Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Отчёт по технологической (проектно-технологической) практике на тему:

РАЗРАБОТКА КЛЕТОЧНОГО АВТОМАТА «ИГРА ЖИЗНЬ» НА ПЛАТФОРМЕ UNITY

Выполнил студент 5.205-1 группы:

_____ Б. Н. Ергали

«___» _____ 2024 г.

Проверил: ст.пр.

_____ И. А. Шмаков

«___» _____ 2024 г.

Барнаул 2024 г.

РЕФЕРАТ

Полный объём работы составляет 35 страниц, включая 12 рисунков.

В отчёте содержатся сведения о курсовой работе.

Курсовая работа заключается в создании кроссплатформенной программы, представляющей Клеточный автомат «Игра жизнь».

В отчёте приведено описание используемой библиотеки Unity, инструментария и системы контроля версий для написания игры на платформе Unity, с использованием языка C#.

Наряду с этим представлены проверка работспособности программы, блок-схема, полный код программы.

Ключевые слова: программа, игра, пакман, алгоритм, кроссплатформенная программа.

Отчёт оформлен с помощью системы компьютерной вёрстки TEX и его расширения XETEX из дистрибутива *TeX Live*.

ABSTRACT

The total amount of work is 35 page’s, include 0 image’s and 0 table’s.

The report contains information about the course work.

The course work consists of creating a cross-platform program representing the Cellular Automaton «Game of Life».

The report provides a description of the Unity library, tools and version control system used to write a game on the Unity platform using the C# language.

Along with this, a program functionality test, a flow chart, and the complete program code are presented.

Keyword: program, game, pacman, algorithm, cross-platform program.

Report is framed using the computer layout system TEX and its extension XETEX from the distribution *TeX Live*.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5
1.1. Техническое задание 5
1.2. Игровой движок 5
1.3. Инструментарий 6
1.4. Клеточный автомат «Игра жизнь» 7
1.4.1. История клеточного автомата «Игра жизнь». 7
1.4.2. Правила игры 8
2. ПРАКТИЧЕСКАЯ ЧАСТЬ. ЭТАПЫ РАЗРАБОТКИ ИГРЫ. 10
2.1. Подготовка и установка окружения. 10
2.2. Реализация основного алгоритма клеточного автомата. 12
2.3. Создание пользовательского интерфейса (UI) 14
2.4. Блок-схема. 15
2.5. Сборка программы 16
2.6. Тестирование программы 18
ЗАКЛЮЧЕНИЕ 19
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 20
ПРИЛОЖЕНИЕ 22

ВВЕДЕНИЕ

Актуальность: Развитие технологий и увеличение доступности программных инструментов для создания игр открывает широкие возможности для исследования сложных систем и процессов через игровую форму. Применение языка программирования **C#** [1] в среде разработки **Unity** [2] для создания таких игр объединяет в себе глубокую актуальность как в образовательной, так и в научно-исследовательской сферах.

Создание игры на основе клеточного автомата в среде Unity 2D позволяет не только демонстрировать сложные поведенческие паттерны [3] и взаимодействия между элементами системы в наглядной и доступной форме, но и дает возможность пользователям влиять на процессы внутри игры, экспериментировать с параметрами и наблюдать за их эффектами.

Интерес к созданию собственных миров и экосистем, где каждое действие может привести к непредсказуемым последствиям, высок среди широкой аудитории. Таким образом, разработка игры, основанной на принципах клеточного автомата, является актуальной задачей, способной привлечь внимание как образовательных учреждений, так и широкой публики, интересующейся наукой, технологиями и играми.

Цель: Создать кроссплатформенный клеточный автомат «Игра жизнь» с использованием языка программирования C#, на платформе Unity.

Задачи:

1. Изучить платформу Unity.
2. Изучить алгоритмы клеточных автоматов.
3. Изучить принцип ООП в C#.
4. Изучить сборку проекта в Unity.
5. Разработать пользовательский интерфейс для взаимодействия с клеточным автоматом.
6. Провести тестирование и отладку проекта на различных устройствах.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Техническое задание

1. Кроссплатформенное приложение, способное запускаться на операционных системах Windows и GNU/Linux.
2. Написание клеточного автомата «Игра жизнь» на платформе Unity, с использованием языка C#.
3. Возможность выбора карты.
4. Возможность генерации случайной карты.
5. Возможность создания новых карт для клеточного автомата.
6. Наличие в игре кнопки «Помощь», при нажатии на которую выводится информация о взаимодействии с игрой.

1.2. Игровой движок

Для создания кроссплатформенной программы на языке программирования C# был использован игровой движок Unity

Unity — это мощная кросс-платформенная среда разработки для создания широкого спектра интерактивного контента, включая видеоигры, обучающие программы, архитектурные визуализации и виртуальную реальность. Она предоставляет интегрированные инструменты для работы как с 2D, так и с 3D-графикой, что делает Unity популярным выбором среди разработчиков игр и интерактивных приложений различного масштаба — от независимых проектов до крупных игровых студий.

Основные особенности и преимущества Unity:

- **Кросс-платформенность:** Unity поддерживает более 25 платформ, включая Windows, macOS, Linux, Android, iOS, WebGL, PlayStation, Xbox и многие другие, что позволяет разработчикам с легкостью портировать свои проекты на различные устройства и экосистемы;
- **Интуитивный интерфейс:** Unity обладает удобным и понятным пользовательским интерфейсом, который упрощает процесс разработки и

позволяет даже начинающим разработчикам быстро освоиться в программе.;

- **Мощные инструменты для работы с графикой и анимацией:** В Unity встроены продвинутые инструменты для создания и редактирования 2D и 3D графики, анимаций, а также системы частиц, что делает возможным создание визуально привлекательных и технически сложных проектов;
- **Скриптование на C#:** Для создания логики игр и приложений в Unity используется язык программирования C#. Это обеспечивает гибкость и мощь в реализации функционала, делая возможным создание сложных систем взаимодействия и поведения объектов в игровом мире;
- **Активное сообщество и обширная база знаний:** Unity имеет одно из самых больших и активных сообществ разработчиков в мире. Это обеспечивает доступ к огромному количеству учебных материалов, руководств, а также готовых активов и плагинов, которые можно использовать в своих проектах;
- **Unity Asset Store:** Магазин активов Unity предлагает тысячи готовых ресурсов и инструментов, включая модели, текстуры, скрипты, инструменты для интеграции с другими сервисами и многое другое, что значительно ускоряет процесс разработки и позволяет сосредоточиться на уникальных аспектах проекта.

1.3. Инструментарий

Для написания отчёта с помощью системы компьютерной верстки в TEX была использована сайт Overleaf.

Для написания кода программы была использована IDE Microsoft Visual Studio.

Для работы с изображениями, используемых в ходе разработки программы, был использован графический редактор Adobe Photoshop 2023 [4].

Для хранения проекта была выбрана система контроля версия GitHub [5].

Проверка работоспособности и сборка программы выполнялась на системе:

- **ОС**: *Windows 10*
- **ЦП**: *AMD Ryzen 5 4600H*
- **ОЗУ**: *8gb*
- **Видеокарта**: *NVIDIA GeForce GTX 1650 Ti*

1.4. Клеточный автомат «Игра жизнь»

1.4.1. История клеточного автомата «Игра жизнь».

«Игра жизнь» — это клеточный автомат, созданный британским математиком **Джоном Конвеем** [6] в 1970 году. Эта «игра» стала одним из самых известных примеров клеточного автомата и занимает особое место в теории вычислительных систем и математической биологии.

История создания «Игры жизнь» началась с желания Конвея изучить возможности простых математических моделей для имитации жизни и эволюции. Интересно, что, несмотря на простоту правил, «Игра жизнь» способна продемонстрировать чрезвычайно сложное и непредсказуемое поведение, что делает ее удивительным примером эмерджентности — появления сложных структур и паттернов из простых взаимодействий. Вместе с коллегами из Кембриджского университета он разработал первые версии игры, которые изначально испытывали на досках для шахмат, заполняя клетки «живыми» или «мертвыми» состояниями и просчитывая следующие поколения вручную.

Опубликованная в октябре 1970 года в колонке Мартина Гарднера в журнале «Scientific American», «Игра жизнь» моментально привлекла внимание широкой аудитории. Люди были поражены идеей, что такие простые правила могут породить бесконечное множество удивительных форм и поведенческих паттернов. В то время как некоторые находили в ней отражение эволюционных процессов и даже возможность изучения искусственного ин-

теллекта, другие видели в «Игре жизнь» искусство и новую форму развлечения.

Важной вехой в истории «Игры жизнь» стало ее распространение среди первых пользователей персональных компьютеров. Программные реализации игры позволили автоматизировать вычисления и наблюдать за развитием системы в реальном времени, что существенно углубило исследования и способствовало обнаружению всё новых и новых удивительных структур, таких как «планеры» (космические корабли), «планерные пушки» и самовоспроизводящиеся конструкции.

«Игра жизнь» сыграла значительную роль в развитии теории клеточных автоматов и комплексных систем, вдохновив множество ученых и разработчиков на создание собственных моделей и исследования в области искусственной жизни, системной биологии и даже криптографии. Она продолжает оставаться популярной не только среди ученых, но и среди художников, программистов и любителей загадок, представляя собой универсальный язык для исследования сложности из простоты.

1.4.2. Правила игры

Правила «Игры жизнь» Джона Конвея просты, но в то же время обладают удивительной глубиной, позволяя моделировать сложные динамические процессы. «Игра» происходит на бесконечной двумерной сетке клеток, каждая из которых может находиться в одном из двух состояний: быть «живой» или «мертвой». Состояние каждой клетки в следующем поколении определяется ее текущим состоянием и состояниями восьми ее соседей по горизонтали, вертикали и диагоналям.

Правила перехода из поколения в поколение следующие:

- **Рождение**: Мертвая клетка становится живой в следующем поколении, если ровно три из ее восьми соседей живы в текущем поколении. Это правило символизирует репродуктивное «рождение» из стабильной, но не перенаселенной среды;

- **Смерть от одиночества**: Живая клетка становится мертвой в следующем поколении, если два или менее из ее восьми соседей живы в текущем поколении. Это моделирует смерть из-за недостатка «социальных» взаимодействий;
- **Выживание**: Живая клетка остается живой в следующем поколении, если у нее два или три живых соседа. Это условие обеспечивает оптимальную среду для устойчивого существования;
- **Смерть от перенаселения**: Живая клетка становится мертвой в следующем поколении, если четыре и более из ее восьми соседей живы в текущем поколении. Это представляет смерть из-за слишком высокой конкуренции за ресурсы.

Несмотря на кажущуюся простоту, эти правила порождают удивительно разнообразные и часто непредсказуемые паттерны, включая стационарные фигуры, осциллирующие структуры, которые повторяются через определенное количество поколений, и даже «космические корабли», перемещающиеся по сетке. Эти паттерны и их взаимодействия исследуются в рамках «Игры жизнь», демонстрируя сложность, возникающую из простых начал.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ. ЭТАПЫ РАЗРАБОТКИ ИГРЫ.

2.1. Подготовка и установка окружения.

Первый шаг на пути к созданию игры — установка **Unity Hub** [7].

Unity Hub — это центральное приложение для управления проектами, версиями Unity и лицензиями.

После установки **Unity Hub**, можно было начать установку самой платформы **Unity**. Затем нужно было добавить наиболее стабильную версию **Unity** и выбрать при необходимости дополнительные компоненты, такие как поддержка различный ОС.

Для написания кода на C# и работы с Unity была необходима интегрированная среда разработки, или же **IDE**. Было решено использовать **Visual Studio** [8], так как она отлично интегрируется с Unity и предоставляет все необходимые инструменты для разработки. Так как Visual Studio уже был установлен, нужно было дополнительно к нему загрузить рабочую нагрузку **Game development with Unity** [9]. Это обеспечило установку всех нужных компонентов для работы с Unity и написания кода на C#.

После установки всех необходимых инструментов был создан новый проект в Unity. В выборе шаблона проекта, выбор пал на 2D, т.к. необходимости в 3D пространстве не было и все можно было выполнить в 2D.

Перед началом работы непосредственно с самим редактором Unity, необходимо было ознакомиться с интерфейсом и основными инструментами, необходимыми для разработки, т.к. изначально интерфейс мог казаться перегруженным различными панелями и кнопками.

Основные инструменты разработки которые следовало изучить, были панели инструментов (см. рисунок 2.2). :

- Панель **Hierarchy** (1) показывает все объекты, находящиеся в текущей сцене.
- Панель **Scene** (2) позволяет визуально редактировать сцену.
- Панель **Game** (3) используется для предпросмотра игры.

- Панель **Inspector** (4) позволяет настраивать свойства выбранного объекта.
- Панель **Project** (5) помогает управлять файлами и папками проекта
- Панель **Console** (6) выводит ошибки, предупреждения и сообщения отладки.

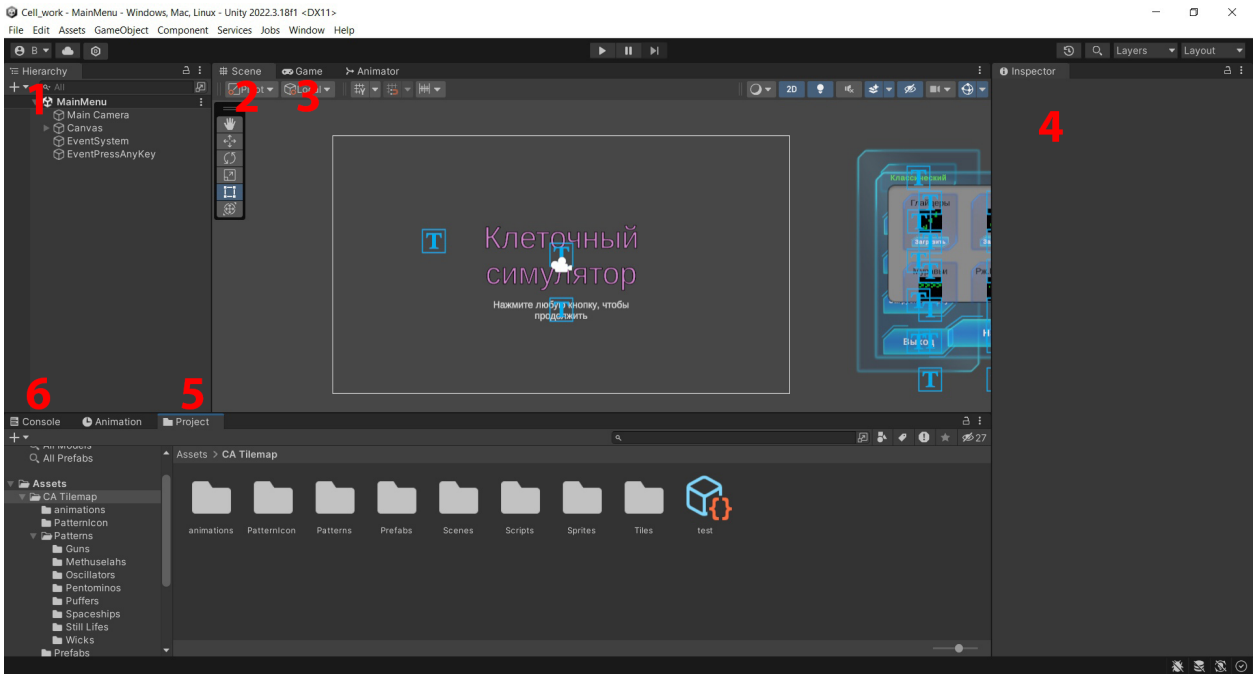

Рис. 2.1 Панели инструментов.

Далее появилась необходимость в изучении теории [10]. В Unity есть несколько ключевых понятий, которые важно понимать с самого начала, т.к. большинство форумов и книг используют их. К ним относятся такие понятия как:

- **GameObject** — это базовый объект в Unity, к которому можно прикреплять различные компоненты, такие как скрипты, физические свойства или визуальные элементы.
- **Component** — это то, что добавляется к GameObject для придания ему различных свойств и функциональностей.
- **Scene** — это пространство, в котором располагаются все GameObjects. Игра может состоять из нескольких сцен, например, меню, уровня и экрана победы.

- **Prefab** — это шаблон объекта, который можно многократно использовать в проекте.

После ознакомления с интерфейсом и основными понятиями, был сделан вывод, что рабочий стол организован и все необходимые инструменты настроены, что значительно упростило дальнейшую работу и позволило сосредоточиться на создании проекта.

2.2. Реализация основного алгоритма клеточного автомата.

После подготовки и установки окружения была выполнена реализация базового алгоритма клеточного автомата. Этот алгоритм представляет собой математическую модель, которая имитирует поведение клеток на двумерной решетке. Каждая клетка может быть либо «живой», либо «мертвой», и её состояние в следующем поколении зависит от состояния соседних клеток.

Вначале было проведено теоретическое изучение основных правил «Игры жизни» [11]. Эти правила включали в себя: смерть клетки от одиночества при наличии менее двух живых соседей, выживание при двух или трёх живых соседях, смерть от перенаселения при наличии более трёх живых соседей и возрождение мёртвой клетки при наличии ровно трёх живых соседей.

После этого началась разработка алгоритма. В Unity был создан скрипт на языке C#, который реализовывал логику обновления состояния клеток. Вместо использования классического двумерного массива, был использован массив, хранящий координаты клеток из **Tilemap** [12]. Это позволило учитывать специфические особенности Unity и обеспечить более точное взаимодействие с визуальной частью игры. В ходе разработки использовался скрипт **GameBoard.cs**(см. **ПРИЛОЖЕНИЕ** на стр. 22), в котором хранилась логика основной симуляции. В этом скрипте были В функции **UpdateState()**(см. **ПРИЛОЖЕНИЕ** на стр. 22 строки 118-164) в котором происходило обновление их состояния и визуализации текущего состояния на **Tilemap**, а функция **SetPattern()**(см. **ПРИЛОЖЕНИЕ** на стр. 22 строки 66-82) отвечала за начальную настройку массива клеток случайными значениями. В частности, использовались методы для определения состояния клетки и её соседей, та-

кие как **IsAlive()** и **CountNeighbors()**(см. **ПРИЛОЖЕНИЕ** на стр. 22 строки 210-213 и строки 166-198).

Tilemap в Unity оказался удобным инструментом для визуализации клеточного автомата. Это компонент, который позволяет легко создавать и редактировать двумерные сетки, используя тайлы (маленькие графические элементы). Для визуализации состояния клеток был создан новый **Tilemap** в сцене Unity. Этот компонент позволил упрощённое отображение и управление клетками на игровом поле.

Был написан скрипт **TilemapClickHandler.cs**(см. **ПРИЛОЖЕНИЕ** на стр. 25), который обеспечивал взаимодействие с клетками на уровне визуализации. Этот скрипт позволял пользователю кликать по клеткам, тем самым создавая их, что делало игровой опыт более интерактивным и удобным и в то же время демонстрировала работу алгоритма.

Во время тестирования и отладки были выявлены несколько проблем. Например, неправильное определение количества соседей у граничных клеток и ошибки при копировании массивов, что приводило к некорректному обновлению состояния клеток. Эти проблемы были исправлены в ходе тщательной отладки кода. Были также внедрены различные методы для проверки корректности работы алгоритма и его соответствия правилам «Игры жизни».

Для обеспечения плавной работы алгоритма на разных устройствах была проведена оптимизация кода. Основное внимание уделялось минимизации количества операций в основном цикле обновления, использованию эффективных методов для копирования и обновления массивов, а также оптимизации рендеринга клеток. В результате проведённых оптимизаций алгоритм стал работать более плавно и эффективно, даже при увеличении размера поля и частоты обновления.

Таким образом алгоритм клеточного автомата «Игры жизни» был реализован, протестирован и визуализирован с использованием Tilemap, что позволило перейти к следующим этапам разработки проекта.

2.3. Создание пользовательского интерфейса (UI)

После реализации и проверки основной механики игры, началось выполнение этапа разработки пользовательского интерфейса (UI) для игры «Игра жизни». Этот этап включал в себя создание интерактивных элементов, которые позволяли пользователям управлять симуляцией, редактировать, загружать и генерировать их, а также изменять параметры клеточного автомата. Основной задачей было сделать интерфейс интуитивно понятным и удобным для пользователя.

Для обеспечения удобства использования и отзывчивости интерфейса были изучены и применены практики UI/UX-дизайна [13], чтобы интерфейс был интуитивно понятным и легко читаемым. Особое внимание было уделено размеру и расположению кнопок, а также цветовой гамме интерфейса.

Для начала было решено создать основные элементы управления симуляцией. В Unity на главном экране игры было разработано основное меню, которое включало несколько ключевых кнопок:

- **Редактор** — переход в режим редактирования клеточного автомата..
- **Случайная генерация** — создание случайного начального состояния.
- **Загрузить карту** — загрузка предварительно сохраненных паттернов.
- **Выход** — завершение работы программы.
- **Помощь** — загрузка описания проекта и автора.

Эти элементы управления были реализованы с использованием стандартных UI-компонентов Unity, таких как Button и Panel.

Первым делом для удобства пользователей была создана отдельная панель, где можно было выбрать предопределенные паттерны клеток. На этой панели отображались кнопки с изображениями и названиями паттернов, таких как Глайдеры, Иви, Блинкер, Муравьи, Ружье Госпера и Вакуумная пушка [14]. Пользователь мог загрузить выбранный паттерн, нажав соответствующую кнопку «Загрузить». Эта функциональность была реализована с помощью скрипта **LevelTransition.cs** (см. **ПРИЛОЖЕНИЕ** на стр. 32)

Следом была создана отдельная панель, где пользователь мог задать размер карты и случайным образом сгенерировать начальное состояние клеток. На этой панели размещались кнопки для выбора размеров карты (20x20, 50x50, 100x100) и кнопка «Сгенерировать», которая инициализировала случайное распределение живых клеток на карте. Эта функциональность была реализована с помощью скрипта **RandomButton.cs**. (см. **ПРИЛОЖЕНИЕ** на стр. 31).

Для режима редактирования была создана отдельная сцена где пользователь мог задать размер карты перед началом редактирования и так же для интерактивного управления клетками была реализована возможность изменения состояния клеток с помощью кликов мыши. Эта функциональность была реализована с помощью скриптов **TilemapClickHandler.cs** и **EditorStartMap.cs**. (см. **ПРИЛОЖЕНИЕ** на стр. 28)

После разработки основных элементов интерфейса было проведено тщательное тестирование. Были выявлены и исправлены ошибки, связанные с отображением информации и взаимодействием с элементами управления. Тестирование проводилось на различных устройствах, чтобы убедиться в корректной работе интерфейса на всех поддерживаемых разрешениях.

2.4. Блок-схема.

Блок-схема «Игры жизни» Джона Конвея иллюстрирует процесс обновления состояния клеток на игровом поле. Алгоритм основан на простых правилах, которые определяют, будет ли клетка «живой» или «мертвой» в следующем поколении, в зависимости от состояния её соседей. Данная блок-схема показывает основные этапы работы алгоритма, начиная с инициализации и заканчивая обновлением состояния клеток (см. рисунок 2.2).

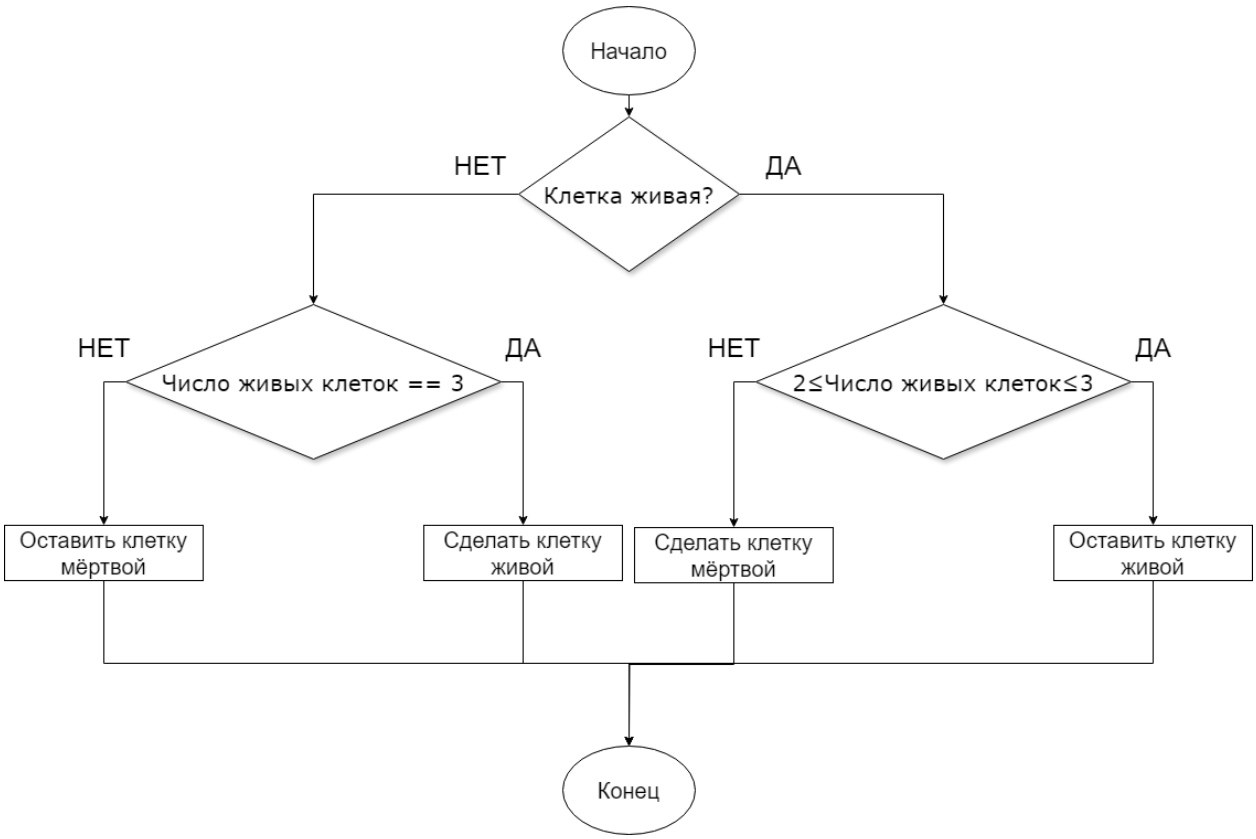

Рис. 2.2 Блок схема алгоритма «Игры жизни».

2.5. Сборка программы

Сборка программы является предфинальным этапом разработки проекта в Unity. На этом этапе проект подготавливается к запуску на различных платформах, таких как Windows, macOS, Linux, iOS и других. В данном разделе описаны шаги по сборке программы, начиная с настройки проекта и заканчивая созданием исполняемых файлов.

Перед сборкой программы необходимо было убедиться, что все параметры проекта настроены правильно. Для этого в Unity используется меню «File > Build Settings...». В открывшемся окне можно выбрать платформу, на которую будет производиться сборка, а также настроить основные параметры.

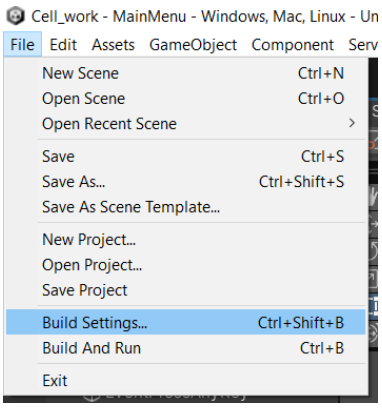

Рис. 2.3 Открытие меню Build Settings.

В окне «Build Settings» выберется платформа, на которую будет производиться сборка (например, PC, Mac & Linux Standalone для Windows и macOS, или Android для мобильных устройств). Были добавлены все необходимые сцены в раздел «Scenes In Build», нажав на кнопку «Add Open Scenes». Необходимо было убедитесь, что все сцены, которые должны быть включены в сборку, добавлены в этот список.

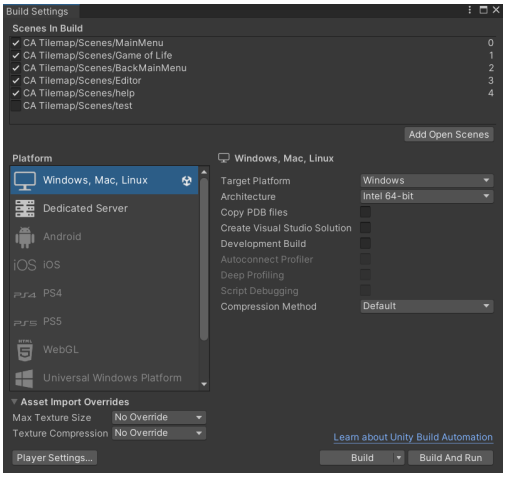

Рис. 2.4 Окно Build Settings.

В зависимости от выбранной платформы могут потребоваться дополнительные настройки. Для Windows и macOS также можно было настроить параметры отображения, значки и другие параметры.

После настройки всех параметров нажав на кнопку «Build», Unity предложило выбрать папку, в которую будет сохранен скомпилированный проект.Выбрав подходящее место, начался процесс сборки.

Скомпилированным проект в папке будет иметь такой вид (см. рисунок 2.5).

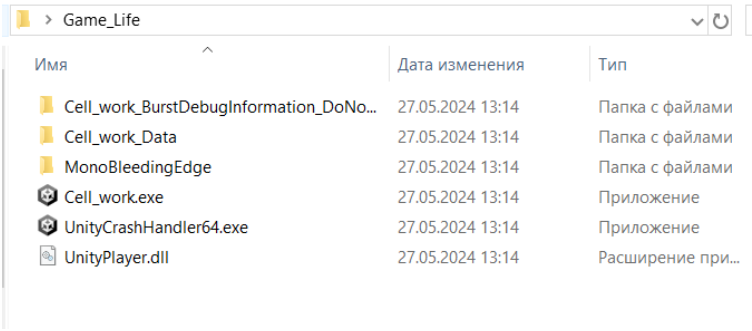

Рис. 2.5 Директория с исполняемым файлом.

2.6. Тестирование программы

После завершения сборки в конце необходимо было протестировать скомпилированный проект на целевой платформе. Запустить созданный исполняемые файлы на соответствующей операционной системе и следовало убедиться, что программа работает корректно. Проверив все основные функции и элементы интерфейса, убедиться в отсутствии ошибок.

При запуске игры нас встречает главное меню (см. **ПРИЛОЖЕНИЕ** на стр. 33).

При нажатии на кнопку «Редактор» меняется экран. На нём отображена возможность редактирования игрового поля и задания размеров карты (см. **ПРИЛОЖЕНИЕ** на стр. 34).

При нажатии на кнопку «Случайная Генерация» появляется панель. На ней отображены настройки размера карты (см. **ПРИЛОЖЕНИЕ** на стр. 35).

При нажатии на кнопку «Загрузить карту» появляется панель. На ней отображены кнопки выбора предопределенных паттернов и загрузки карты (см. **ПРИЛОЖЕНИЕ** на стр. 35).

Основное игровое поле выглядит следующим образом (см. **ПРИЛОЖЕНИЕ** на стр. 36).

Пример ситуации, когда запускается симуляция «Игры жизни», некоторые клетки появляются, исчезают или остаются неизменными в зависимости от правил, приведены ниже (см. **ПРИЛОЖЕНИЕ** на стр. 36).

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была создана кроссплатформенная программа с использованием языка программирования C# и платформы Unity.

Программа создавалась в объектно-ориентированной парадигме.

Программа представляет собой симулятор клеточного автомата «Игра жизни».

Кнопки, экраны «Помощь», «Редактор», «Случайная генерация», «Загрузить карту», «Выход» – все эти объекты были добавлены в программу. Кнопки срабатывают при нажатии на них, при этом меняется экран. Выбор размеров карты и типов начальных паттернов успешно выполняется. Симуляция запускается соответствующими кнопками. Все взаимодействия между пользователем и игровым полем успешно отрабатываются.

Программы выполнена в соответствии с **Техническим заданием** на странице 5.

Были решены все поставленные задачи (см. **Задачи** на стр. 4).

Разработка программы и отчёта производилась с использованием системы контроля версий Git [15], а конкретнее с помощью онлайн-хранилища GitHub.

По коду программы была создана блок-схема (см. на стр. 17.

Была успешна выполнена сборка игры под операционную систему Microsoft Windows (см. **Сборка программы** на стр. 18).

Проверка работоспособности программы была проведена успешно (см. **Проверка работоспособности программы** на стр. **??**).

Отчёт оформлен с помощью системы компьютерной вёрстки TEX и его расширения XETEX из дистрибутива *TeX Live*.

Ссылка на репозиторий проекта: https://github.com/KapKanZhan/Game-Life-Coursework.git

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Библия C#. 6-е издание Фленов Михаил Евгеньевич, свободный. - Загл. с экрана. – Яз. англ. — (Дата обр. 03.06.2024).
2. Джозеф Хокинг: Unity в действии. Мультиплатформенная разработка на C#, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 03.06.2024).
3. Паттерны, игра ”Жизнь” [Электронный ресурс] Википедия — свободная энциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/Долгожитель (конфигурация клеточного автомата), свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 14.06.2024).
4. Adobe Photoshop [Электронный ресурс] Добро пожаловать в Руководство пользователя Photoshop. Режим доступа: https://helpx.adobe.com/ru/photoshop/user-guide.html, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 23.06.2024).
5. GitHub [Электронный ресурс] Википедия — свободная энциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/GitHub, свободный. - Загл. с экрана. – Яз. рус., англ. — (Дата обр. 22.06.2024).
6. Конвей, Джон Хортон [Электронный ресурс] Википедия — свободная энциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/Конвей, Джон Хортон, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 14.06.2024).
7. Unity [Электронный ресурс] Что такое Unity Hub. Режим доступа: https://unity.com/ru/unity-hub, свободный. - Загл. с экрана. – Яз. рус., англ. — (Дата обр. 06.06.2024).
8. Visual Studio Code [Электронный ресурс] Microsoft Lern. Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022, свободный. - Загл. с экрана. – Яз. англ. — (Дата обр. 06.06.2024).

9. Game dev. with Unity [Электронный ресурс] Создание игр Unity с помощью Visual Studio. Режим доступа: https://visualstudio.microsoft.com/ru/vs/unity-tools/, свободный. - Загл. с экрана. – Яз. рус., англ. — (Дата обр. 22.06.2024).
10. Основы Unity [Электронный ресурс] Unity — руководство. Режим доступа: https://docs.unity3d.com/ru/530/Manual/UnityBasics.html, свободный. - Загл. с экрана. – Яз. рус., англ. — (Дата обр. 06.06.2024).
11. Игра ”Жизнь” [Электронный ресурс] Википедия — свободная энциклопедия: https://ru.wikipedia.org/wiki/Игра «Жизнь», свободный. - Загл. с экрана. – Яз. рус., англ. — (Дата обр. 03.06.2024).
12. Tilemap [Электронный ресурс] Введение в систему тайловых карт Unity. Режим доступа: https://habr.com/ru/articles/412765/, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 06.06.2024).
13. UI/Ux-дизайн [Электронный ресурс] Яндекс практикум. Режим доступа: https://practicum.yandex.ru/blog/chto-takoe-ux-ui-dizayn/, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 03.06.2024).
14. Паттерны [Электронный ресурс] Конфигурации Игры «Жизнь». Режим доступа: https://life.written.ru/game_of_life_review_by_gardner, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 23.06.2024).
15. Что такое Git? [Электронный ресурс] Atlassian. Режим доступа: https://www.atlassian.com/ru/git/tutorials/what-is-git, свободный. - Загл. с экрана. – Яз. рус. — (Дата обр. 23.06.2024).

ПРИЛОЖЕНИЕ

Текст программы

Основной скрипт инициализации карты и алгоритма игры

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;

public class GameBoard : MonoBehaviour
{
    [SerializeField] private Tilemap currentState;
    [SerializeField] private Tilemap nextState;
    //[SerializeField] private Tilemap BorderTilemap; // □□□□□□ □□ Tilemap □□□ □□□□
    //[SerializeField] private Tile borderTile; // □□□□□□ □□ □□□□ □□□ □□□□□□□
    [SerializeField] private Tile aliveTile;
    [SerializeField] private Tile deadTile;
    [SerializeField] private Pattern pattern;
    //[SerializeField] private Pattern pattern;
    [SerializeField] private float updateInterval = 0.05f;

    public bool test = false;

    public HashSet<Vector3Int> aliveCells;
    private HashSet<Vector3Int> cellsToCheck;

    public int population { get; private set; }
    public int iterations { get; private set; }
    public float time { get; private set; }

    BorderDraw border;



    private void Awake()
    {
        aliveCells = new HashSet<Vector3Int>();
        cellsToCheck = new HashSet<Vector3Int>();
        border = GetComponent<BorderDraw>();
    }

    private void Start()
    {
        if (!test)
        {
            if (!EditorStartMap.EditorActive)
            {
                if (!RandomButton.RandomActive)
                {
                    pattern = LevelTransition.MyPattern;
                }
                else
                {
                    pattern = RandomButton.MyPattern;
                }
            }
            else
            {
                pattern = EditorStartMap.MyPattern;
            }

        }

        SetPattern(pattern);
        //DrawBorders();
    }

```

```
64
65  private void SetPattern(Pattern pattern)
66  {
67      Clear(); // □□□□□□□□ □□□□□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□
68
69      Vector2Int center = pattern.GetCenter(); // □□□□□□□□□□ □□□□□□□ □□□□□□□□□ □□□□ □□□□□□□□□□□□ □□□□□□□□□□□ □□ □□□□□□□□ □□□□
70
71      for (int i = 0; i < pattern.cells.Length; i++) // □□□□□□□□ □□□□□ □□□□□□□ □ □□□□□□□□□
72      {
73          Vector3Int cell = (Vector3Int)(pattern.cells[i]); // □□□□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□ □□ □□□□□□□□□ □ □□□□□□□□ □□ □□□□□□□□ □□□□□
74          currentState.SetTile(cell, aliveTile); // □□□□□□□□□□ □□□□□□ □□□□ □□□□□□ □□□□□□□ □□ □□□□□□□□ Tilemap
75          aliveCells.Add(cell); // □□□□□□□□□□□□ □□□□□□□ □ □□□□□□□ □□□□□□ □□□□□□□
76
77
78      }
79
80      population = aliveCells.Count; // □□□□□□□□□□□□ □□□□□□ □□□□□□ □□□□□□□ □ □□□□□□□□□□
81  }
82
83  private void Clear()
84  {
85      aliveCells.Clear(); // □□□□□□□□ □□□□□□□ □□□□□□ □□□□□□□
86      cellsToCheck.Clear(); // □□□□□□□□ □□□□□□□ □□□□□□□ □□□□ □□□□□□□□□
87      currentState.ClearAllTiles(); // □□□□□□□□ □□□□□ □□□□□□□ □□ □□□□□□□□ Tilemap
88      nextState.ClearAllTiles(); // □□□□□□□□ □□□□□ □□□□□□□ □□ □□□□□□□□□□ Tilemap
89      population = 0; // □□□□□□ □□□□□□□□□□□□ □□□□□□ □□□□□□□
90      iterations = 0; // □□□□□□ □□□□□□□□□□□□ □□□□□□□□□
91      time = 0f; // □□□□□□ □□□□□□□□ □□□□□□□□□□
92  }
93
94  private void OnEnable()
95  {
96      StartCoroutine(Simulate());
97  }
98
99  private IEnumerator Simulate()
100 {
101     var interval = new WaitForSeconds(updateInterval); // □□□□□□□□□ □□□□□□ □□□□□□□ □□□□□□ □□□□□□□□□□□
102     yield return interval; // □□□□□□□ □□□□□□ □□□□□□ □□□□□□□□ □□□□□□
103
104     while (enabled) // □□□□□ □□□□□□□□□□ □□□□□□□□, □□□□□□□□□□ □□□□□□□□□□
105     {
106         UpdateState(); // □□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□ □□□□□
107
108         population = aliveCells.Count; // □□□□□□□□□□□ □□□□□□□□□□□ □□□□□□ □□□□□□□
109         iterations++; // □□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□
110         time += updateInterval; // □□□□□□□□□□□ □□□□□□□ □□□□□□□□ □□□□□□□□□□
111
112         yield return interval; // □□□□□□□□□ □□ □□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□□
113     }
114 }
115
116 private void UpdateState()
117 {
118     cellsToCheck.Clear(); // □□□□□□□□ □□□□□□□ □□□□□□□ □□□□ □□□□□□□□□
119
120     // □□□□□□□□□ □□□□ □□□□□□□, □□□□□□□□ □□□□□□ □□□□□□□□□□
121     foreach (Vector3Int cell in aliveCells)
122     {
123         //Debug.Log(cell);
124         for (int x = -1; x <= 1; x++)
125         {
126             for (int y = -1; y <= 1; y++)
127             {
128                 cellsToCheck.Add(cell + new Vector3Int(x, y));
129             }
130         }
131     }
132
133     // □□□□□□□□ □□□□□□□ □ □□□□□□□□□□ □□□□□□□□□□
134     foreach (Vector3Int cell in cellsToCheck)
135     {
136
```

```
137         int neighbors = CountNeighbors(cell); // □□□□□□□□ □□□□□□□□ □□□□□□□
138
139         bool alive = IsAlive(cell); // □□□□□□□□□, □□□□ □□ □□□□□□□
140
141
142         if (!alive && neighbors == 3)
143         {
144             nextState.SetTile(cell, aliveTile);// □□□□□□□□□□ □□□□□□□, □□□□ □□□□□ 3 □□□□□ □□□□□□□
145             aliveCells.Add(cell); //□□□□□□□□□□ □□□□□ □□□□□ □□□□□□□
146         }
147         else if (alive && (neighbors < 2 || neighbors > 3))
148         {
149             nextState.SetTile(cell, deadTile); // □□□□□□□□□ □□□□□□□ □□-□□ □□□□□□□□□ □□□ □□□□□□□□□□□□□□
150             aliveCells.Remove(cell); //□□□□□□□□□ □□□□□ □□□□□□□
151         }
152         else // □□□□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□, □□□□ □□□□□□□□□□ □□ □□□□□□□□□□
153         {
154             nextState.SetTile(cell, currentState.GetTile(cell));
155         }
156     }
157
158     // □□□□□□ □□□□□□□□□ □ □□□□□□□□□□□□ □□□□□□□□□□
159     Tilemap temp = currentState;
160     currentState = nextState;
161     nextState = temp;
162     nextState.ClearAllTiles();
163 }
164
165 private int CountNeighbors(Vector3Int cell)
166 {
167     int count = 0;
168     int minCoordX = -1 * (border.width - 2);// □□□□□□□□□□□□ □□□□□□□□□□□
169     int maxCoordX = border.width - 2;// □□□□□□□□□□□□□□ □□□□□□□□□□□
170     int minCoordY = -1 * (border.height - 2);// □□□□□□□□□□□□ □□□□□□□□□□□
171     int maxCoordY = border.height - 2;// □□□□□□□□□□□□□□ □□□□□□□□□□□
172
173
174     for (int x = -1; x <= 1; x++)
175     {
176         for (int y = -1; y <= 1; y++)
177         {
178             if (x == 0 && y == 0)
179                 continue; // □□□□□□□□□□□ □□□□□ □□□□□□□
180
181             int neighborX = cell.x + x;
182             int neighborY = cell.y + y;
183
184             if (neighborX >= minCoordX && neighborX <= maxCoordX && neighborY >= minCoordY && neighborY <= maxCoordY)
185             {
186                 Vector3Int neighbor = new Vector3Int(neighborX, neighborY, cell.z);
187                 if (IsAlive(neighbor))
188                 {
189                     count++;
190                 }
191             }
192
193         }
194     }
195
196     return count;
197 }
198
199 // □□□□□□□□□□□ □□□□□□□□ □□□□ □□□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□, □□□□□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□
200 private int Mod(int x, int m)
201 {
202     int r = x % m;
203     return r < 0 ? r + m : r;
204 }
205
206
207
208
209 private bool IsAlive(Vector3Int cell)
```

```
210     {
211         return currentState.GetTile(cell) == aliveTile;
212     }
213 
214 }
```

Скрипт создания клеток по клику мыши

```
1   using System.Collections.Generic;
2   using UnityEngine;
3   using UnityEngine.Tilemaps;
4   
5   public class TilemapClickHandler : MonoBehaviour
6   {
7       public Tilemap tilemap;
8       public Tile replacementTile;
9       public int width = 20; // □□□□□□ □□□□
10      public int height = 20; // □□□□□□ □□□□
11      public GameObject EditStart;
12  
13      private int tileX;
14      private int tileY;
15  
16  
17  
18      void Update()
19      {
20          if (Input.GetMouseButtonDown(0))
21          {
22              Vector3 mouseWorldPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
23              Vector3Int cellPosition = tilemap.WorldToCell(mouseWorldPos);
24  
25              if (IsWithinBounds(cellPosition))
26              {
27                  ReplaceTile(cellPosition);
28              }
29          }
30      }
31  
32      bool IsWithinBounds(Vector3Int cellPosition)
33      {
34          int halfWidth = width / 2;
35          int halfHeight = height / 2;
36          return cellPosition.x >= -halfWidth+1 && cellPosition.x <= halfWidth-1 && cellPosition.y >= -halfHeight+1 && cellPosition.y <= halfHeight-1;
37      }
38  
39      public void ReplaceTile(Vector3Int cellPosition)
40      {
41          tilemap.SetTile(cellPosition, replacementTile);
42          tileX = cellPosition.x;
43          tileY = cellPosition.y;
44          EditStart.GetComponent<EditorStartMap>().pattern.AddCell(tileX, tileY);
45          Debug.Log("Tile replaced at: " + cellPosition);
46          Debug.Log("Tile coordinates: x = " + tileX + ", y = " + tileY);
47      }
48  }
```

Скрипт возвращения в главное меню

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5   
6   public class BackMainMenu : MonoBehaviour
7   {
8       public int scene;
9   
10      public void changeScene()
11      {
```

```
12        RandomButton.RandomActive = false;
13        SceneManager.LoadScene(scene);
14    }
15 }
```

Скрипт возвращения в главное меню

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class BackMainMenu : MonoBehaviour
7  {
8     public int scene;
9
10    public void changeScene()
11    {
12       RandomButton.RandomActive = false;
13       SceneManager.LoadScene(scene);
14    }
15 }
```

Скрипт отрисовки границ карты

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Tilemaps;
5
6  public class BorderDraw : MonoBehaviour
7  {
8
9     [SerializeField] private Tilemap BorderTilemap; // □□□□□□ □□ Tilemap □□□ □□□□
10    [SerializeField] private Tile borderTile; // □□□□□□ □□ □□□□ □□□ □□□□□□□
11    [SerializeField] private Tilemap CellTilemap; // □□□□□□ □□ Tilemap □□□ □□□□
12    public int width = 20; // □□□□□□
13    public int height = 20; // □□□□□□
14    public bool test = false;
15    public bool editing = EditorButton.EditorActive;
16    public GameObject ClickSize;
17    // Start is called before the first frame update
18    void Start()
19    {
20       if (!test)
21       {
22          if (EditorStartMap.EditorActive)
23          {
24             width = EditorStartMap.MyPattern.width;
25             height = EditorStartMap.MyPattern.height;
26          }
27          else if(LevelTransition.LevelActive)
28          {
29             width = LevelTransition.MyPattern.width;
30             height = LevelTransition.MyPattern.height;
31          }
32          else if(RandomButton.RandomActive)
33          {
34             width = RandomButton.MyPattern.width;
35             height = RandomButton.MyPattern.height;
36          }
37          else
38          {
39             width = 20;
40             height = 20;
41          }
42       }
43
44
45
```

```
46      width = (width) / 2;
47      height = (height) / 2;
48      DrawBorders();
49  }
50
51  private void DrawBorders()
52  {
53      // □□□□□□□□□□□□ □ □□□□□□□□□□□□□ □□□□□□□□□□□ □□ □□□□ X □ Y
54      int minX = -1 * width; // □□□□□□□□□□□□ □□□□□□□□□□□ □□ X
55      int maxX = width; // □□□□□□□□□□□□□ □□□□□□□□□□□ □□ X
56      int minY = -1 * height; // □□□□□□□□□□□□ □□□□□□□□□□□ □□ Y (□□□□□□□ □□□ □□□□□□□□□□□□□□□□)
57      int maxY = height; // □□□□□□□□□□□□□ □□□□□□□□□□□ □□ Y (□□□□□□□ □□□ □□□□□□□□□□□□□□□□)
58
59      // □□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□ □□□□□□□□
60      for (int x = minX; x <= maxX; x++)
61      {
62          BorderTilemap.SetTile(new Vector3Int(x, minY, 0), borderTile); // □□□□□□□ □□□□□□□□
63          BorderTilemap.SetTile(new Vector3Int(x, maxY, 0), borderTile); // □□□□□□□□ □□□□□□□□
64      }
65
66      // □□□□□□□ □□□□□□□□□□□□□□ □□□□□□ □□□□□□□□
67      for (int y = minY; y <= maxY; y++)
68      {
69          BorderTilemap.SetTile(new Vector3Int(minX, y, 0), borderTile); // □□□□□□ □□□□□□□□
70          BorderTilemap.SetTile(new Vector3Int(maxX, y, 0), borderTile); // □□□□□□□ □□□□□□□□
71      }
72  }
73
74  private void ClearBorders()
75  {
76      BorderTilemap.ClearAllTiles();
77      CellTilemap.ClearAllTiles();
78  }
79
80  public void SetSize20x20()
81  {
82      ClearBorders();
83      width = (20) / 2;
84      height = (20) / 2;
85      gameObject.GetComponent<TilemapClickHandler>().width = 20;
86      gameObject.GetComponent<TilemapClickHandler>().height = 20;
87      DrawBorders();
88  }
89
90  public void SetSize50x50()
91  {
92      ClearBorders();
93      width = (50) / 2;
94      height = (50) / 2;
95      gameObject.GetComponent<TilemapClickHandler>().width = 50;
96      gameObject.GetComponent<TilemapClickHandler>().height = 50;
97      DrawBorders();
98  }
99
100 public void SetSize100x100()
101 {
102     ClearBorders();
103     width = (100) / 2;
104     height = (100) / 2;
105     gameObject.GetComponent<TilemapClickHandler>().width = 100;
106     gameObject.GetComponent<TilemapClickHandler>().height = 100;
107     DrawBorders();
108 }
109
110 }
```

Скрипт динамической камеры

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Tilemaps;
```

```
5
6  public class CameraAdjuster : MonoBehaviour
7  {
8      public Tilemap map; // □□□□□□ □□ Tilemap
9      BorderDraw border;
10     public float minCameraSize = 30f; // □□□□□□□□□□□□ □□□□□□ □□□□□□
11     private void Awake()
12     {
13         border = GetComponent<BorderDraw>();
14     }
15
16     void Start()
17     {
18         AdjustCamera();
19     }
20
21     public void AdjustCamera()
22     {
23         Camera cam = Camera.main;
24
25         float tileHeight = map.cellSize.y; // □□□□□□ □□□□□
26         float requiredHeight = (border.height*2) * tileHeight; // □□□□□□□□□□□ □□□□□□□ □□□□□□
27         //cam.orthographicSize = requiredHeight / 2; // □□□□□□□□□ orthographicSize
28
29         float requiredWidth = (border.width*2) * map.cellSize.x; // □□□□□□□□□□□ □□□□□□□ □□□□□□
30         float screenWidth = requiredWidth / (2 * cam.aspect); // □□□□□□□□□□□□ □□□□□□ □□□□□□ □□□□□□ □□□□□□ □□ □□□□□□
31         //cam.orthographicSize = Mathf.Max(cam.orthographicSize, screenWidth);
32         cam.orthographicSize = Mathf.Max(minCameraSize, Mathf.Max(requiredHeight, screenWidth));
33         // □□□□□□□□□□□□□□□□□ □□□□□□ □□ □□□□□□ Tilemap
34         cam.transform.position = new Vector3(0,
35         0,
36         -10);
37     }
38 }
```

Скрипт перехода на сцену редактирования

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class EditorButton : MonoBehaviour
7  {
8      private int scene = 3;
9      public static bool EditorActive = false;
10     // Start is called before the first frame update
11     public void changeScene()
12     {
13         EditorActive = true;
14         SceneManager.LoadScene(scene);
15     }
16 }
```

Скрипт запуска созданной карты в режиме редактирования

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using UnityEngine.UI;
6  using TMPro;
7
8  public class EditorStartMap : MonoBehaviour
9  {
10
11     private int scene = 1;
12     public static bool EditorActive = false;
13     public int width = 20;
14     public int height = 20;
```

```
    public int quantityfCells = 190;
    [SerializeField] public Pattern pattern;
    public static Pattern MyPattern;
    public TextMeshProUGUI SizeText;

    void Awake()
    {
        EditorActive = false;
    }

    void Start()
    {

        pattern.cells = new Vector2Int[0];
    }


    public void changeScene()
    {
        EditorActive = true;
        pattern.width = width;
        pattern.height = height;
        MyPattern = pattern;
        SceneManager.LoadScene(scene);
    }

    public void changeSize20x20()
    {
        pattern.cells = new Vector2Int[0];
        width = 20;
        height = 20;
        SizeText.text = "20x20";
    }

    public void changeSize50x50()
    {
        pattern.cells = new Vector2Int[0];
        width = 50;
        height = 50;
        SizeText.text = "50x50";
    }

    public void changeSize100x100()
    {
        pattern.cells = new Vector2Int[0];
        width = 100;
        height = 100;
        SizeText.text = "100x100";
    }
}

```

Скрипт выхода из игры

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ExitButton : MonoBehaviour
{
    // Start is called before the first frame update
    public void Exit()
    {
        Application.Quit();
    }
}
```

Скрипт создания паттерна

```
using UnityEngine;

[CreateAssetMenu(menuName = "Game of Life/Pattern")]
public class Pattern : ScriptableObject
{
    public int width = 20;
    public int height = 20;

    public Vector2Int[] cells;

    public Vector2Int GetCenter()
    {
        if (cells == null || cells.Length == 0) {
            return Vector2Int.zero;
        }

        Vector2Int min = Vector2Int.zero;
        Vector2Int max = Vector2Int.zero;

        for (int i = 0; i < cells.Length; i++)
        {
            Vector2Int cell = cells[i];
            min.x = Mathf.Min(min.x, cell.x);
            min.y = Mathf.Min(min.y, cell.y);
            max.x = Mathf.Max(max.x, cell.x);
            max.y = Mathf.Max(max.y, cell.y);
        }

        return (min + max) / 2;
    }

    public void GenerateRandomPattern(int numberOfCells, int minX, int maxX, int minY, int maxY)
    {
        cells = new Vector2Int[numberOfCells];
        for (int i = 0; i < numberOfCells; i++)
        {
            int x = Random.Range(minX, maxX + 1);
            int y = Random.Range(minY, maxY + 1);
            cells[i] = new Vector2Int(x, y);
        }
    }

    public void AddCell(int x, int y)
    {
        Vector2Int newCell = new Vector2Int(x, y);
        if (cells == null)
        {
            cells = new Vector2Int[] { newCell };
        }
        else
        {
            Vector2Int[] newCells = new Vector2Int[cells.Length + 1];
            for (int i = 0; i < cells.Length; i++)
            {
                newCells[i] = cells[i];
            }
            newCells[cells.Length] = newCell;
            cells = newCells;
        }
    }

}
```

Скрипт выключения заставки

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PressAnyKey : MonoBehaviour
{
```

```
7
8     private Animator animPanel;
9     private Animator animMainTitle;
10    public GameObject PanelMenu;
11    public GameObject StartTitle;
12    public GameObject MainTitle;
13
14    bool Clck = false;
15
16    // Start is called before the first frame update
17    void Start()
18    {
19        animPanel = PanelMenu.GetComponent<Animator>();
20        animMainTitle = MainTitle.GetComponent<Animator>();
21    }
22
23    // Update is called once per frame
24    void Update()
25    {
26        if (Input.anyKey && Clck == false)
27        {
28            StartTitle.SetActive(false);
29            Clck = true;
30            animMainTitle.SetTrigger("IntroStart");
31
32            animPanel.SetTrigger("StartMenu");
33        }
34    }
35 }
```

Скрипт выключения заставки по нажатию любой кнопки

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PressAnyKey : MonoBehaviour
6  {
7
8      private Animator animPanel;
9      private Animator animMainTitle;
10     public GameObject PanelMenu;
11     public GameObject StartTitle;
12     public GameObject MainTitle;
13
14     bool Clck = false;
15
16     // Start is called before the first frame update
17     void Start()
18     {
19         animPanel = PanelMenu.GetComponent<Animator>();
20         animMainTitle = MainTitle.GetComponent<Animator>();
21     }
22
23     // Update is called once per frame
24     void Update()
25     {
26         if (Input.anyKey && Clck == false)
27         {
28             StartTitle.SetActive(false);
29             Clck = true;
30             animMainTitle.SetTrigger("IntroStart");
31
32             animPanel.SetTrigger("StartMenu");
33         }
34     }
35 }
```

Скрипт генерации и запуска карты в режиме Случайная Генерация

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;

public class RandomButton : MonoBehaviour
{
    private int scene = 1;
    public static bool RandomActive = false;
    public int width = 20;
    public int height = 20;
    public int quantityfCells = 190;
    [SerializeField] private Pattern pattern;
    public static Pattern MyPattern;
    public TextMeshProUGUI SizeText;

    void Awake()
    {
        RandomActive = false;
    }

    public void changeScene()
    {
        pattern.cells = new Vector2Int[0];
        RandomActive = true;
        pattern.width = width;
        pattern.height = height;
        pattern.GenerateRandomPattern(quantityfCells, (-1 * (width/2 - 1)), width/2 - 1, (-1 * (height/2 - 1)), height/2 - 1);
        MyPattern = pattern;
        SceneManager.LoadScene(scene);
    }

    public void changeSize20x20()
    {
        width = 20;
        height = 20;
        quantityfCells = 45;
        SizeText.text = "20x20";
    }

    public void changeSize50x50()
    {
        width = 50;
        height = 50;
        quantityfCells = 260;
        SizeText.text = "50x50";
    }

    public void changeSize100x100()
    {
        width = 100;
        height = 100;
        quantityfCells = 1100;
        SizeText.text = "100x100";
    }
}
```

Скрипт готового паттерна в режиме Загрузить Карту

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelTransition : MonoBehaviour
{
    private int scene = 1;
    [SerializeField] private Pattern pattern;
    public static Pattern MyPattern;
```

```
11    public static bool LevelActive = false;
12
13    void Awake()
14    {
15        LevelActive = false;
16    }
17
18    public void changeScene()
19    {
20        LevelActive = true;
21        MyPattern = pattern;
22        SceneManager.LoadScene(scene);
23    }
24 }
```

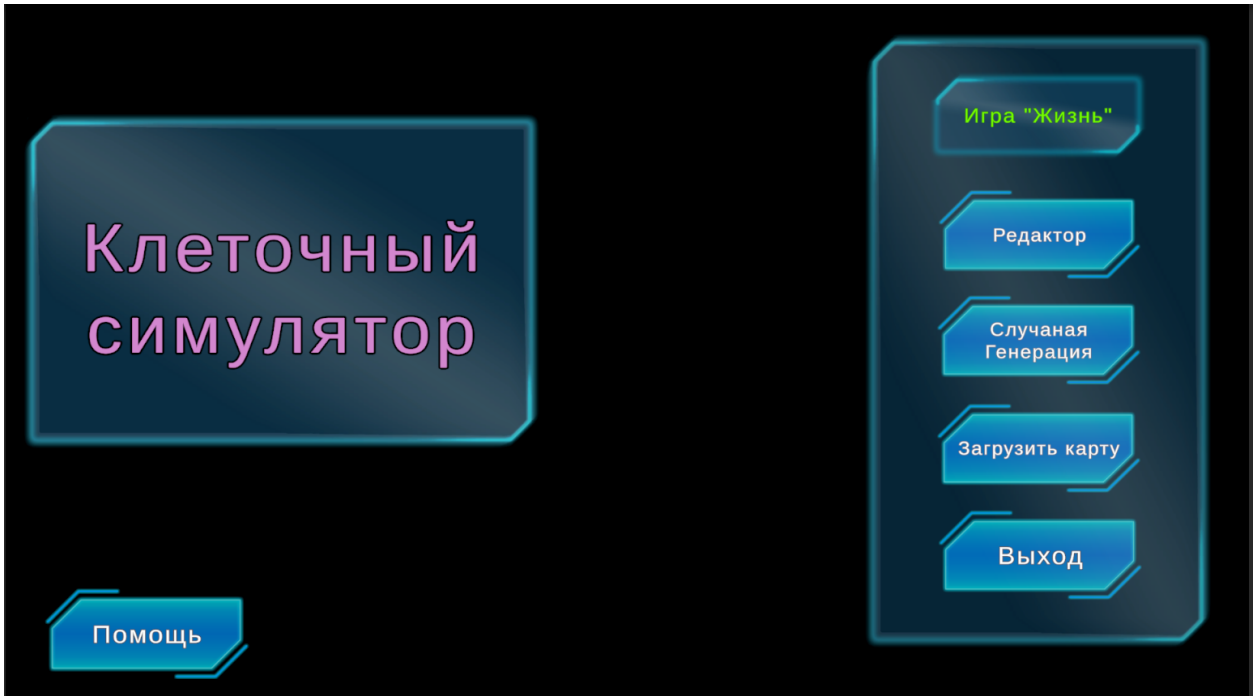

Рис. 6 Главное меню «Игры жизнь».

Рис. 7 Редактор.

Рис. 8 Окно редактора.

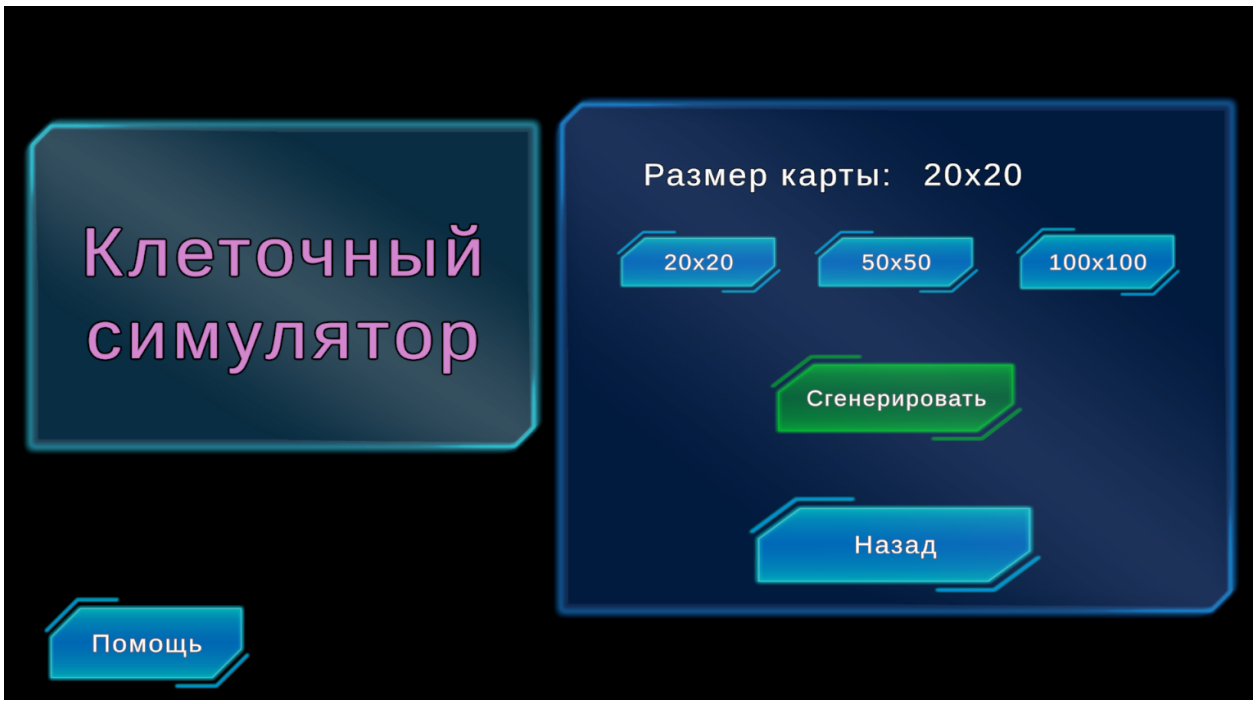

Рис. 9 Панель Случайной генерации.

Рис. 10 Панель Загрузки карт.

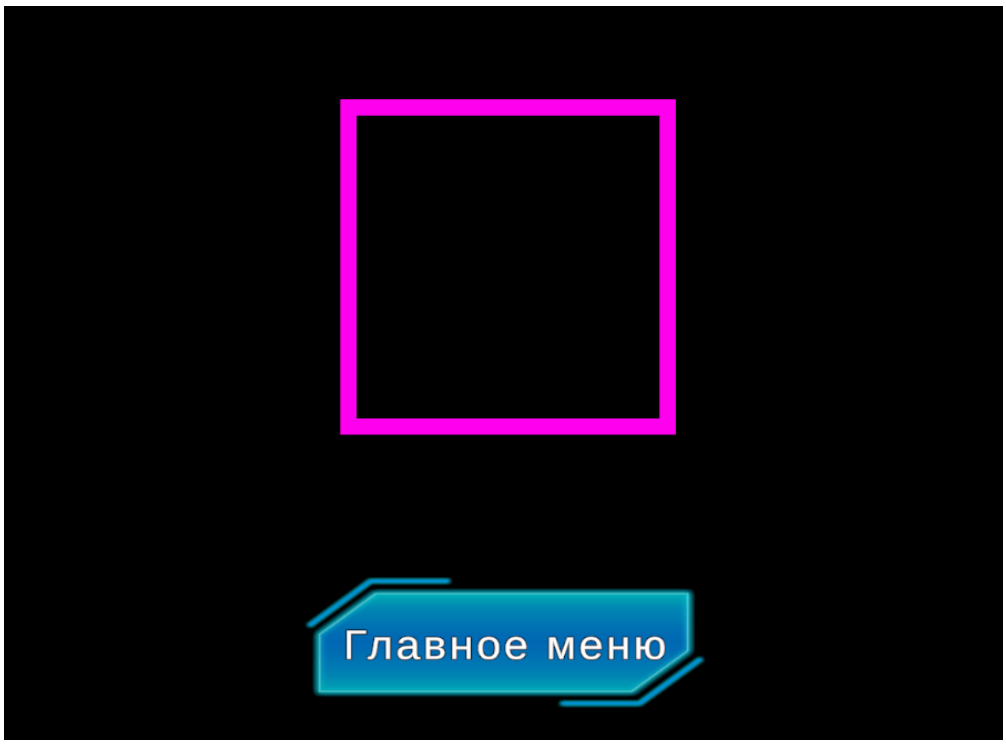

Рис. 11 Игровое поле.

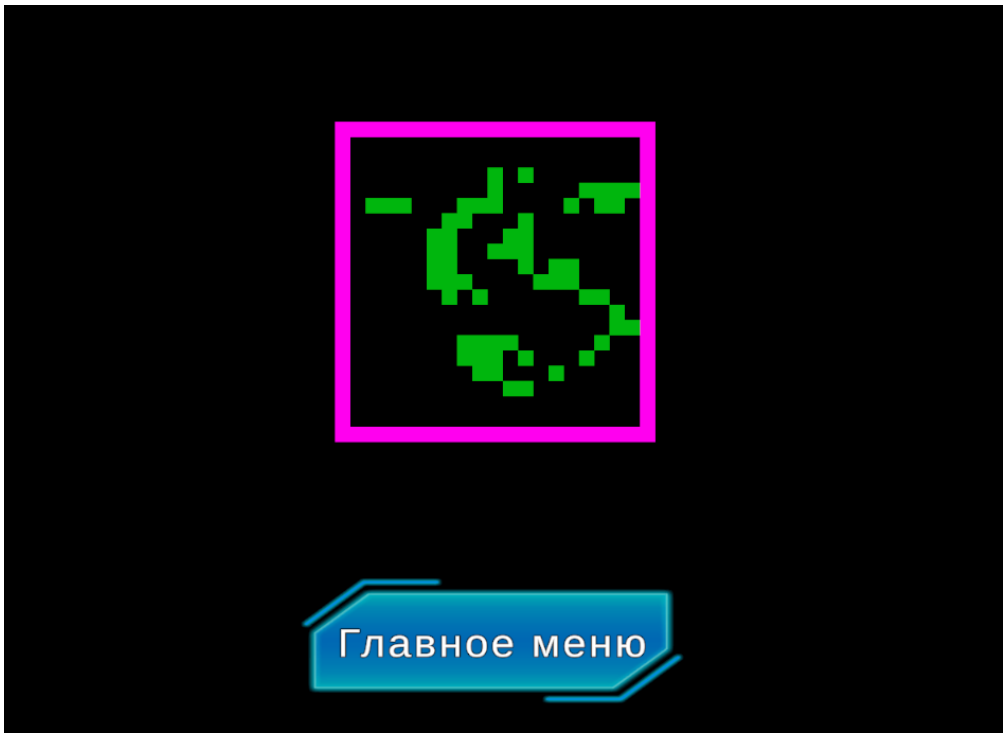

Рис. 12 Симуляция.