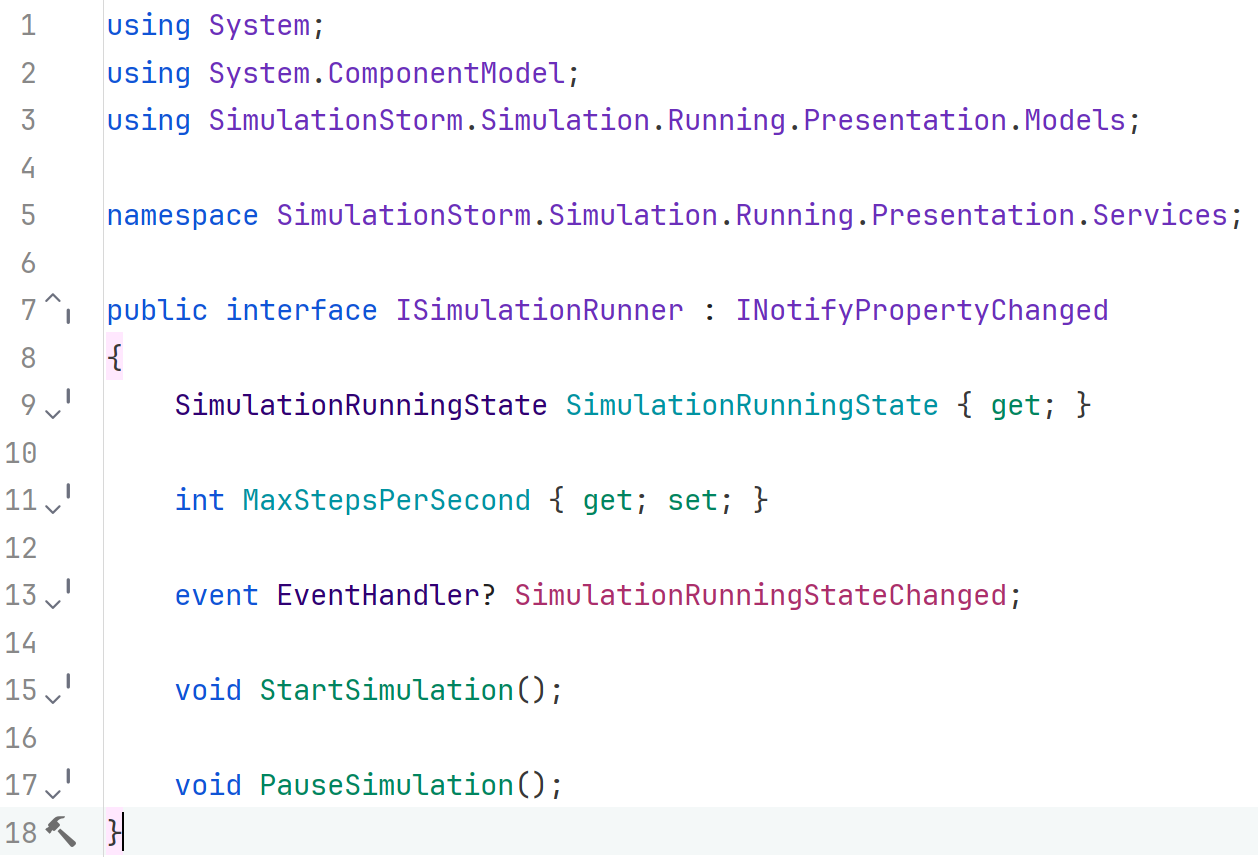


Рисунок 4.5 Сервис SimulationRunner

Этот сервис использует обеспечивает вызов метода симуляции Advance через заданные промежутки времени

### 4.1.5 Модуль сброса симуляции к начальному состоянию

Этот модуль предоставляет одну единственную функцию – возможность сброса симуляции к состоянию, в котором симуляция находилась при запуске (Рисунок 4.6).

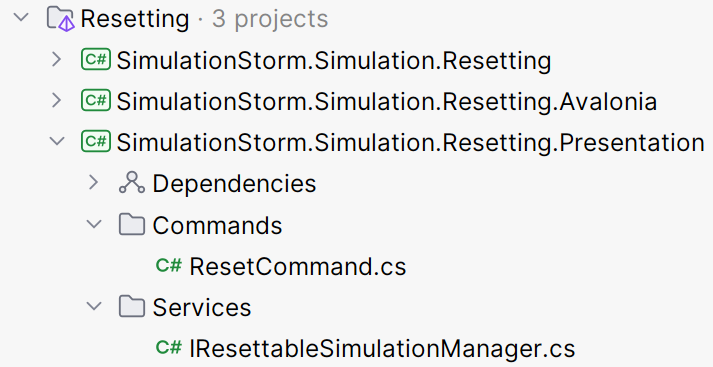


Рисунок 4.6 Модуль сброса симуляции к начальному состоянию

Для использования этого модуля, менеджер симуляции должен реализовать интерфейс IResettableSimulationManager, который предоставляет метод Reset, выполняющий сброс симуляции к исходному состоянию (Рисунок 4.7).

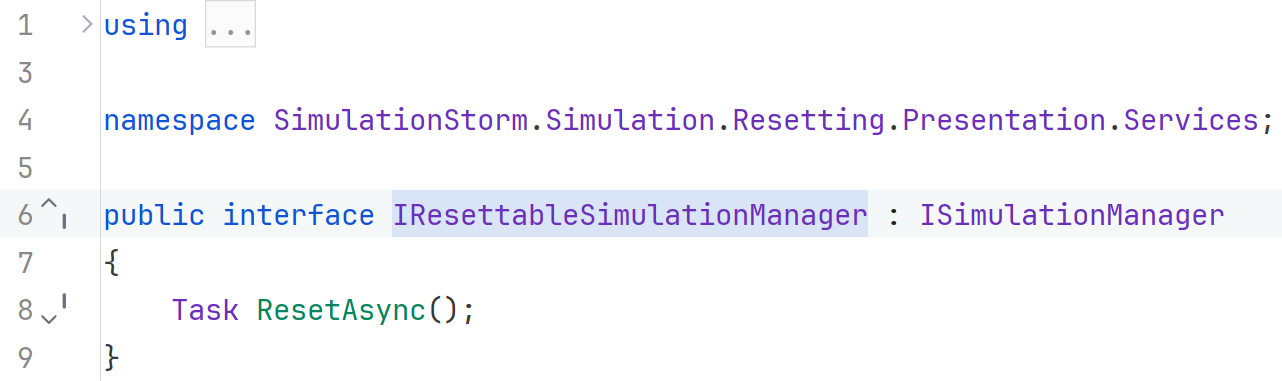


Рисунок 4.7 Менеджер симуляции с возможностью сброса

Когда вызывается метод Reset, менеджер симуляции создаёт объект, представляющий команду, и ставит созданный объект команды в очередь команд (Рисунок 4.8).

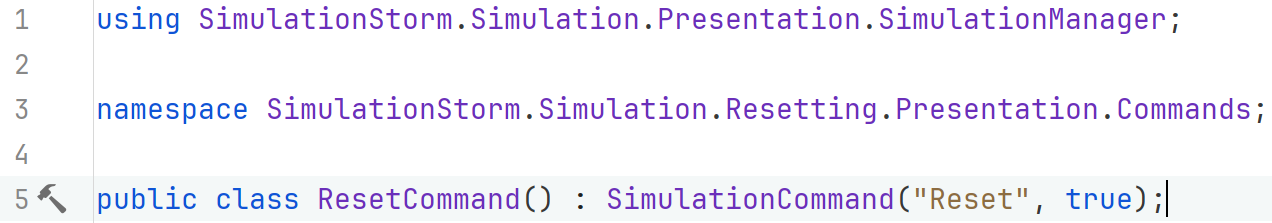


Рисунок 4.8 Команда сброса симуляции

Позже, эта команда будет выполнена менеджером симуляции.

### 4.1.6 Модуль для симуляций с ограниченным миром

Данный модуль необходим для симуляций, в которых размер мира (или поля) является ограниченным. Этот модуль предоставляет возможность изменения размера мира симуляции, а также компонент отрисовки, способный визуализировать симуляцию с ограниченным миром и настраивать параметры визуализации (Рисунок 4.9).

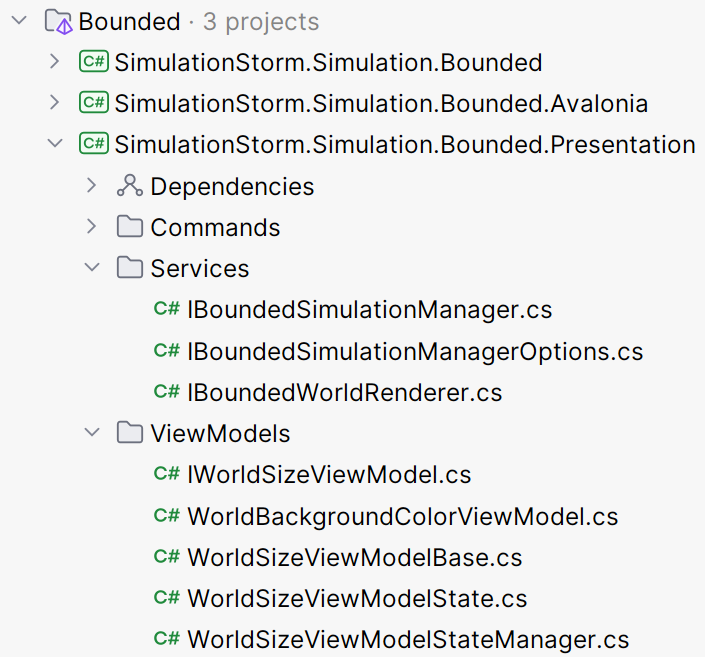


Рисунок 4.9 Модуль для симуляций с ограниченным миром

Для того, чтобы использовать возможности, предоставляемые данным модулем, менеджер симуляции должен реализовать интерфейс IBoundedSimulationManager (Рисунок 4.10).

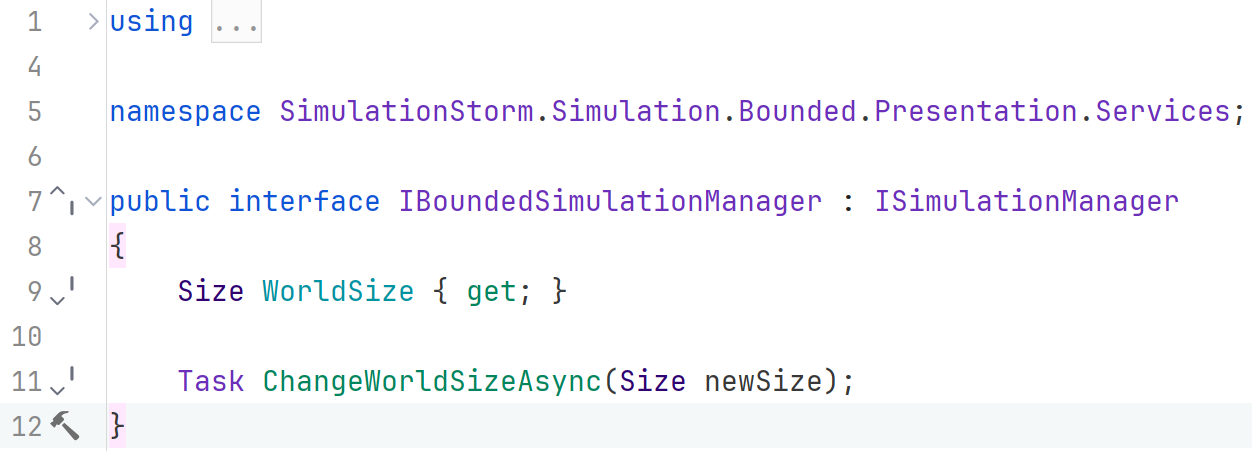


Рисунок 4.10 Менеджер симуляции с ограниченным миром

Данный интерфейс предоставляет свойство WorldSize, возвращающее текущий размер мира симуляции, а также метод ChangeWorldSize, который изменяет размер мира симуляции в соответствии с переданным значением.

### 4.1.7 Модуль для клеточных симуляций

Модуль для клеточных симуляций предоставляет компоненты для эффективной визуализации клеточных симуляций (Рисунок 4.11).

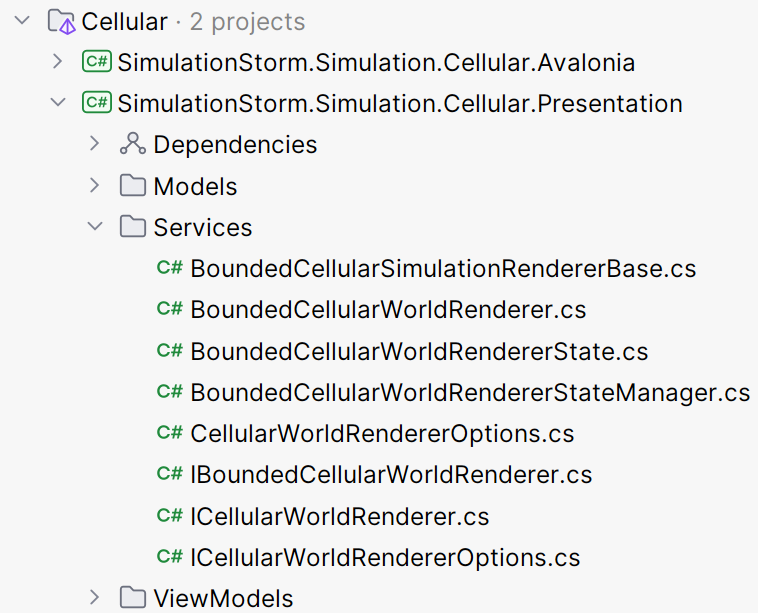


Рисунок 4.11 Модуль для клеточных симуляций

Наиболее интересным компонентом данного модуля является сервис отрисовки мира симуляции, который специализируется на отрисовке конкретно клеточных симуляций (Рисунок 4.12).

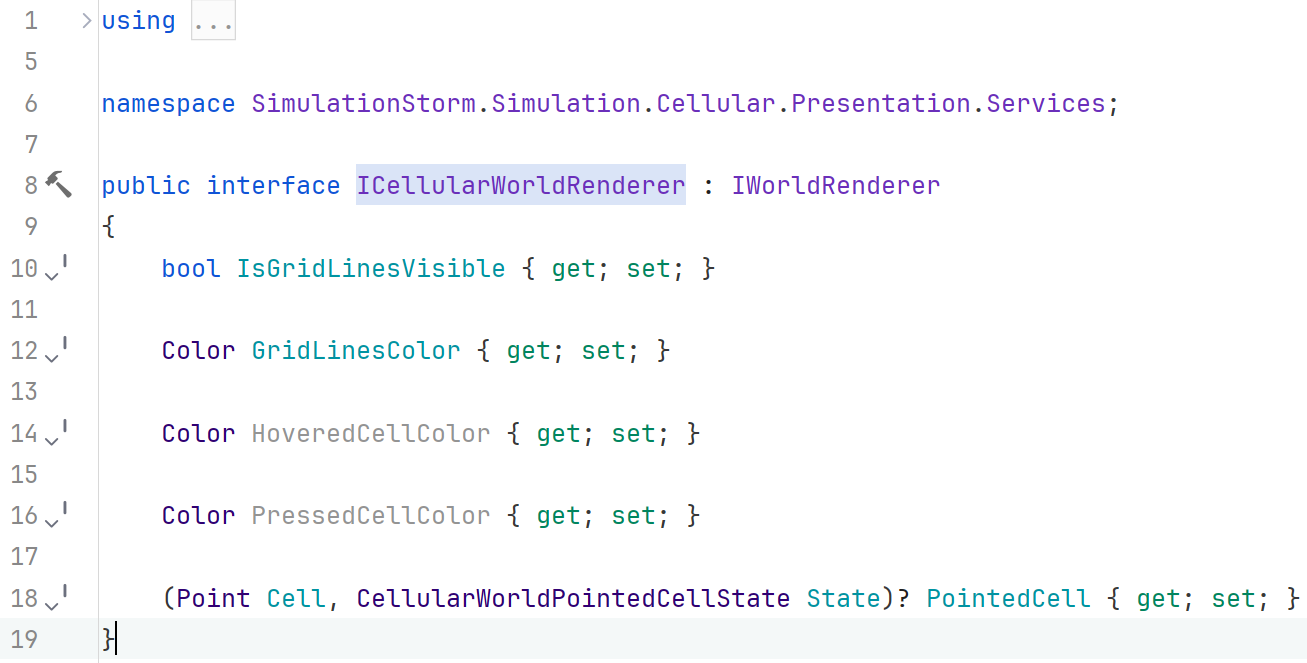


Рисунок 4.12 Сервис отрисовки мира клеточной симуляции

Данный сервис предоставляет свойства, с помощью которых можно управлять видимостью и цветом границ клеток (или линий сетки) поля, а также задавать клетку, над которой в данный момент находится курсор мыши, для визуализации этой клетки.

### 4.1.8 Модуль для клеточных автоматов

Этот модуль позволяет изменять состояния отдельных клеток поля клеточного автомата и рисовать произвольные фигуры, создавая различные изображения и узоры (Рисунок 4.13).

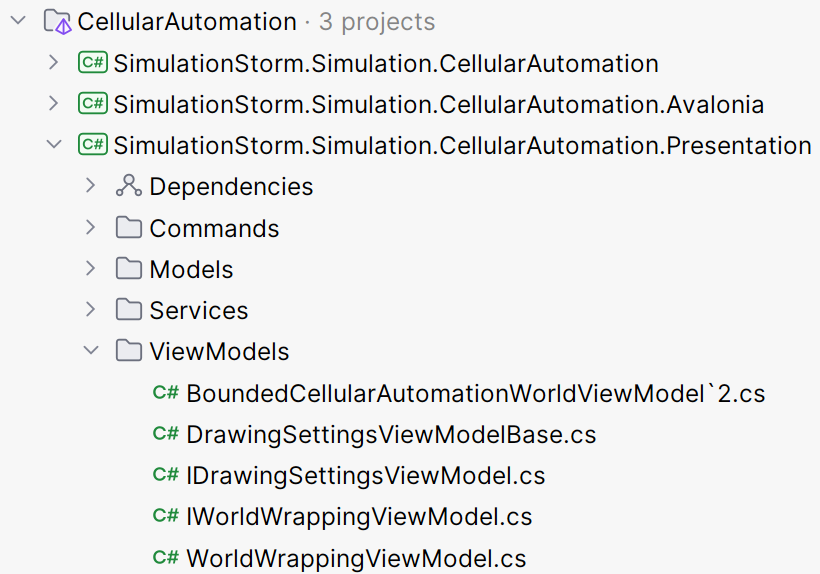


Рисунок 4.13 Модуль для клеточных автоматов

Имеется производный интерфейс, который представляет собой клеточный автомат с ограниченным размером мира (Рисунок 4.14).

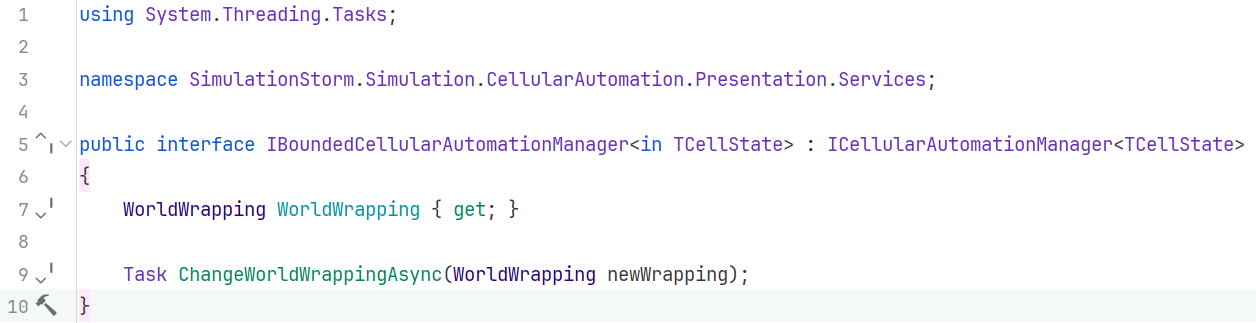


Рисунок 4.14 Менеджер клеточного автомата с ограниченным миром

Для того, чтобы использовать возможности, предоставляемые данным модулем, менеджер симуляции должен реализовывать интерфейс ICellularAutomationManager (Рисунок 4.15).

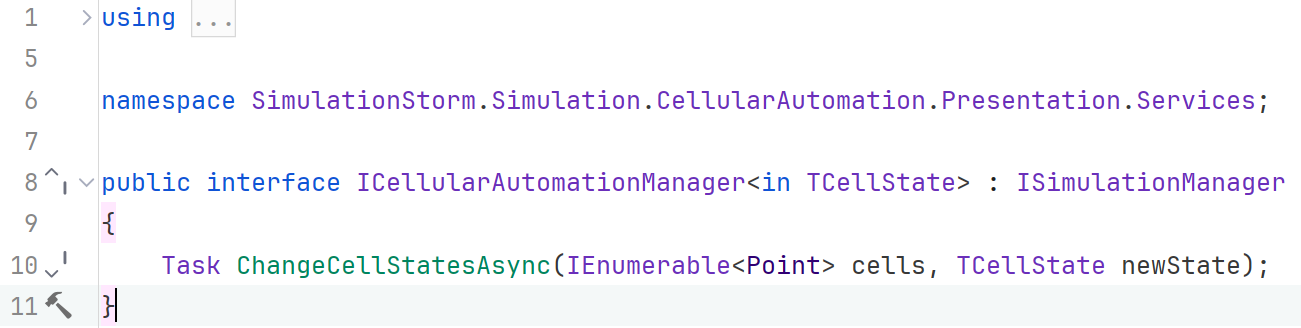


Рисунок 4.15 Модуль для клеточных автоматов

Данный интерфейс предоставляет единственный метод ChangeCellStates, который для переданного набора координат клеток изменяет состояние на переданное.

## 4.2 Разработка лаунчера симуляций

Для возможности запуска разных симуляций из единого приложения разработан лаунчер симуляций (Рисунок 4.16).

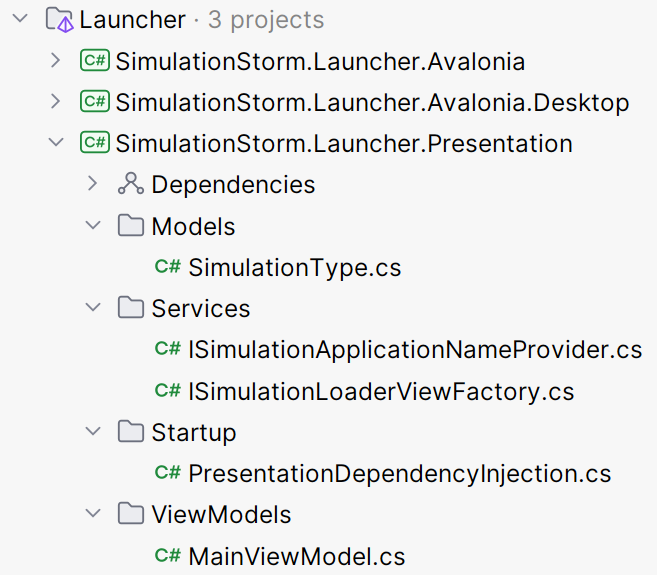


Рисунок 4.16 Структура проекта лаунчера симуляций

Основным компонентом лаунчера является модель представления, которая позволяет изменять некоторые настройки интерфейса, выбирать желаемую для запуска симуляцию и запускать выбранную симуляцию (Рисунок 4.17)

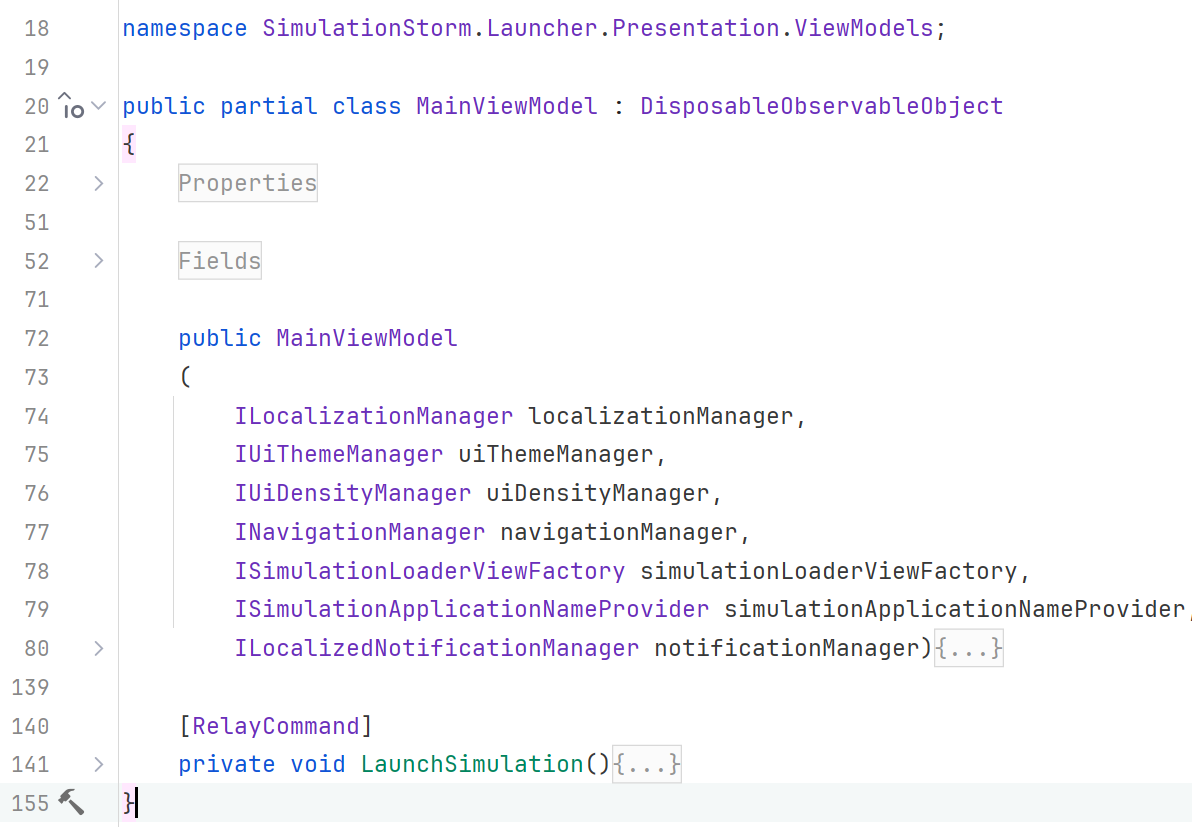


Рисунок 4.17 Модель представления лаунчера

После запуска выбранной симуляции, представление лаунчера сменяется на представление загрузчика симуляции.

Для того, чтобы иметь возможность запускать различные симуляции из лаунчера, используется сервис, который для выбранного типа симуляции конструирует загрузчик представления этой симуляции (Рисунок 4.18).

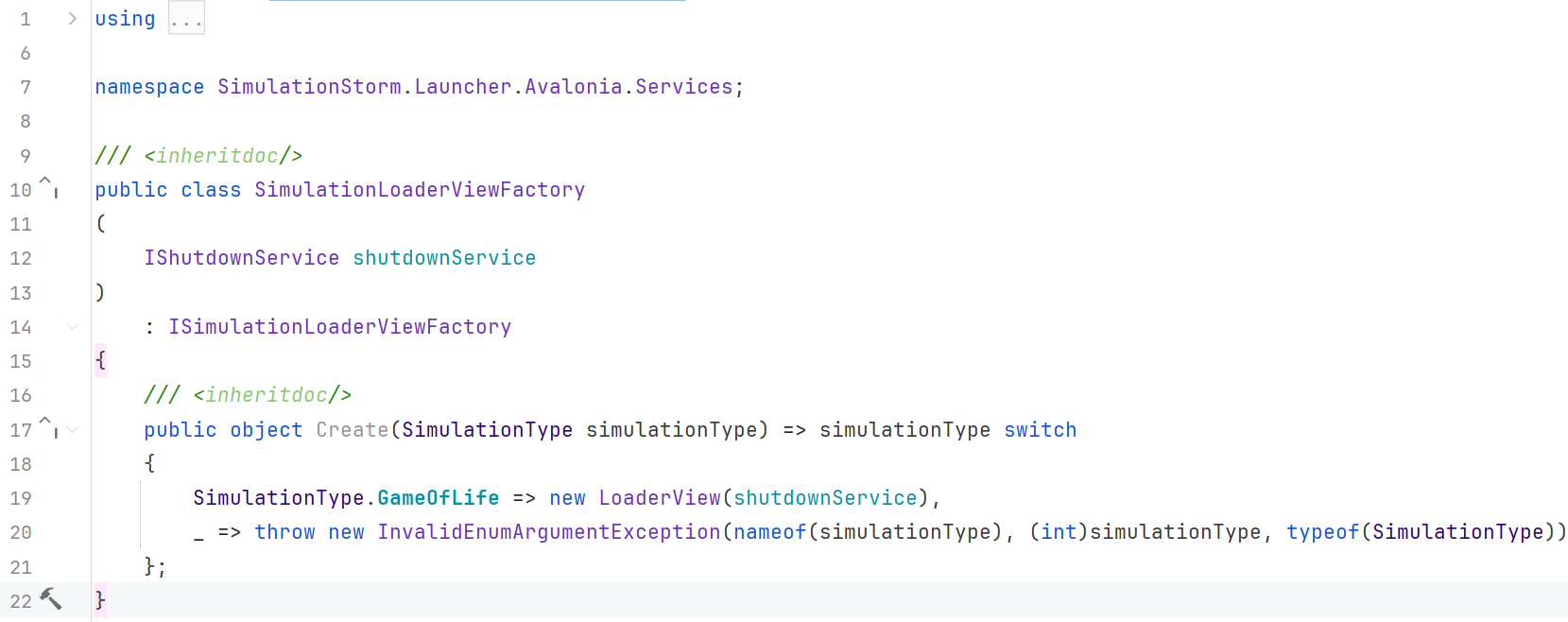


Рисунок 4.18 Сервис создания загрузчиков симуляций

Точкой входа в каждое приложение симуляции является представление LoaderView, которое отображает экран загрузки симуляции и в фоновом потоке выполняет инициализацию контейнера всех сервисов, которые участвуют в работе симуляции. Такой механизм загрузки необходим для обеспечения положительного пользовательского опыта, так как некоторые симуляции могут требовать длительного времени для инициализации всех сервисов и без подобного механизма отображения загрузки, будет казаться, что приложение не запустилось, когда на самом деле приложение запустилось, но ещё не отобразилось.

Дополнительной особенностью симуляций, разработанных с использованием инструментария, является гарантия того, что одновременно может быть запущен только один экземпляр приложения симуляции. Такой механизм обеспечивается с помощью сервиса ApplicationProcessManager.

# 5 Тестирование программного продукта

## 5.1 Общие понятия тестирования

Тестирование программного обеспечения (Software Testing) — это процесс исследования, проверки и оценки программного продукта с целью обнаружения ошибок, улучшения качества, обеспечения правильного функционирования и соответствия требованиям заказчика.

Процесс тестирования программного обеспечения включает в себя следующие этапы:

* определение целей тестирования, разработка стратегии и плана тестирования, выделение ресурсов;
* создание конкретных сценариев проверки функциональности, производительности, безопасности и других аспектов программы;
* запуск тестовых случаев, регистрация результатов, фиксация найденных ошибок;
* организация процесса управления дефектами, их отслеживание, повторное тестирование после устранения ошибок.

Основная цель тестирования ПО – обнаружение дефектов до выпуска продукта в эксплуатацию.

В данном проекте использовано два вида тестирования: тестовые сценарии и модульные тесты.

## 5.2 Создание и реализация модульных тестов

Модульное тестирование является существенным методом проверки программного обеспечения.

Основная суть данного метода тестирования заключается в том, что проверяются отдельные части кода, такие как функции, методы и классы, в изоляции от остального приложения. Это отличает модульное тестирование от функционального тестирования, которое проверяет работу приложения в целом.

Целью модульного тестирования является обеспечение правильной работы каждого модуля системы.

Преимуществами модульного тестирования являются:

* модульные тесты помогают выявить дефекты и ошибки на ранних этапах разработки, что способствует повышению надежности программного продукта;
* наличие хорошо написанных модульных тестов упрощает процесс изменения кода и рефакторинга;
* модульные тесты могут служить в качестве документации к коду, позволяя быстрее понять, какие функции должны выполнять отдельные компоненты программы;
* раннее выявление дефектов с помощью позволяет сэкономить время и ресурсы на последующих этапах разработки.

Благодаря юнит-тестам улучшается общее качество кода, ускоряется процесс разработки и облегчается его поддержка в будущем. Помимо этого, наличие юнит-тестов повышает уверенность в стабильности и исправности программы, снижая риск возникновения ошибок в процессе эксплуатации.

Для тестирования механизма работы добавления сервисов в коллекции был создан соответствующий Unit-текст (Рисунок 5.1).

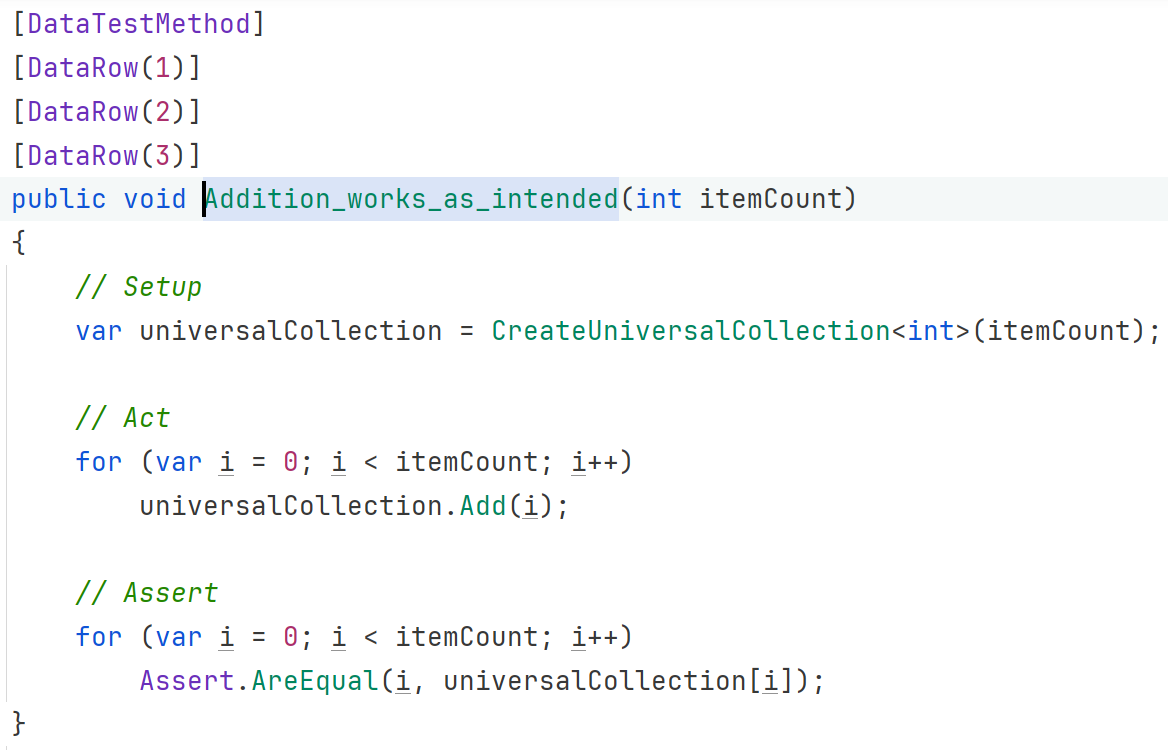


Рисунок 5.1 Тестирование работы механизма добавления элементов в коллекцию

Тестирование компонентов модуля, реализующих работу с коллекциями, является важным, так как коллекции повсеместно используются в проекте.

Для регулировки и корректной работы сервисов требуется отслеживать счетчик, для проверки данного функционала был создан соответствующий модульный тест (Рисунок 5.2).

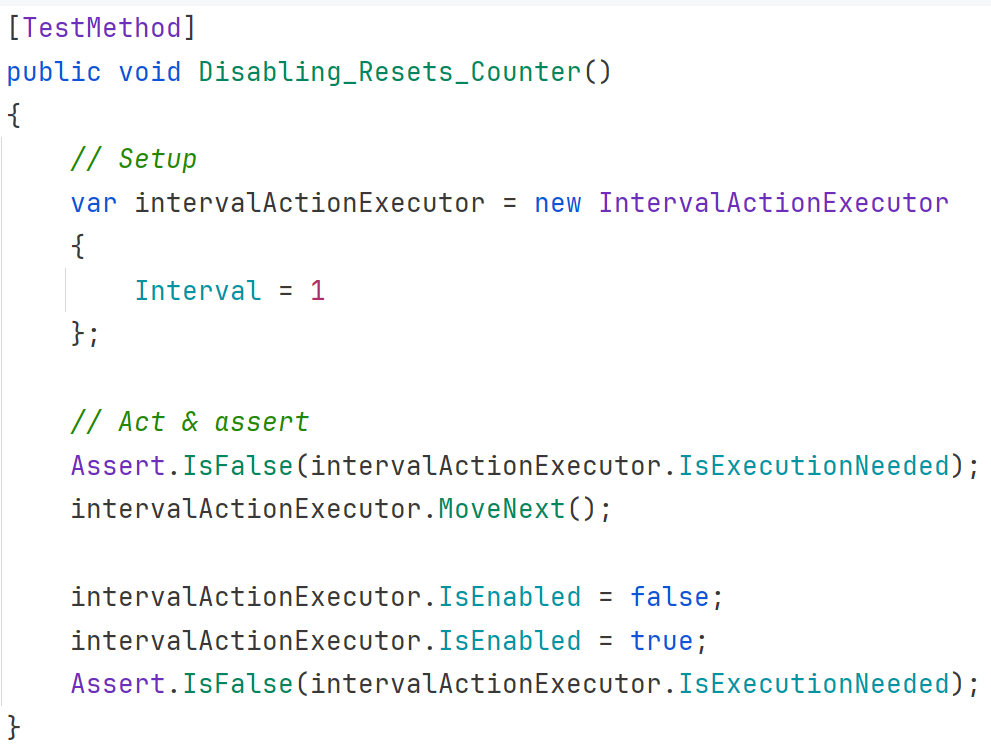


Рисунок 5.2 Тестирование сброса счетчика действий

Таким образом Unit-тесты обеспечивают надежность кода, ускоряeт процесс разработки, облегчают поддержку и доработку инструментария в будущем.

## 5.3 Тест-кейсы

Для более глубокого понимания корректности реализованного инструментария были применены тест-кейсы.

Тест-кейс — это набор шагов и ожидаемых результатов, который позволяет проверить определенный функциональный аспект или сценарий использования инструментария.

Как правило, один тест-кейс описывает небольшую последовательность действий с одним конкретным результатом. Например, успешную авторизацию на сайте для конкретного пользователя или добавление одного конкретного товара в корзину.

Благодаря тест-кейсам специалисты всегда знают, как и что протестировать оптимальным количеством проверок, и не забывают о нюансах, так как записан каждый шаг.

Тестовый сценарий № 1 проверяет работоспособность функции увеличения и уменьшения масштаба камеры поля (Таблица 5.1).

Таблица 5.1 – Тест кейс № 1

|  |  |
| --- | --- |
| Тест-кейс № | 1 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции изменения масштаба камеры |
| Резюме испытания | Проверка того, что при изменении масштаба камеры в панели настроек камеры, используя мышь или горячие клавиши, поле приближается или отдаляется в соответствии с масштабом |
| Шаги тестирования | 1. открыть панель камеры;  2. изменить масштаб камеры;  3. изменить масштаб камеры с помощью мыши;  4. изменить масштаб камеры с помощью горячих клавиш. |
| Данные тестирования | 1. масштаб – 1000%;  2. масштаб – 50%;  3. масштаб – 10%;  4. масштаб – 5000%;  5. масштаб – 10 000%. |
| Ожидаемый результат | Поле должно приблизиться или отдалиться, в зависимости от значения масштаба. |
| Фактический результат | Указанный масштаб был применён к полю |
| Предпосылки | Масштаб камеры равен ста процентам |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Масштабирование камеры — это функция, которая даёт возможность увеличить или уменьшить изображение поля симуляции.

Путем изменения масштаба изображения пользователь может увеличить детали и углубиться в изучение конкретных участков, что часто бывает полезно при анализе данных или планировании действий.

Тестовый сценарий № 2 проверяет работоспособность механизма сохранения всего приложения (Таблица 5.2).

Таблица 5.2 – Тест кейс № 2

|  |  |
| --- | --- |
| Тест-кейс № | 2 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции сохранения приложения |
| Резюме испытания | Проверка того, что при загрузке сохранения рабочее окружение будет обновлено в соответствии с данными их сохранения |
| Шаги тестирования | 1. выполнить какое-либо действие над симуляцией или изменить настройку интерфейса;  2. выполнить сохранение приложения;  3. перезапустить приложение;  4. загрузить предыдущее сохранение. |
| Данные тестирования | 1. изменить размер поля симуляции;  2. изменить тему интерфейса на противоположную;  3. изменить масштаб камеры;  4. изменить интервал отрисовки симуляции;  5. отключить отрисовку линий сетки;  6. включить сбор статистики. |
| Ожидаемый результат | При загрузке сохранения рабочее окружение должно быть восстановлено из сохранения |
| Фактический результат | Рабочее окружение восстанавливается |
| Предпосылки | – |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Функция сохранения всего приложения предоставляет возможность сохранения текущего рабочего окружения приложения. Это включает в себя сохранение всех данных, настроек, состояния пользовательского интерфейса и другой информации, которая позволяет пользователю работать эффективнее в рамках приложения.

Тестовый сценарий № 3 служит для проверки работоспособности функции сбора статистики симуляции (Таблица 5.3).

Таблица 5.3 – Тест кейс № 3

|  |  |
| --- | --- |
| Тест-кейс № | 3 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка функции сбора статистики симуляции |
| Резюме испытания | Проверка того, что при выполнении действия над симуляцией создаётся новая запись, отображаемая в панели статистики симуляции |
| Шаги тестирования | 1. открыть панель статистики симуляции;  2. включить сбор статистики;  3. выполнить действие над симуляцией. |
| Данные тестирования | 1. команда «выполнить шаг симуляции»;  2. команда «сбросить состояние симуляции»;  3. команда «продвинуть симуляцию»;  4. команда «изменить размер мира симуляции». |
| Ожидаемый результат | Появление записи в панели статистике |
| Фактический результат | При выполнении действия над симуляцией, в панели статистики появляется запись |
| Предпосылки | Сбор статистки симуляции должен быть отключён. |
| Постусловия | – |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Данная функция предоставляет возможность быстрого анализа оборота данных в симуляции, что означает, что пользователь может легко и быстро оценить, как данные перемещаются или изменяются в процессе симуляции. Б Благодаря этой функции, пользователь может более эффективно изучать процессы, протекающие в симуляции, и принимать информированные решения на основе полученных данных.

Собранные статистические данные позволяют пользователям проводить детальный анализ каждого шага в симуляции, выявлять узкие места, оптимизировать процессы и принимать верные решения на основе фактов.

Тестовый сценарий № 4 проверяет то, что одна и та же симуляция может быть запущена одновременно только в одном экземпляре (Таблица 5.4).

Таблица 5.4 – Тест кейс 4

|  |  |
| --- | --- |
| Тест-кейс № | 4 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка невозможности запуска симуляции в двух экземплярах |
| Резюме испытания | Проверка того, что одна и та же симуляция не может быть одновременно запущена в двух экземплярах для обеспечения отсутствия конфликтов доступа к данным |
| Шаги тестирования | 1. запустить исполняемый файл симуляции и дождаться запуска;  2. снова запустить исполняемый файл симуляции. |
| Данные тестирования | – |
| Ожидаемый результат | При повторном запуске исполняемого файла симуляции должно появиться сообщение о том, что данная симуляция уже запущена |
| Фактический результат | Появляется сообщение о том, что симуляция уже запущена |
| Предпосылки | Симуляция не должна быть запущена |
| Постусловия | Должен остаться только один запущенный экземпляр симуляции |
| Статус (Pass/Fail) | Pass |
| Комментарии | – |

Такое поведение, позволяет обеспечить надежность и целостность базы данных, с которой взаимодействует симуляция. Это означает, что любые изменения, происходящие в данных в процессе работы с симуляцией, будут корректно обрабатываться и сохранять целостность всей базы. Благодаря этой особенности, пользователи могут быть уверены в том, что информация, с которой работает симуляция, остается надежной и актуальной, и любые операции с данными проводятся без потерь и искажений.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Поскольку данный проект представляет собой инструментарий (или библиотеку), то руководство пользователя предназначается для разработчиков.

В руководстве будет подробно описан процесс разработки симуляции, используя предоставленный инструментарий, на примере демонстрационной симуляции «Игра Жизнь».

Все симуляции, разрабатываемые с использованием инструментария, имеют следующий состав:

* проект доменной модели (или предметной области) симуляции, который должен содержать в себе программный код непосредственно только симуляции, без ссылок на сторонние библиотеки, в роде библиотек графического и пользовательского интерфейса;
* проект уровня представления, который содержит сервисы, модели и модели представлений, соответствующие паттерну MVVM, и является как бы адаптером между доменной моделью симуляции и пользовательским интерфейсом. Ссылок на конкретные фреймворки пользовательского интерфейса данный проект также не содержит, но может содержать ссылки на библиотеки для работы с MVVM;
* проект уровня пользовательского интерфейса, который использует один из фреймворков пользовательского интерфейса. На данный момент, стандартным фреймворком ПИ в инструментарии является AvaloniaUI, при этом это может быть любой другой фреймворк. Гибкость выбора фреймворка может повлиять на итоговое качество и удобство использования интерфейса.

Следование такой структуре обеспечивает четкое разделение ответственностей, повышает гибкость и удобство использования инструментария при создании симуляций.

## 6.1 Подготовительный этап

Для начала работы, необходимо иметь среду разработки, поддерживающую язык C#. Это JetBrains Rider или Visual Studio.

Первоначально следует клонировать репозиторий инструментария (Рисунок 6.1).

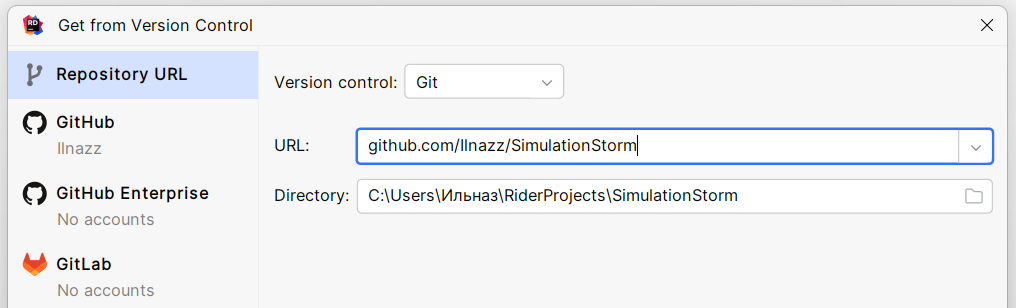


Рисунок 6.1 Клонирование репозитория с инструментарием

После клонирования репозитория имеется возможность работать с проектом в офлайн режиме, не имея постоянного доступа к интернету, также клонирование репозитория позволяет использовать всю мощь системы контроля версий локально, работая над проектом в собственной среде разработки.

Также важным преимуществом клонирования репозитория является возможность безопасно вносить изменения и экспериментировать с кодом, не боясь навредить основной ветке проекта.

После клонирования, необходимо создать новую папку решения, назвав её по имени симуляции (Рисунок 6.2).

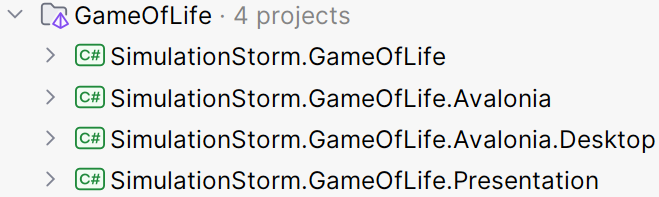


Рисунок 6.2 создание папки решения для новой симуляции

Данное действие необходимо для создания окружения проекта и дальнейшей работы по созданию симуляции.

После создания окружения проекта, следующий шаг - работа над созданием симуляции. Это процесс, включающий в себя моделирование поведения системы или процесса.

## 6.2 Разработка доменной модели симуляции

После подготовки окружения, можно начать разработку доменной модели симуляции. В случае, если программный код симуляции уже разработан, то этот код может быть легко адаптирован к использованию с данным инструментарием.

В проекте симуляции необходимо иметь один основной класс или интерфейс, который выступает в роли фасада симуляции. Данный класс или интерфейс должен централизованно управлять всеми аспектами симуляции, обеспечивая удобный и единый доступ к функциональности моделирования процессов (Рисунок 6.3).

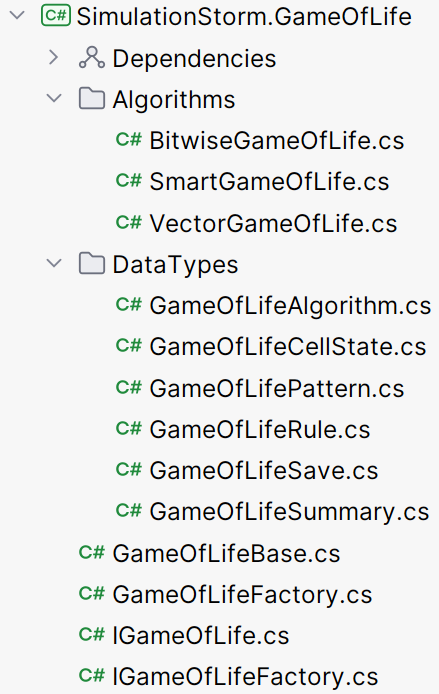


Рисунок 6.3 Структура симуляции

Реализация фасада симуляции позволяет упростить взаимодействие с комплексной системой симуляции, а также повысить модульность и расширяемость проекта.

В данном случае фасад симуляции реализует интерфейсы из модулей проведения, сброса, статистики, истории и модуля для клеточных автоматов.

Фасад симуляции реализует интерфейсы из различных модулей, например, для проведения симуляции, учета статистики, ведения истории событий и работы с клеточными автоматами.

## 6.3 Разработка проекта уровня представления.

После того, как был разработан с нуля или адаптирован существующий проект доменной модели симуляции, можно начать работу над проектом уровня представления (Рисунок 6.4).

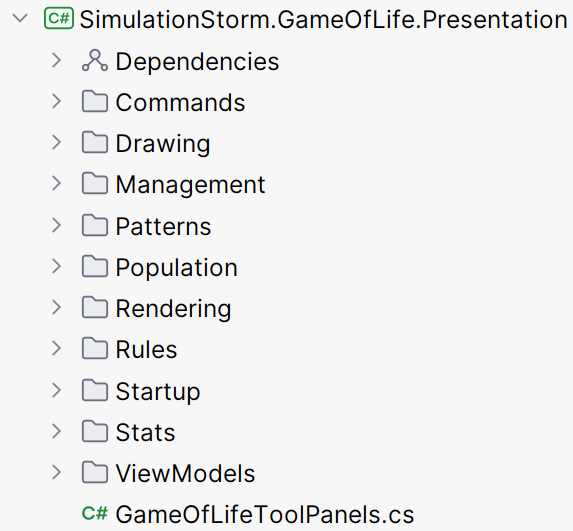


Рисунок 6.4 Структура представления симуляции

Один из основных сервисов, который необходимо реализовать в этом проекте, это менеджер симуляции (Рисунок 6.5).

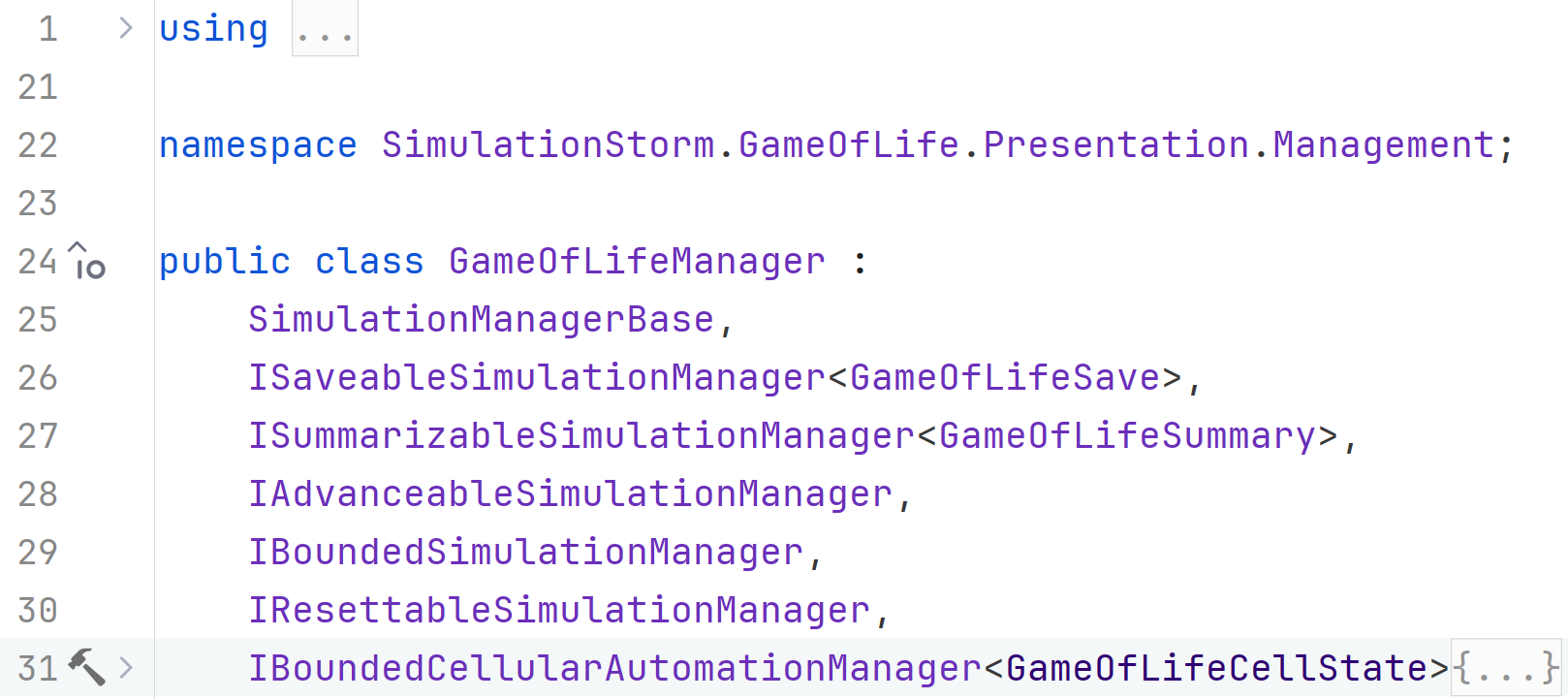


Рисунок 6.5 Менеджер симуляции

Менеджер обеспечивает правильное взаимодействие симуляции с другими сервисами.

Менеджер в проекте выполняет роль реализации интерфейсов из выбранных модулей на уровне представления, предоставляя удобный доступ к функциональности симуляции через соответствующие интерфейсы.

Следующий важный сервис — это сервис отрисовки симуляции. Данный сервис может быть унаследован от базовой реализации сервиса отрисовки, или создан с нуля (Рисунок 6.6).

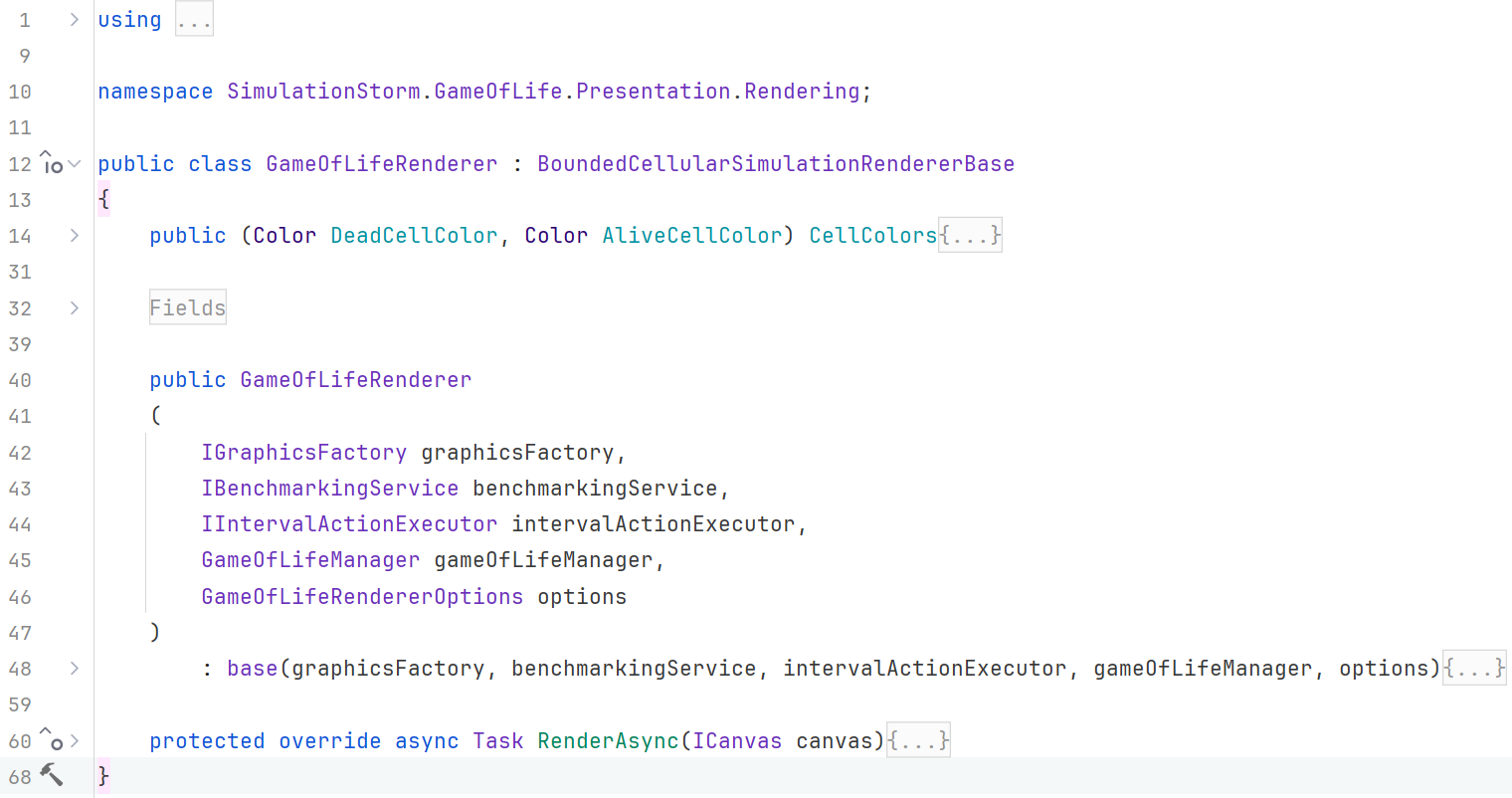


Рисунок 6.6 Сервис отрисовки симуляции

Наследование от базовой реализации сервиса отрисовки позволяет упростить процесс разработки других модулей симуляции.

Далее, необходимо создать компоненты, которые будут участвовать в процессе конфигурации приложения во время запуска.

Созданные данных компонентов на ранних стадиях реализации проекта, помогает более корректно организовать структуру проекта.

Первый компонент, называемый PresentationConfiguration, должен содержать объекты опций (Рисунок 6.7).



Рисунок 6.7 Компонент, содержащий объекты опций

PresentationConfiguration - это специальный класс, который предназначен для хранения настроек различных сервисов, используемых в приложении.

Создание PresentationConfiguration позволяет легко управлять и настраивать сервисы приложения, централизуя все необходимые параметры в одном месте.

Второй компонент, PresentationDependencyInjection, служит для проведения регистрации всех необходимых сервисов уровня представления в контейнер инъекции зависимостей (Рисунок 6.8).



Рисунок 6.8 Компонент регистраций сервисов уровня представления

В случае, если симуляция использует модуль статистики, то необходимо создать модели представлений для желаемых типов диаграмм (Рисунок 6.9).

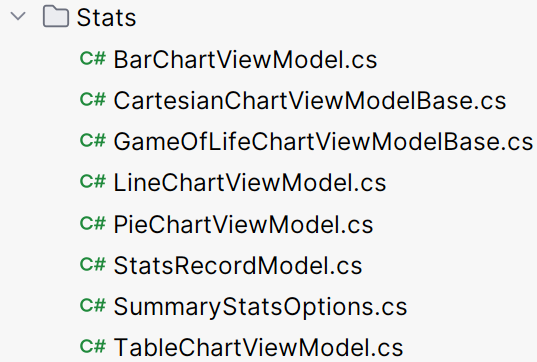


Рисунок 6.9 Директория, содержащая желаемые типы диаграмм

Модели представлений диаграмм предоставляют данные для отображения представлениями.

В случае симуляции «Игра Жизнь», это табличная, круговая, столбчатая и линейная диаграммы.

## 6.4 Разработка проекта уровня пользовательского интерфейса.

Данный проект должен содержать представления, компоненты конфигурации и, опционально, ресурсы локализации (переводы строк на разные языки) (Рисунок 6.10).

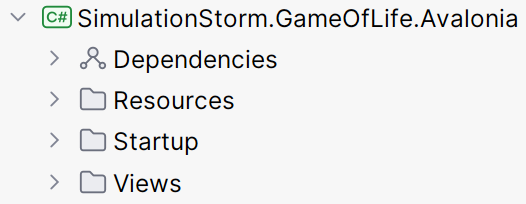


Рисунок 6.10 Ресурсы, компоненты конфигурации и ресурсы локализации

Компоненты конфигурации в проекте данного уровня включают в себя аналоги из проекта уровня представления (Configuration и DependencyInjection), а также компонент-загрузчик (LoaderView), ответственный за инициализацию зарегистрированных в контейнере инъекции зависимостей сервисов и загрузку основного дерева представлений (Рисунок 6.11).

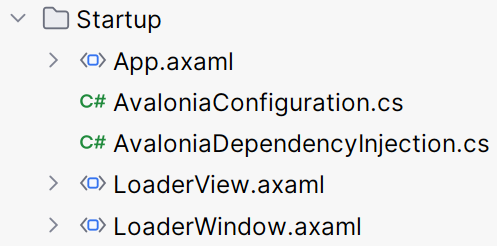


Рисунок 6.11 Компоненты конфигурации

Ресурсы локализации представляют собой словари ресурсов и содержат переводы ключевых строк на разные языки (Рисунок 6.12).

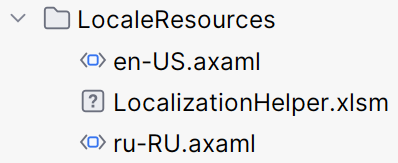


Рисунок 6.12 Ресурсы локализации

В данном случае, это русский и английский языки.

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта был разработан инструментарий для создания симуляций с применением платформы .NET, фреймворка Avalonia, языка C# и дополнительных библиотек.

Данный инструментарий был разработан с несколькими основными целями:

* инструментарий предоставляет удобные и эффективные средства для создания различных симуляций, что позволяет разработчикам быстро и качественно моделировать различные процессы и необходимые модули;
* используя этот набор инструментов, разработчики могут взаимодействовать друг с другом, обмениваться опытом, делиться знаниями и лучшими практиками. Это способствует росту сообщества разработчиков и улучшает коллективный опыт;
* благодаря удобству использования инструментария, его функциональности и поддержке сообщества, этот набор инструментов способствует привлечению новых пользователей и разработчиков, которые заинтересованы в развитии или использовании симуляций в своей деятельности;
* инструментарий предоставляет функционал для анализа собранных данных из симуляций;
* набор инструментов разработан с учетом возможности масштабирования симуляций и обработки данных в больших объемах, обеспечивая при этом высокую производительность и эффективность работы;
* данный набор инструментов также ориентирован на помощь исследователям и образовательным учреждениям в проведении исследований, создании образовательных материалов и обучении студентов с помощью симуляций и анализа данных.

Имея возможность выбора только необходимых модулей, разработчики могут оптимизировать процесс создания проектов, сокращая издержки и уменьшая сложность кодовой базы. Модульность инструментария позволяет легко подключать и отключать функции по мере необходимости, обеспечивая гибкость и простоту в управлении проектами.

Благодаря встроенным механизмам расширения и возможности создания новых модулей, разработчики могут адаптировать инструментарий под конкретные требования любого проекта, что повышает его масштабируемость и способствует быстрой интеграции с другими системами.

Благодаря готовому набору библиотек и модулей, доступных в инструментарии, разработчики могут сосредоточиться на уникальных особенностях каждой конкретной симуляции, повышая тем самым качество и инновационность своих проектов. Это также способствует повышению производительности и эффективности процесса разработки, поскольку разработчики могут быстро переходить к реализации и тестированию новых идей, минуя стандартные этапы разработки базовой функциональности.

Благодаря возможности легкого масштабирования и переиспользования готовых модулей в различных проектах, инструментарий не только ускоряет разработку, но и обеспечивает стабильность и надежность процесса создания симуляций. Это позволяет командам разработчиков быстрее реагировать на изменения в требованиях заказчиков или рыночных условиях, обеспечивая гибкость и адаптивность в разработке программных проектов.

Таким образом, разработанный инструментарий предоставляет программистам возможность быстро разрабатывать разнообразные симуляции, не тратя время на разработку типовых модулей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Клеточные автоматы [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Клеточный\_автомат, свободный.
2. Руководство работы с C# [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/, свободный.
3. Инструкция по работе с платформой Avalonia [Электронный ресурс]. Режим доступа: https://docs.avaloniaui.net/docs/welcome, свободный.
4. Изучение Rider [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/ru-ru/academy/#learn, свободный.
5. Общие сведения о набор средств MVVM [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/communitytoolkit/mvvm/, свободный.
6. Реактивные расширения [Электронный ресурс]. Режим доступа: https://habr.com/ru/companies/nix/articles/261031/, свободный.
7. Инструкция по работе с шаблоном проектирования MVVM [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/338518/, свободный.
8. 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.
9. Разработка на SQLite [Электронный ресурс]. Режим доступа: https://sqlite.org/, свободный.
10. Возможности .Net [Электронный ресурс]. Режим доступа: https://checkroi.ru/blog/, свободный.
11. Справочник по SQL [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/SQL, свободный.
12. Инструкция по работе со сценариями использования [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/699522/, свободный.
13. GitHub [Электронный ресурс]. Режим доступа: https://github.com/, свободный.
14. Проектирование веб-сайта [Электронный ресурс]. Режим доступа: https://wp.z-go.ru/, свободный.
15. Программное-обеспечение [Электронный ресурс]. Режим доступа: https://creately.com/ru/lp, свободный.
16. Учебное пособие [Электронный ресурс]. Режим доступа: https://ru.stackoverflow.com/questions/613336, свободный.
17. Программное-обеспечение [Электронный ресурс]. Режим доступа: https://qna.habr.com/q/292075, свободный.
18. Entity Framework Core [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/ef/core/, свободный.
19. Use case [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/699522/, свободный.
20. ASP.NET Core [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/, свободный.
21. Транзакции в SQL Server [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/transactions-transact-sql/, свободный.
22. Microsoft SQL Server Management Studio (SSMS) [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/sql/ssms/, свободный.
23. Типы данных в SQL [Электронный ресурс]. Режим доступа: https://metanit.com/sql/sqlite/2.2.php, свободный.
24. 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.

# ПРИЛОЖЕНИЕ А

Техническое задание

Целью данного проекта является упрощение и ускорение процесса разработки различных компьютерных симуляций, путём предоставления системы библиотек и модулей, реализующих типовые аспекты симуляций.

Для достижения поставленной цели необходимо решить следующие задачи: провести анализ предметной области симуляций, выделить общие для разнообразных симуляций свойства и характеристики, спроектировать модули, которые будет предоставлять реализацию определённых свойств и характеристик симуляций, разработать и протестировать модули.

Проект должен соответствовать следующим требованиям: поддерживать настольные ОС Windows и Linux, иметь удобный, интуитивно понятный пользовательский интерфейс, предоставлять тёмную и светлую темы интерфейса, предоставлять варианты интерфейса на русском и английском языках с возможностью динамического изменения языка во время работы приложения, поддерживать пользовательский интерфейс отзывчивым, обеспечивать надёжность сохраняемых данных, быть отказоустойчивым, а структура проекта должна способствовать лёгкости дальнейшего расширения функционала.

Для реализации проекта должны быть использованы следующие технологии: платформа .NET 8+, являющаяся кроссплатформенной и достаточно производительной, язык программирования C#, обладающий богатой экосистемой готовых библиотек и фреймворков, фреймворк пользовательского интерфейса Avalonia UI, позволяющий создавать интерфейсы любой сложности, поддерживающий языки разметки и стилизации интерфейса по аналогии с HTML и CSS и поддерживающий паттерн проектирования MVVM. Такой стек обеспечивает возможность разрабатывать весь программный код проекта (модули бизнес-логики; модули, соединяющие пользовательский интерфейс и бизнес-логику; пользовательский интерфейс) на одной платформе и на одном языке.

# ПРИЛОЖЕНИЕ Б

Интерфейс менеджера симуляции в файле ISimulationManager.cs

using System;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

namespace SimulationStorm.Simulation.Presentation.SimulationManager;

public interface ISimulationManager

{

ReadOnlyObservableCollection<SimulationCommand> CommandQueue { get; }

/// <summary>

/// Occurs before the start of the command execution.

/// </summary>

event EventHandler<SimulationCommandExecutingEventArgs>? CommandExecuting;

/// <summary>

/// Occurs after the command has been executed.

/// </summary>

event EventHandler<SimulationCommandExecutedEventArgs>? CommandExecuted;

Task ClearCommandQueueAsync();

// Task WaitForAllQueuedCommandsExecutingAsync(); // to implement this, we can use Task.WhenAll(queuedCommands.Tasks)

Базовый класс менеджера симуляции в файле SimulationManagerBase.cs

using System;

using System.Collections.ObjectModel;

using System.Threading;

using System.Threading.Channels;

using System.Threading.Tasks;

using DotNext.Threading;

using SimulationStorm.Utilities;

using SimulationStorm.Utilities.Benchmarking;

namespace SimulationStorm.Simulation.Presentation.SimulationManager;

public abstract class SimulationManagerBase : ISimulationManager, IAsyncDisposable

{

public ReadOnlyObservableCollection<SimulationCommand> CommandQueue { get; }

#region Events

public event EventHandler<SimulationCommandExecutingEventArgs>? CommandExecuting;

public event EventHandler<SimulationCommandExecutedEventArgs>? CommandExecuted;

#endregion

#region Fields

private readonly AsyncReaderWriterLock \_readerWriterLock = new();

private readonly IBenchmarkingService \_benchmarkingService;

private readonly ISimulationManagerOptions \_options;

private Channel<QueuedSimulationCommand> \_commandChannel;

private CancellationTokenSource \_commandProcessingCycleCancellationSource = new();

private Task \_commandProcessingCycleTask;

private readonly ObservableCollection<SimulationCommand> \_commandQueue = [];

#endregion

protected SimulationManagerBase

(

IBenchmarkingService benchmarkingService,

ISimulationManagerOptions options)

{

\_benchmarkingService = benchmarkingService;

\_options = options;

\_commandChannel = Channel.CreateUnbounded<QueuedSimulationCommand>(new UnboundedChannelOptions

{

SingleReader = true

});

CommandQueue = new ReadOnlyObservableCollection<SimulationCommand>(\_commandQueue);

\_commandProcessingCycleTask = ProcessCommandsInCycleAsync(\_commandProcessingCycleCancellationSource.Token)

.ThrowWhenFaulted();

}

private readonly AsyncReaderWriterLock \_commandChannelRwLock = new();

public async Task ClearCommandQueueAsync()

{

await \_commandChannelRwLock.EnterWriteLockAsync();

\_commandChannel.Writer.Complete();

await \_commandProcessingCycleCancellationSource.CancelAsync();

await \_commandProcessingCycleTask;

\_commandProcessingCycleCancellationSource.Dispose();

\_commandChannel = Channel.CreateUnbounded<QueuedSimulationCommand>(new UnboundedChannelOptions

{

SingleReader = true

});

\_commandProcessingCycleCancellationSource = new CancellationTokenSource();

\_commandProcessingCycleTask = ProcessCommandsInCycleAsync(\_commandProcessingCycleCancellationSource.Token)

.ThrowWhenFaulted();

\_commandChannelRwLock.Release();

}

public async ValueTask DisposeAsync()

{

\_commandChannel.Writer.Complete();

await \_commandProcessingCycleCancellationSource.CancelAsync();

await \_commandProcessingCycleTask;

\_commandProcessingCycleCancellationSource.Dispose();

await \_readerWriterLock.DisposeAsync();

GC.SuppressFinalize(this);

}

#region Protected methods

protected abstract void ExecuteCommand(SimulationCommand command);

protected Task<TimeSpan> MeasureWithReadLockAsync(Action action) =>

WithReadLockAsync(() => Measure(action));

protected Task<BenchmarkResult<T>> MeasureWithReadLockAsync<T>(Func<T> function) =>

WithReadLockAsync(() => \_benchmarkingService.Measure(function));

protected async Task WithReadLockAsync(Action action)

{

await \_readerWriterLock.EnterReadLockAsync();

action();

\_readerWriterLock.Release();

}

protected async Task<T> WithReadLockAsync<T>(Func<T> function)

{

await \_readerWriterLock.EnterReadLockAsync();

var functionResult = function();

\_readerWriterLock.Release();

return functionResult;

}

protected TimeSpan Measure(Action action) => \_benchmarkingService.Measure(action).ElapsedTime;

protected BenchmarkResult<T> Measure<T>(Func<T> function) => \_benchmarkingService.Measure(function);

protected async Task QueueCommandExecutionAsync(SimulationCommand command)

{

await \_commandChannelRwLock.EnterReadLockAsync();

var queuedCommand = new QueuedSimulationCommand(command);

\_commandQueue.Add(command);

await \_commandChannel.Writer.WriteAsync(queuedCommand);

\_commandChannelRwLock.Release();

await queuedCommand.Task;

}

#endregion

private async Task ProcessCommandsInCycleAsync(CancellationToken cancellationToken)

{

await foreach (var queuedCommand in \_commandChannel.Reader.ReadAllAsync(cancellationToken))

{

// Try this instead of passing token to enter write lock async, because it causes exception...

if (cancellationToken.IsCancellationRequested)

break;

await \_readerWriterLock.EnterWriteLockAsync();

NotifyCommandExecuting(queuedCommand.Command);

var elapsedTime = Measure(() => ExecuteCommand(queuedCommand.Command));

\_readerWriterLock.Release();

\_commandQueue.RemoveAt(0);

queuedCommand.NotifyExecuted();

await using var eventSynchronizer = new AsyncCountdownEvent(\_options.CommandExecutedEventHandlerCount);

NotifyCommandExecuted(queuedCommand.Command, elapsedTime, eventSynchronizer);

// [NOTE] If cancellation token will be passed, then services

// could not signal the event synchronized when disposing.

await eventSynchronizer.WaitAsync();

}

}

#region Notifying

private void NotifyCommandExecuting(SimulationCommand command) =>

CommandExecuting?.Invoke(this, new SimulationCommandExecutingEventArgs(command));

private void NotifyCommandExecuted(SimulationCommand command, TimeSpan elapsedTime, IAsyncEvent synchronizer) =>

CommandExecuted?.Invoke(this, new SimulationCommandExecutedEventArgs(command, elapsedTime, synchronizer));

#endregion

}

Базовый класс сервиса отрисовки в файле RendererBase.cs

using System;

using System.Threading;

using System.Threading.Tasks;

using DotNext.Threading;

using SimulationStorm.Graphics;

using SimulationStorm.Primitives;

using SimulationStorm.Utilities;

using SimulationStorm.Utilities.Benchmarking;

using SimulationStorm.Utilities.Disposing;

namespace SimulationStorm.Simulation.Presentation.Renderer;

public abstract class RendererBase : AsyncDisposableObservableObject, IRenderer

{

public IBitmap? RenderedImage => \_bitmapCopy;

#region Events

public event EventHandler? RenderingStarted;

public event EventHandler<RenderingCompletedEventArgs>? RenderingCompleted;

#endregion

protected IGraphicsFactory GraphicsFactory { get; }

protected abstract Size SizeToRender { get; }

#region Fields

private readonly IBenchmarkingService \_benchmarkingService;

private readonly AsyncAutoResetEvent \_renderingCycleSynchronizer = new(false);

private readonly CancellationTokenSource \_renderingCycleCancellationSource = new();

private readonly Task \_renderingCycleTask;

private IBitmap? \_bitmap;

private IBitmap? \_bitmapCopy;

private ICanvas? \_bitmapCanvas;

#endregion

protected RendererBase(IGraphicsFactory graphicsFactory, IBenchmarkingService benchmarkingService)

{

GraphicsFactory = graphicsFactory;

\_benchmarkingService = benchmarkingService;

\_renderingCycleTask = RenderInCycleAsync(\_renderingCycleCancellationSource.Token)

.ThrowWhenFaulted();

}

public override async ValueTask DisposeAsync()

{

\_renderingCycleSynchronizer.Set();

await \_renderingCycleCancellationSource.CancelAsync();

await \_renderingCycleTask;

await \_renderingCycleSynchronizer.DisposeAsync();

\_renderingCycleCancellationSource.Dispose();

\_bitmap?.Dispose();

\_bitmapCanvas?.Dispose();

\_bitmapCopy?.Dispose();

await base.DisposeAsync();

GC.SuppressFinalize(this);

}

#region Rendering cycle

public void RequestRerender() => \_renderingCycleSynchronizer.Set();

private async Task RenderInCycleAsync(CancellationToken cancellationToken)

{

while (true)

{

await \_renderingCycleSynchronizer.WaitAsync(cancellationToken);

if (cancellationToken.IsCancellationRequested)

break;

await RenderAndNotifyRenderingCompletedAsync();

}

}

#endregion

#region Rendering methods

protected abstract Task RenderAsync(ICanvas canvas);

protected virtual async Task RenderAndNotifyRenderingCompletedAsync()

{

if (SizeToRender.Width is 0 || SizeToRender.Height is 0)

return;

NotifyRenderingStarted();

var benchmarkResult = await \_benchmarkingService.MeasureAsync(RenderCoreAsync);

NotifyRenderingCompleted(benchmarkResult.ElapsedTime);

}

protected async Task RenderCoreAsync()

{

RecreateBitmapAndCanvasIfSizeChanged();

await RenderAsync(\_bitmapCanvas!);

\_bitmapCanvas!.Flush();

\_bitmapCopy?.Dispose();

\_bitmapCopy = \_bitmap!.Copy();

}

#endregion

#region Notifying

private void NotifyRenderingStarted() =>

RenderingStarted?.Invoke(this, EventArgs.Empty);

protected virtual void NotifyRenderingCompleted(TimeSpan elapsedTime) =>

RenderingCompleted?.Invoke(this, new RenderingCompletedEventArgs(elapsedTime));

#endregion

#region Bitmap and canvas creation

private void RecreateBitmapAndCanvasIfSizeChanged()

{

if (\_bitmap?.Size == SizeToRender)

return;

\_bitmap?.Dispose();

\_bitmapCanvas?.Dispose();

CreateBitmapAndCanvas();

}

private void CreateBitmapAndCanvas()

{

\_bitmap = GraphicsFactory.CreateBitmap(SizeToRender);

\_bitmapCanvas = GraphicsFactory.CreateCanvas(\_bitmap);

}

#endregion

}

Базовый класс сервиса отрисовки симуляции в файле SimulationRendererBase.cs

using System;

using System.Collections.Concurrent;

using System.Reactive.Disposables;

using System.Reactive.Linq;

using System.Threading.Tasks;

using SimulationStorm.Graphics;

using SimulationStorm.Simulation.Presentation.Renderer;

using SimulationStorm.Simulation.Presentation.SimulationManager;

using SimulationStorm.Utilities;

using SimulationStorm.Utilities.Benchmarking;

namespace SimulationStorm.Simulation.Presentation.SimulationRenderer;

public abstract class SimulationRendererBase : RendererBase, ISimulationRenderer

{

#region Properties

public bool IsRenderingEnabled

{

get => \_intervalActionExecutor.IsEnabled;

set => SetProperty(\_intervalActionExecutor.IsEnabled, value, \_intervalActionExecutor, (o, v) => o.IsEnabled = v);

}

public int RenderingInterval

{

get => \_intervalActionExecutor.Interval;

set => SetProperty(\_intervalActionExecutor.Interval, value, \_intervalActionExecutor, (o, v) => o.Interval = v);

}

#endregion

public new event EventHandler<SimulationRenderingCompletedEventArgs>? RenderingCompleted;

#region Fields

private readonly IIntervalActionExecutor \_intervalActionExecutor;

private SimulationCommandExecutedEventArgs? \_simulationCommandExecutedEvent;

private readonly ConcurrentQueue<SimulationCommand> \_commandQueue = new();

#endregion

protected SimulationRendererBase

(

IGraphicsFactory graphicsFactory,

IBenchmarkingService benchmarkingService,

IIntervalActionExecutor intervalActionExecutor,

ISimulationManager simulationManager,

ISimulationRendererOptions options

)

: base(graphicsFactory, benchmarkingService)

{

\_intervalActionExecutor = intervalActionExecutor;

IsRenderingEnabled = options.IsRenderingEnabled;

RenderingInterval = options.RenderingInterval;

WithDisposables(disposables =>

{

Observable

.FromEventPattern<EventHandler<SimulationCommandExecutedEventArgs>, SimulationCommandExecutedEventArgs>

(

h => simulationManager.CommandExecuted += h,

h => simulationManager.CommandExecuted -= h

)

.Select(e => e.EventArgs)

.Subscribe(e =>

{

\_simulationCommandExecutedEvent = e;

if (!e.Command.ChangesWorld)

{

SignalSimulationCommandExecutedEventHandled();

return;

}

if (\_intervalActionExecutor.GetIsExecutionNeededAndMoveNext())

{

RememberCommand(e.Command);

RequestRerender();

}

else

SignalSimulationCommandExecutedEventHandled();

})

.DisposeWith(disposables);

});

}

protected override async Task RenderAndNotifyRenderingCompletedAsync()

{

await base.RenderAndNotifyRenderingCompletedAsync();

SignalSimulationCommandExecutedEventHandled();

}

protected void RememberCommand(SimulationCommand command) => \_commandQueue.Enqueue(command);

protected override void NotifyRenderingCompleted(TimeSpan elapsedTime)

{

var command = GetFirstRememberedCommand();

RenderingCompleted?.Invoke(this, new SimulationRenderingCompletedEventArgs(command, elapsedTime));

}

private SimulationCommand? GetFirstRememberedCommand()

{

\_commandQueue.TryDequeue(out var command);

return command;

}

private void SignalSimulationCommandExecutedEventHandled()

{

\_simulationCommandExecutedEvent?.Synchronizer.Signal();

\_simulationCommandExecutedEvent = null;

}

}

Класс модели представления настроек отрисовки симуляции в файле SimulationRenderingSettingsViewModel.cs

using System;

using System.Reactive.Disposables;

using System.Reactive.Linq;

using CommunityToolkit.Mvvm.Input;

using DynamicData.Binding;

using SimulationStorm.Primitives;

using SimulationStorm.Threading.Presentation;

using SimulationStorm.Utilities.Disposing;

namespace SimulationStorm.Simulation.Presentation.SimulationRenderer;

public partial class SimulationRenderingSettingsViewModel : DisposableObservableObject

{

#region Properties

public bool IsRenderingEnabled

{

get => \_simulationRenderer.IsRenderingEnabled;

set => \_simulationRenderer.IsRenderingEnabled = value;

}

public int RenderingInterval

{

get => \_simulationRenderer.RenderingInterval;

set => \_simulationRenderer.RenderingInterval = value;

}

public Range<int> RenderingIntervalRange => \_options.RenderingIntervalRange;

#endregion

[RelayCommand(CanExecute = nameof(CanResetRenderingInterval))]

private void ResetRenderingInterval() => RenderingInterval = \_options.RenderingInterval;

private bool CanResetRenderingInterval() => RenderingInterval != \_options.RenderingInterval;

#region Fields

private readonly ISimulationRenderer \_simulationRenderer;

private readonly ISimulationRendererOptions \_options;

#endregion

public SimulationRenderingSettingsViewModel

(

IUiThreadScheduler uiThreadScheduler,

ISimulationRenderer simulationRenderer,

ISimulationRendererOptions options)

{

\_simulationRenderer = simulationRenderer;

\_options = options;

WithDisposables(disposables =>

{

\_simulationRenderer

.WhenValueChanged(x => x.IsRenderingEnabled, false)

.ObserveOn(uiThreadScheduler)

.Subscribe(\_ => OnPropertyChanged(nameof(IsRenderingEnabled)))

.DisposeWith(disposables);

\_simulationRenderer

.WhenValueChanged(x => x.RenderingInterval, false)

.ObserveOn(uiThreadScheduler)

.Subscribe(\_ =>

{

OnPropertyChanged(nameof(RenderingInterval));

ResetRenderingIntervalCommand.NotifyCanExecuteChanged();

})

.DisposeWith(disposables);

});

}

}

Класс модели представления очереди команд в файле CommandQueueViewModel.cs

using System;

using System.Collections.ObjectModel;

using System.Reactive.Concurrency;

using System.Reactive.Disposables;

using System.Reactive.Linq;

using CommunityToolkit.Mvvm.ComponentModel;

using DynamicData.Binding;

using SimulationStorm.Simulation.Presentation.Models;

using SimulationStorm.Threading.Presentation;

using SimulationStorm.Utilities.Disposing;

using SimulationStorm.Utilities.Indexing;

namespace SimulationStorm.Simulation.Presentation.SimulationManager;

public partial class CommandQueueViewModel : DisposableObservableObject

{

#region Properties

public ReadOnlyObservableCollection<SimulationCommandModel> CommandQueue => \_commandQueue;

private bool \_isCommandQueueVisible;

public bool IsCommandQueueVisible

{

get => \_isCommandQueueVisible;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isCommandQueueVisible, value));

}

[ObservableProperty] private bool \_areThereCommands;

#endregion

#region Fields

private readonly IUiThreadScheduler \_uiThreadScheduler;

private ReadOnlyObservableCollection<SimulationCommandModel> \_commandQueue = null!;

#endregion

public CommandQueueViewModel(IUiThreadScheduler uiThreadScheduler, ISimulationManager simulationManager)

{

\_uiThreadScheduler = uiThreadScheduler;

WithDisposables(disposables =>

{

simulationManager.CommandQueue

.IndexItemsAndBind<ReadOnlyObservableCollection<SimulationCommand>, SimulationCommand, SimulationCommandModel>

(

command => new SimulationCommandModel(command.Name, command.ChangesWorld),

out \_commandQueue,

scheduler: uiThreadScheduler

)

.DisposeWith(disposables);

simulationManager.CommandQueue

.WhenValueChanged(x => x.Count)

.ObserveOn(uiThreadScheduler)

.Subscribe(commandCount => AreThereCommands = commandCount is not 0)

.DisposeWith(disposables);

});

}

}

Интерфейс камеры мира в файле IWorldCamera.cs

using System;

using System.Numerics;

using SimulationStorm.Primitives;

namespace SimulationStorm.Simulation.Presentation.Camera;

public interface IWorldCamera

{

#region Properties

Matrix3x2 Matrix { get; }

float Zoom { get; }

float ZoomChange { get; set; }

Vector2 Translation { get; }

float TranslationChange { get; set; }

#endregion

event EventHandler<CameraMatrixChangedEventArgs>? MatrixChanged;

#region Zooming methods

bool CanZoomToViewportCenter(CameraZoomDirection zoomDirection);

void ZoomToViewportCenter(CameraZoomDirection zoomDirection);

bool CanZoomToViewportCenter(float zoom);

void ZoomToViewportCenter(float zoom);

bool CanZoomToPoint(PointF point, CameraZoomDirection zoomDirection);

void ZoomToPoint(PointF point, CameraZoomDirection zoomDirection);

bool CanZoomToPoint(PointF point, float zoom);

void ZoomToPoint(PointF point, float zoom);

bool CanResetZoom();

void ResetZoom();

#endregion

#region Translation methods

void Translate(CameraTranslationDirection translationDirection);

void Translate(Vector2 offset);

void SetTranslation(Vector2 translation);

bool CanResetTranslation();

void ResetTranslation();

#endregion

}

Класс камеры в файле WorldCamera.cs

using System;

using System.ComponentModel;

using System.Numerics;

using System.Reactive.Disposables;

using System.Reactive.Linq;

using SimulationStorm.Graphics.Extensions;

using SimulationStorm.Primitives;

using SimulationStorm.Primitives.Extensions;

using SimulationStorm.Simulation.Presentation.Viewport;

using SimulationStorm.Utilities.Disposing;

namespace SimulationStorm.Simulation.Presentation.Camera;

public class WorldCamera : DisposableObject, IWorldCamera

{

#region Properties

public Matrix3x2 Matrix => \_matrix;

public float Zoom => \_matrix.GetScale().X;

public float ZoomChange { get; set; }

public Vector2 Translation => GetNormalizedTranslation();

public float TranslationChange { get; set; }

#endregion

public event EventHandler<CameraMatrixChangedEventArgs>? MatrixChanged;

#region Fields

private readonly IWorldViewport \_worldViewport;

private readonly WorldCameraOptions \_options;

private Matrix3x2 \_matrix = Matrix3x2.Identity;

#endregion

public WorldCamera(IWorldViewport worldViewport, WorldCameraOptions options)

{

\_worldViewport = worldViewport;

\_options = options;

ZoomChange = options.DefaultZoomChange;

TranslationChange = options.DefaultTranslationChange;

WithDisposables(disposables =>

{

var isInitialized = false;

Observable

.FromEventPattern<EventHandler<ViewportSizeChangedEventArgs>, ViewportSizeChangedEventArgs>

(

h => worldViewport.SizeChanged += h,

h => worldViewport.SizeChanged -= h

)

.Subscribe(\_ =>

{

if (!isInitialized)

{

ZoomToViewportCenter(options.DefaultZoom);

isInitialized = true;

return;

}

ZoomToViewportCenter(Zoom);

})

.DisposeWith(disposables);

});

}

#region Public methods

#region Zooming methods

public bool CanZoomToViewportCenter(CameraZoomDirection zoomDirection) =>

CanZoomToViewportCenter(GetZoomByZoomDirection(zoomDirection));

public void ZoomToViewportCenter(CameraZoomDirection zoomDirection) =>

ZoomToViewportCenter(GetZoomByZoomDirection(zoomDirection));

public bool CanZoomToViewportCenter(float zoom) =>

CanZoomToPoint(GetViewportCenterPoint(), zoom);

public void ZoomToViewportCenter(float zoom)

{

var previousMatrix = Matrix;

\_matrix = Matrix3x2.Identity;

ZoomToPointWithoutNotification(GetViewportCenterPoint(), zoom);

NotifyMatrixChangedIfNeeded(previousMatrix);

}

public bool CanZoomToPoint(PointF point, CameraZoomDirection zoomDirection) =>

CanZoomToPoint(point, GetZoomByZoomDirection(zoomDirection));

public void ZoomToPoint(PointF point, CameraZoomDirection zoomDirection) =>

ZoomToPoint(point, GetZoomByZoomDirection(zoomDirection));

public bool CanZoomToPoint(PointF point, float zoom) =>

zoom > \_options.ZoomRange.Minimum && zoom < \_options.ZoomRange.Maximum;

public void ZoomToPoint(PointF point, float zoom)

{

var previousMatrix = Matrix;

ZoomToPointWithoutNotification(point, zoom);

NotifyMatrixChangedIfNeeded(previousMatrix);

}

public bool CanResetZoom() => Math.Abs(Zoom - \_options.DefaultZoom) > 0.001f;

public void ResetZoom() => ZoomToViewportCenter(\_options.DefaultZoom);

#endregion

#region Translation methods

public void Translate(CameraTranslationDirection translationDirection) =>

Translate(GetOffsetByTranslationDirection(translationDirection));

public void Translate(Vector2 offset)

{

var previousMatrix = Matrix;

SetTranslationWithoutNotification(\_matrix.GetTranslation() + offset);

NotifyMatrixChangedIfNeeded(previousMatrix);

}

public void SetTranslation(Vector2 translation)

{

var previousMatrix = Matrix;

SetTranslationWithoutNotification(translation + GetDefaultTranslationForCurrentZoom());

NotifyMatrixChangedIfNeeded(previousMatrix);

}

public bool CanResetTranslation() => Translation != \_options.DefaultTranslation;

public void ResetTranslation() => SetTranslation(\_options.DefaultTranslation);

#endregion

#endregion

#region Private methods

private void ZoomToPointWithoutNotification(PointF point, float zoom)

{

zoom = Math.Clamp(zoom, \_options.ZoomRange.Minimum, \_options.ZoomRange.Maximum);

float zoomRatio = zoom / Zoom,

invertedZoomRatio = 1 - zoomRatio;

SetTranslationWithoutNotification(new Vector2

(

point.X \* invertedZoomRatio + \_matrix.GetTranslation().X \* zoomRatio,

point.Y \* invertedZoomRatio + \_matrix.GetTranslation().Y \* zoomRatio

));

\_matrix.SetScale(new Vector2(zoom, zoom));

}

private void SetTranslationWithoutNotification(Vector2 translation)

{

var translationRange = \_options.TranslationRange;

PointF minimumTranslation = translationRange.Minimum,

maximumTranslation = translationRange.Maximum;

translation.X = Math.Clamp(translation.X, minimumTranslation.X, maximumTranslation.X);

translation.Y = Math.Clamp(translation.Y, minimumTranslation.Y, maximumTranslation.Y);

\_matrix.SetTranslation(translation);

}

private Vector2 GetNormalizedTranslation() => \_matrix.GetTranslation() - GetDefaultTranslationForCurrentZoom();

private Vector2 GetDefaultTranslationForCurrentZoom()

{

var viewportCenter = GetViewportCenterPoint().ToVector2();

var sign = Zoom < 1 ? 1 : -1;

var translation = Zoom < 1 ? (1 - Zoom) \* viewportCenter : (Zoom - 1) \* viewportCenter;

return translation \* sign;

}

private PointF GetViewportCenterPoint() => new

(

\_worldViewport.Size.Width / 2f,

\_worldViewport.Size.Height / 2f

);

private float GetZoomByZoomDirection(CameraZoomDirection zoomDirection) =>

Zoom + ZoomChange \* (zoomDirection is CameraZoomDirection.ZoomIn ? 1 : -1);

private Vector2 GetOffsetByTranslationDirection(CameraTranslationDirection translationDirection) => translationDirection switch

{

CameraTranslationDirection.TopLeft => new Vector2(TranslationChange, TranslationChange),

CameraTranslationDirection.Top => new Vector2(0, TranslationChange),

CameraTranslationDirection.TopRight => new Vector2(-TranslationChange, TranslationChange),

CameraTranslationDirection.Right => new Vector2(-TranslationChange, 0),

CameraTranslationDirection.BottomRight => new Vector2(-TranslationChange, -TranslationChange),

CameraTranslationDirection.Bottom => new Vector2(0, -TranslationChange),

CameraTranslationDirection.BottomLeft => new Vector2(TranslationChange, -TranslationChange),

CameraTranslationDirection.Left => new Vector2(TranslationChange, 0),

\_ => throw new InvalidEnumArgumentException(nameof(translationDirection), (int)translationDirection, typeof(CameraTranslationDirection))

};

private void NotifyMatrixChangedIfNeeded(Matrix3x2 previousMatrix)

{

if (Matrix != previousMatrix)

MatrixChanged?.Invoke(this, new CameraMatrixChangedEventArgs(previousMatrix, Matrix));

}

#endregion

}

Класс модели представления строки состояния в файле StatusBarViewModel.cs

using System;

using System.Reactive.Concurrency;

using System.Reactive.Disposables;

using System.Reactive.Linq;

using SimulationStorm.Simulation.Presentation.Renderer;

using SimulationStorm.Simulation.Presentation.SimulationManager;

using SimulationStorm.Simulation.Presentation.SimulationRenderer;

using SimulationStorm.Simulation.Presentation.WorldRenderer;

using SimulationStorm.Threading.Presentation;

using SimulationStorm.Utilities.Disposing;

namespace SimulationStorm.Simulation.Presentation.StatusBar;

public class StatusBarViewModel : DisposableObservableObject

{

#region Properties

#region Simulation command execution

private bool \_isCommandProgressVisible;

public bool IsCommandProgressVisible

{

get => \_isCommandProgressVisible;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isCommandProgressVisible, value));

}

private bool \_isCommandTimeVisible;

public bool IsCommandTimeVisible

{

get => \_isCommandTimeVisible;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isCommandTimeVisible, value));

}

private SimulationCommand? \_executingCommand;

public SimulationCommand? ExecutingCommand

{

get => \_executingCommand;

set => \_uiThreadScheduler.Schedule(() =>

{

if (SetProperty(ref \_executingCommand, value))

OnPropertyChanged(nameof(IsCommandExecuting));

});

}

public bool IsCommandExecuting => ExecutingCommand is not null;

private TimeSpan \_commandTime;

public TimeSpan CommandTime

{

get => \_commandTime;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_commandTime, value));

}

#endregion

#region Simulation rendering

private bool \_isSimulationRenderingProgressVisible;

public bool IsSimulationRenderingProgressVisible

{

get => \_isSimulationRenderingProgressVisible;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isSimulationRenderingProgressVisible, value));

}

private bool \_isSimulationRenderingInProgress;

public bool IsSimulationRenderingInProgress

{

get => \_isSimulationRenderingInProgress;

private set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isSimulationRenderingInProgress, value));

}

private bool \_isSimulationRenderingTimeVisible;

public bool IsSimulationRenderingTimeVisible

{

get => \_isSimulationRenderingTimeVisible;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isSimulationRenderingTimeVisible, value));

}

private TimeSpan \_simulationRenderingTime;

public TimeSpan SimulationRenderingTime

{

get => \_simulationRenderingTime;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_simulationRenderingTime, value));

}

#endregion

#region World rendering

private bool \_isWorldRenderingTimeVisible;

public bool IsWorldRenderingTimeVisible

{

get => \_isWorldRenderingTimeVisible;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_isWorldRenderingTimeVisible, value));

}

private TimeSpan \_worldRenderingTime;

public TimeSpan WorldRenderingTime

{

get => \_worldRenderingTime;

set => \_uiThreadScheduler.Schedule(() => SetProperty(ref \_worldRenderingTime, value));

}

#endregion

#endregion

private readonly IUiThreadScheduler \_uiThreadScheduler;

public StatusBarViewModel

(

IUiThreadScheduler uiThreadScheduler,

ISimulationManager simulationManager,

ISimulationRenderer simulationRenderer,

IWorldRenderer worldRenderer,

StatusBarOptions options)

{

\_uiThreadScheduler = uiThreadScheduler;

\_isCommandProgressVisible = options.IsCommandProgressVisible;

\_isCommandTimeVisible = options.IsCommandTimeVisible;

\_isSimulationRenderingProgressVisible = options.IsSimulationRenderingProgressVisible;

\_isSimulationRenderingTimeVisible = options.IsSimulationRenderingTimeVisible;

\_isWorldRenderingTimeVisible = options.IsWorldRenderingTimeVisible;

WithDisposables(disposables =>

{

Observable

.FromEventPattern<EventHandler<SimulationCommandExecutingEventArgs>, SimulationCommandExecutingEventArgs>

(

h => simulationManager.CommandExecuting += h,

h => simulationManager.CommandExecuting -= h

)

.Where(\_ => IsCommandProgressVisible)

.Select(e => e.EventArgs)

.Subscribe(e => ExecutingCommand = e.Command)

.DisposeWith(disposables);

Observable

.FromEventPattern<EventHandler<SimulationCommandExecutedEventArgs>, SimulationCommandExecutedEventArgs>

(

h => simulationManager.CommandExecuted += h,

h => simulationManager.CommandExecuted -= h

)

.Do(\_ => ExecutingCommand = null)

.Where(\_ => IsCommandTimeVisible)

.Select(e => e.EventArgs)

.Subscribe(e => CommandTime = e.ElapsedTime)

.DisposeWith(disposables);

Observable

.FromEventPattern<EventHandler, EventArgs>

(

h => simulationRenderer.RenderingStarted += h,

h => simulationRenderer.RenderingStarted -= h

)

.Where(\_ => IsSimulationRenderingProgressVisible)

.Subscribe(\_ => IsSimulationRenderingInProgress = true)

.DisposeWith(disposables);

Observable

.FromEventPattern<EventHandler<SimulationRenderingCompletedEventArgs>, SimulationRenderingCompletedEventArgs>

(

h => simulationRenderer.RenderingCompleted += h,

h => simulationRenderer.RenderingCompleted -= h

)

.Do(\_ => IsSimulationRenderingInProgress = false)

.Where(\_ => IsSimulationRenderingTimeVisible)

.Select(e => e.EventArgs)

.Subscribe(e => SimulationRenderingTime = e.ElapsedTime)

.DisposeWith(disposables);

Observable

.FromEventPattern<EventHandler<RenderingCompletedEventArgs>, RenderingCompletedEventArgs>

(

h => worldRenderer.RenderingCompleted += h,

h => worldRenderer.RenderingCompleted -= h

)

.Where(\_ => IsWorldRenderingTimeVisible)

.Select(e => e.EventArgs)

.Subscribe(e => WorldRenderingTime = e.ElapsedTime)

.DisposeWith(disposables);

});

}

}

Класс сервиса отрисовки мира клеточной симуляции в файле BoundedCellularWorldRenderer.cs

using System;

using System.Collections.Generic;

using System.Reactive.Disposables;

using System.Reactive.Linq;

using System.Threading.Tasks;

using DotNext.Threading;

using SimulationStorm.Graphics;

using SimulationStorm.Presentation.Colors;

using SimulationStorm.Primitives;

using SimulationStorm.Simulation.Cellular.Presentation.Models;

using SimulationStorm.Simulation.Presentation.Camera;

using SimulationStorm.Simulation.Presentation.SimulationRenderer;

using SimulationStorm.Simulation.Presentation.Viewport;

using SimulationStorm.Simulation.Presentation.WorldRenderer;

using SimulationStorm.Utilities.Benchmarking;

using SimulationStorm.Utilities.Disposing;

namespace SimulationStorm.Simulation.Cellular.Presentation.Services;

public class BoundedCellularWorldRenderer : WorldRendererBase, IBoundedCellularWorldRenderer

{

#region Properties

public Color DefaultBackgroundColor => \_uiColorProvider.BackgroundColor;

public Color BackgroundColor

{

get => \_backgroundColor;

set

{

if (value == \_backgroundColor)

return;

\_backgroundColor = value;

OnPropertyChanged();

RequestRerender();

}

}

public Rect SimulationImageRect => GetSimulationImageRect();

public bool IsGridLinesVisible

{

get => \_isGridLinesVisible;

set

{

if (value == \_isGridLinesVisible)

return;

\_isGridLinesVisible = value;

OnPropertyChanged();

RequestRerender();

}

}

public Color GridLinesColor

{

get => \_gridLinesPaint.Color;

set

{

if (value == \_gridLinesPaint.Color)

return;

\_gridLinesPaint.Color = value;

OnPropertyChanged();

RequestRerender();

}

}

public Color HoveredCellColor

{

get => \_hoveredCellPaint.Color;

set

{

if (value == \_hoveredCellPaint.Color)

return;

\_hoveredCellPaint.Color = value;

OnPropertyChanged();

RequestRerender();

}

}

public Color PressedCellColor

{

get => \_pressedCellPaint.Color;

set

{

if (value == \_pressedCellPaint.Color)

return;

\_pressedCellPaint.Color = value;

OnPropertyChanged();

RequestRerender();

}

}

public (Point Cell, CellularWorldPointedCellState State)? PointedCell

{

get => \_pointedCell;

set

{

if (value == \_pointedCell)

return;

\_pointedCell = value;

OnPropertyChanged();

RequestRerender();

}

}

#endregion

#region Fields

private readonly ISimulationRenderer \_simulationRenderer;

private IBitmap? \_simulationImage;

private readonly AsyncExclusiveLock \_simulationImageLock = new();

private readonly IUiColorProvider \_uiColorProvider;

private Color \_backgroundColor;

private readonly IPaint \_gridLinesPaint;

private bool \_isGridLinesVisible;

private readonly IPaint \_hoveredCellPaint;

private readonly IPaint \_pressedCellPaint;

private readonly IPaint \_simulationImageBorderPaint;

private (Point Cell, CellularWorldPointedCellState State)? \_pointedCell;

#endregion

public BoundedCellularWorldRenderer

(

IGraphicsFactory graphicsFactory,

IBenchmarkingService benchmarkingService,

IWorldViewport worldViewport,

IWorldCamera worldCamera,

ISimulationRenderer simulationRenderer,

IUiColorProvider uiColorProvider,

ICellularWorldRendererOptions options

)

: base(graphicsFactory, benchmarkingService, worldViewport, worldCamera)

{

\_simulationRenderer = simulationRenderer;

\_uiColorProvider = uiColorProvider;

\_backgroundColor = uiColorProvider.BackgroundColor;

\_isGridLinesVisible = options.IsGridLinesVisible;

\_gridLinesPaint = GraphicsFactory.CreatePaint();

\_gridLinesPaint.Color = options.GridLinesColor;

\_hoveredCellPaint = GraphicsFactory.CreatePaint();

\_hoveredCellPaint.Color = options.HoveredCellColor;

\_pressedCellPaint = GraphicsFactory.CreatePaint();

\_pressedCellPaint.Color = options.PressedCellColor;

\_simulationImageBorderPaint = GraphicsFactory.CreatePaint();

\_simulationImageBorderPaint.Color = HoveredCellColor;

\_simulationImageBorderPaint.Style = PaintStyle.Stroke;

WithDisposables(disposables =>

{

Observable

.FromEventPattern<EventHandler<UiColorChangedEventArgs>, UiColorChangedEventArgs>

(

h => \_uiColorProvider.BackgroundColorChanged += h,

h => \_uiColorProvider.BackgroundColorChanged -= h

)

.Select(e => e.EventArgs)

.Where(e => BackgroundColor == e.PreviousColor) // Set background color from ui color provider only if it was not changed manually

.Subscribe(e => BackgroundColor = e.NewColor)

.DisposeWith(disposables);

Observable

.FromEventPattern<EventHandler<SimulationRenderingCompletedEventArgs>, SimulationRenderingCompletedEventArgs>

(

h => \_simulationRenderer.RenderingCompleted += h,

h => \_simulationRenderer.RenderingCompleted -= h

)

.Subscribe(e =>

{

\_ = UpdateSimulationImageCopy();

RequestRerender();

})

.DisposeWith(disposables);

disposables.AddRange(\_gridLinesPaint, \_hoveredCellPaint, \_pressedCellPaint);

});

RequestRerender();

}

protected override async Task RenderAsync(ICanvas canvas)

{

canvas.Clear(BackgroundColor);

canvas.SetMatrix(WorldCamera.Matrix);

await RenderSimulationImage(canvas);

RenderPointedCellIfNeeded(canvas);

RenderGridLinesIfNeeded(canvas);

RenderSimulationImageBorder(canvas);

}

#region Simulation image rendering

private async Task UpdateSimulationImageCopy()

{

await \_simulationImageLock.AcquireAsync();

\_simulationImage?.Dispose();

\_simulationImage = \_simulationRenderer.RenderedImage?.Copy();

\_simulationImageLock.Release();

}

private async Task RenderSimulationImage(ICanvas canvas)

{

await \_simulationImageLock.AcquireAsync();

if (\_simulationImage is not null)

{

var position = GetPositionToRenderSimulationImageAt().ToPointF();

canvas.DrawBitmap(\_simulationImage, position);

}

\_simulationImageLock.Release();

}

private Point GetPositionToRenderSimulationImageAt() =>

GetSimulationImageRect().Position;

private Rect GetSimulationImageRect() =>

CenterChildSizeRelativeToParentSize(WorldViewport.Size, \_simulationRenderer.RenderedImage?.Size ?? Size.Zero);

#endregion

#region Pointed cell rendering

private bool IsPointedCellRenderingNeeded() => WorldCamera.Zoom >= 1 && PointedCell is not null;

private void RenderPointedCellIfNeeded(ICanvas canvas)

{

if (!IsPointedCellRenderingNeeded())

return;

var pointedCell = PointedCell!.Value;

var simulationImagePosition = GetPositionToRenderSimulationImageAt();

var cellAbsolutePosition = pointedCell.Cell + simulationImagePosition;

var paint = pointedCell.State is CellularWorldPointedCellState.Hovered ? \_hoveredCellPaint : \_pressedCellPaint;

canvas.DrawRect(cellAbsolutePosition.X, cellAbsolutePosition.Y, 1, 1, paint);

}

#endregion

#region Grid lines rendering

private bool IsGridLinesRenderingNeeded() => WorldCamera.Zoom >= 2 && IsGridLinesVisible;

private void RenderGridLinesIfNeeded(ICanvas canvas)

{

if (!IsGridLinesRenderingNeeded())

return;

var (startX, startY) = GetPositionToRenderSimulationImageAt();

int endX = startX + \_simulationRenderer.RenderedImage!.Size.Width,

endY = startY + \_simulationRenderer.RenderedImage!.Size.Height;

var gridLines = new List<LineF>();

// Vertical lines

for (var x = startX; x <= endX; x++)

gridLines.Add(new LineF(x, startY, x, endY));

// Horizontal lines

for (var y = startY; y <= endY; y++)

gridLines.Add(new LineF(startX, y, endX, y));

canvas.DrawLines(gridLines, \_gridLinesPaint);

}

#endregion

private void RenderSimulationImageBorder(ICanvas canvas)

{

var simulationImageRect = GetSimulationImageRect();

var borderRect = new RectF(simulationImageRect.Left, simulationImageRect.Top,

simulationImageRect.Right + 1, simulationImageRect.Bottom + 1);

canvas.DrawRect(borderRect, \_simulationImageBorderPaint);

}

private static Rect CenterChildSizeRelativeToParentSize(Size parentSize, Size childSize)

{

var position = new Point

(

(parentSize.Width - childSize.Width) / 2,

(parentSize.Height - childSize.Height) / 2

);

var end = new Point

(

position.X + childSize.Width - 1,

position.Y + childSize.Height - 1

);

return new Rect

(

position.X,

position.Y,

end.X,

end.Y

);

}

}

Класс универсальной коллекции в файле UniversalCollection.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.ComponentModel;

using System.Linq;

using DynamicData;

using SimulationStorm.Collections.Lists;

using SimulationStorm.Collections.Pointed;

using SimulationStorm.Collections.StorageControl;

namespace SimulationStorm.Collections.Universal;

public class UniversalCollection<T> : IUniversalCollection<T>

{

private const int DefaultPointerPosition = -1;

#region Properties

public int Count => \_listStorage.Count;

public T this[int index]

{

get => \_listStorage[index];

set => \_listStorage[index] = value;

}

public CollectionStorageLocation StorageLocation

{

get => \_storageLocation;

set => SetStorageLocation(value);

}

public int Capacity

{

get => \_capacity;

set => SetCapacity(value);

}

public int PointerPosition { get; private set; } = DefaultPointerPosition;

// Todo: For now, set this property here.

public PointerBehavior PointerBehavior { get; set; } =

PointerBehavior.RemoveItemsAheadOfPointerBeforeAddingOrInsertingItems

| PointerBehavior.MoveToEndAfterAddingOrInsertingItems;

public bool IsReadOnly => false;

public bool UseRemoveActionOverReset { get; set; }

int ICollection<T>.Count => \_listStorage.Count;

int IReadOnlyCollection<T>.Count => \_listStorage.Count;

#endregion

#region Events

public event NotifyCollectionChangedEventHandler? CollectionChanged;

public event PropertyChangedEventHandler? PropertyChanged;

public event EventHandler<CollectionPointerMovedEventArgs>? PointerMoved;

#endregion

#region Fields

private readonly IListFactory \_listFactory;

private IExtendedList<T> \_listStorage;

private CollectionStorageLocation \_storageLocation;

private int \_capacity;

// private readonly IList<Range> \_hiddenRanges = new List<Range>();

#endregion

public UniversalCollection(IListFactory listFactory, CollectionStorageLocation storageLocation, int capacity)

{

\_listFactory = listFactory;

\_storageLocation = storageLocation;

ValidateCapacity(capacity);

\_capacity = capacity;

\_listStorage = \_listFactory.CreateList<T>(StorageLocation, Capacity);

}

#region Public methods

#region Adding methods

public void Add(T item) => Insert(Count, item);

public void AddRange(IEnumerable<T> collection) => InsertRange(collection, Count);

public void Insert(int index, T item)

{

var oldCount = Count;

RemoveItemsAfterPointerIfThereAreIfNeeded(out var removedItemCount);

index -= removedItemCount;

EnsureThereIsRequiredSpace(1, out removedItemCount);

index -= removedItemCount;

\_listStorage.Insert(index, item);

NotifyCountChangedIfChanged(oldCount);

NotifyCollectionChanged

(

new NotifyCollectionChangedEventArgs

(

NotifyCollectionChangedAction.Add,

item,

index

)

);

MovePointerToEndIfNeeded();

}

public void InsertRange(IEnumerable<T> collection, int index)

{

var oldCount = Count;

var itemsToInsert = collection

.TakeLast(Capacity)

.ToList();

RemoveItemsAfterPointerIfThereAreIfNeeded(out var removedItemCount);

index -= removedItemCount;

EnsureThereIsRequiredSpace(itemsToInsert.Count, out removedItemCount);

index -= removedItemCount;

\_listStorage.InsertRange(itemsToInsert, index);

NotifyCountChangedIfChanged(oldCount);

NotifyCollectionChanged

(

new NotifyCollectionChangedEventArgs

(

NotifyCollectionChangedAction.Add,

itemsToInsert,

index

)

);

MovePointerToEndIfNeeded();

}

#endregion

#region Removing methods

public bool Remove(T item)

{

var index = \_listStorage.IndexOf(item);

if (index is -1)

return false;

RemoveAt(index);

return true;

}

public void RemoveAt(int index)

{

var oldCount = Count;

var itemToRemove = \_listStorage[index];

\_listStorage.RemoveAt(index);

CheckPointerPositionAndMoveIfNeeded();

NotifyCountChangedIfChanged(oldCount);

NotifyCollectionChanged

(

new NotifyCollectionChangedEventArgs

(

NotifyCollectionChangedAction.Remove,

itemToRemove,

index

)

);

}

public void RemoveRange(int index, int count)

{

var oldCount = Count;

var itemsToRemove = new List<T>(count);

for (var i = index; i < index + count; i++)

itemsToRemove.Add(\_listStorage[i]);

\_listStorage.RemoveRange(index, count);

CheckPointerPositionAndMoveIfNeeded();

NotifyCountChangedIfChanged(oldCount);

NotifyCollectionChanged

(

new NotifyCollectionChangedEventArgs

(

NotifyCollectionChangedAction.Remove,

itemsToRemove,

index

)

);

}

public void Clear()

{

if (UseRemoveActionOverReset)

ClearUsingRemoveRange();

else

ClearUsingClear();

}

#endregion

public void Move(int original, int destination)

{

var item = \_listStorage[original];

\_listStorage.RemoveAt(original);

\_listStorage.Insert(destination, item);

NotifyCollectionChanged

(

new NotifyCollectionChangedEventArgs

(

NotifyCollectionChangedAction.Move,

item,

destination,

original

)

);

}

#region Searching methods

public bool Contains(T item) => \_listStorage.Contains(item);

public int IndexOf(T item) => \_listStorage.IndexOf(item);

#endregion

public void MovePointer(int newPosition)

{

ValidatePointerPosition(newPosition);

var oldPosition = PointerPosition;

if (newPosition == oldPosition)

return;

PointerPosition = newPosition;

NotifyPointerMoved(oldPosition);

}

// #region Visibility changing methods

// // Here, it also needed to create custom enumerator.

// // And permit collection modifying when there are hidden items.

// public void HideRange(int index, int count)

// {

// // Check if index already hidden

// var endIndex = index + count;

// if (\_hiddenRanges.Any(r => r.StartIndex() >= index || r.EndIndex() <= endIndex))

// ; // is already hidden

//

// \_hiddenRanges.Add(new Range(new Index(index), new Index(index + count)));

//

// var itemsToHide = \_listStorage.Skip(index).Take(count);

// NotifyCollectionChanged

// (

// new NotifyCollectionChangedEventArgs

// (

// NotifyCollectionChangedAction.Remove,

// itemsToHide,

// index

// )

// );

// }

//

// public void RevealRange(int index, int count)

// {

// }

//

// public void RemoveHidden()

// {

// // Take all hidden items and remove them.

// // var hiddenItems = ;

// }

// #endregion

public void CopyTo(T[] array, int arrayIndex) => \_listStorage.CopyTo(array, arrayIndex);

public IEnumerator<T> GetEnumerator() => \_listStorage.GetEnumerator();

#endregion

IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

#region Private methods

#region Storage location setting

private void SetStorageLocation(CollectionStorageLocation newStorageLocation)

{

var newStorage = \_listFactory.CreateList<T>(newStorageLocation, Count);

for (var i = 0; i < Count; i++)

newStorage.Add(\_listStorage[i]);

\_listStorage.Clear();

\_listStorage = newStorage;

\_storageLocation = newStorageLocation;

NotifyStorageLocationChanged();

}

private void NotifyStorageLocationChanged() => NotifyPropertyChanged(nameof(StorageLocation));

#endregion

#region Capacity setting

private void SetCapacity(int newCapacity)

{

ValidateCapacity(newCapacity);

if (Count > newCapacity)

{

var itemCountToRemove = Count - newCapacity;

RemoveRange(0, itemCountToRemove);

}

\_capacity = newCapacity;

NotifyCapacityChanged();

}

private static void ValidateCapacity(int capacity) =>

ArgumentOutOfRangeException.ThrowIfLessThan(capacity, 1, nameof(capacity));

private void NotifyCapacityChanged() => NotifyPropertyChanged(nameof(Capacity));

#endregion

#region Pointer movement

private void MovePointerToEndIfNeeded()

{

if (PointerBehavior.HasFlag(PointerBehavior.MoveToEndAfterAddingOrInsertingItems))

MovePointerToEnd();

}

private void MovePointerToEnd() => MovePointer(GetMaxIndex());

private void CheckPointerPositionAndMoveIfNeeded()

{

var maxItemIndex = GetMaxIndex();

if (PointerPosition > maxItemIndex)

MovePointer(maxItemIndex);

}

private void ValidatePointerPosition(int pointerPosition)

{

ArgumentOutOfRangeException.ThrowIfLessThan(pointerPosition, DefaultPointerPosition, nameof(pointerPosition));

ArgumentOutOfRangeException.ThrowIfGreaterThan(pointerPosition, GetMaxIndex(), nameof(pointerPosition));

}

#endregion

#region Removing items after pointer

private void RemoveItemsAfterPointerIfThereAreIfNeeded(out int removedItemCount)

{

removedItemCount = 0;

if (PointerBehavior.HasFlag(PointerBehavior.RemoveItemsAheadOfPointerBeforeAddingOrInsertingItems))

RemoveItemsAfterPointerIfThereAre(out removedItemCount);

}

private void RemoveItemsAfterPointerIfThereAre(out int removedItemCount)

{

removedItemCount = GetMaxIndex() - PointerPosition;

if (removedItemCount > 0)

RemoveRange(PointerPosition + 1, removedItemCount);

}

#endregion

#region Notification methods

private void NotifyCountChangedIfChanged(int oldCount)

{

if (Count != oldCount)

NotifyPropertyChanged(nameof(Count));

}

private void NotifyPointerMoved(int oldPosition) =>

PointerMoved?.Invoke(this, new CollectionPointerMovedEventArgs(oldPosition, PointerPosition));

private void NotifyPropertyChanged(string propertyName) =>

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

private void NotifyCollectionChanged(NotifyCollectionChangedEventArgs e) => CollectionChanged?.Invoke(this, e);

#endregion

#region Clearing methods

private void ClearUsingRemoveRange() => RemoveRange(0, Count);

private void ClearUsingClear()

{

var oldCount = Count;

\_listStorage.Clear();

MovePointer(DefaultPointerPosition);

NotifyCountChangedIfChanged(oldCount);

NotifyCollectionChanged(new NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset));

}

#endregion

private void EnsureThereIsRequiredSpace(int requiredSpace, out int removedItemCount)

{

var availableSpace = GetAvailableSpace();

if (availableSpace >= requiredSpace)

{

removedItemCount = 0;

return;

}

removedItemCount = requiredSpace - availableSpace;

RemoveRange(0, removedItemCount);

}

private int GetMaxIndex() => Count - 1;

private int GetAvailableSpace() => Capacity - Count;

#endregion

}