Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет к[омпьютерных систем](https://www.bsuir.by/ru/fksis) и сетей

Кафедра информатики

Дисциплина «Основы алгоритмизации и программирования»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  Ассистент кафедры Информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_ И. С. Чайкин |
|  | \_\_\_.\_\_\_\_.2024 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему:

**«РаБОТА АЛГОРИТМОВ генерации КАРТ В ИГРАХ НА ЯЗЫКе C++»**

БГУИР КР 6-05-0612-02 18 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 353501  Слюсарь Станислав Юрьевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовая работа представлена на проверку \_\_\_.\_\_\_\_.2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 3](#__RefHeading___Toc15216_3037588987)

[1 Язык программирования *C++* и сферы его применения 4](#__RefHeading___Toc15218_3037588987)

[1.1 Язык *С++* и его сфера применения. 4](#__RefHeading___Toc15220_3037588987)

[2 Среда разработки и кроссплатформенная среда разработки компьютерных игр 7](#__RefHeading___Toc15222_3037588987)

[2.1 *Visual Studio Code* 7](#__RefHeading___Toc15224_3037588987)

[2.2 *Unreal Engine* 9](#__RefHeading___Toc15226_3037588987)

[3 Алгоритмизация 12](#__RefHeading___Toc15228_3037588987)

[3.1 Понятие Алгоритмизация и история 12](#__RefHeading___Toc15230_3037588987)

[4 Процедурная генерация: разработка и описание алгоритмов 14](#__RefHeading___Toc15232_3037588987)

[4.1 Процедурная генерация 14](#__RefHeading___Toc15234_3037588987)

[4.2 Описание алгоритма Шум Перлина (*Perlin Noise*) 15](#__RefHeading___Toc15240_3037588987)

[4.3 Реализация алгоритма Шум Перлина (*Perlin Noise*) 18](#__RefHeading___Toc15242_3037588987)

[4.4 Описание алгоритма *Diamond-Square (DS)* 20](#__RefHeading___Toc15244_3037588987)

[4.5 Описание алгоритма клеточные автоматы (*Cellular Automata, CA*) 21](#__RefHeading___Toc15246_3037588987)

[4.6 Описание алгоритма *Voronoi Diagrams* (Диаграммы Вороного) 22](#__RefHeading___Toc15248_3037588987)

[Заключение 24](#__RefHeading___Toc15250_3037588987)

[Список использованной литературы 25](#__RefHeading___Toc15252_3037588987)

[Приложение А 26](#__RefHeading___Toc15258_3037588987)

[Приложение Б 29](#__RefHeading___Toc15260_3037588987)

[Приложение В 31](#__RefHeading___Toc15262_3037588987)

[Приложение Г 33](#__RefHeading___Toc2023_3318719214_Copy_a)

[Приложение Д 35](#__RefHeading___Toc15266_3037588987)

[Приложение Е 37](#__RefHeading___Toc15268_3037588987)

[Приложение Ё 39](#__RefHeading___Toc15270_3037588987)

[Приложение Ж 41](#__RefHeading___Toc15272_3037588987)

[Приложение З 43](#__RefHeading___Toc15274_3037588987)

**ВВЕДЕНИЕ**

В современном мире компьютерных технологий особое внимание уделяется разработке и оптимизации алгоритмов, которые лежат в основе создания сложных и интерактивных приложений, таких как видеоигры. Одним из ключевых аспектов разработки игр является генерация игровых карт, которая требует не только творческого подхода, но и технической точности для создания уникальных и захватывающих игровых миров.

Целью данной курсовой работы является разработка эффективных алгоритмов для генерации игровых карт, которые могут быть реализованы на популярном языке программирования *C++.* Этот язык был выбран из-за его широкого распространения в индустрии игрового дизайна, а также благодаря способности обеспечивать высокую производительность и гибкость при работе с графическими и алгоритмическими задачами.

В ходе работы будет проведен анализ существующих методов генерации карт, исследованы их преимущества и недостатки. Особое внимание будет уделено алгоритмам процедурной генерации, которые позволяют создавать большие и детализированные игровые пространства с минимальными затратами времени и ресурсов.

Таким образом, курсовая работа направлена на решение актуальной задачи в области компьютерных игр и программирования, и представляет собой вклад в развитие методов генерации игровых карт и повышение качества игровых продуктов.

Пояснительная записка оформлена в соответствии с СТП 01-2024.

1. ЯЗЫК ПРОГРАММИРОВАНИЯ *С++* И СФЕРЫ ЕГО ПРИМЕНЕНИЯ
   1. Язык *С++* и его сфера применения.

Язык программирования *C++* имеет богатую историю. Он был создан в 1983 году Бьёрном Страуструпом в *Bell Laboratories* как расширение языка *C*, чтобы добавить возможности объектно-ориентированного программирования. *С++* сочетает свойства как высокоуровневых, так и низкоуровневых языков, поддерживая процедурное программирование, объектно-ориентированное программирование и обобщённое программирование.



|  |
| --- |
|  |
| Рисунок 2.1 – Логотип языка программирования *C++* |

С тех пор язык продолжал развиваться, и его стандарты обновлялись для включения новых возможностей.

Изначально одним из принципов разработки *C++* было сохранение совместимости с *C*, хотя *C++* не является в строгом смысле надмножеством *C*. Множество программ на *C* могут быть успешно скомпилированы как компиляторами *C*, так и компиляторами *C++*, но не все. Это делает *C++* уникальным инструментом, который находит применение в разработке операционных систем, приложений для встраиваемых систем, высокопроизводительных серверов и компьютерных игр.

Язык программирования *C++* обладает рядом преимуществ, которые делают его популярным выбором для разработчиков:

­- Скорость: программы на *C++* могут быть очень быстрыми, так как это компилируемый язык с близким к металлу уровнем доступа к памяти.

-­ Экономичность: программы на *C++* имеют малое потребление оперативной памяти, и исполняемый файл обычно имеет небольшой размер.

-­ Кроссплатформенность: программы, написанные на *C++*, могут быть скомпилированы и запущены на различных платформах и операционных системах.

-­ Возможность работы на низком уровне: *C++* предоставляет возможности для работы с памятью, адресами и портами, что важно для системного программирования.

-­ Обобщённое программирование: язык поддерживает шаблоны, позволяющие создавать обобщенные алгоритмы для разных типов данных.

Эти достоинства делают *C++* мощным инструментом для создания высокопроизводительного и эффективного программного обеспечения в различных областях.

Несмотря на множество преимуществ, язык программирования C++ имеет и некоторые недостатки:

**-­ Сложность**: *C++* известен своей сложностью, особенно для новичков. Он имеет множество возможностей и тонкостей, которые могут быть трудными для понимания и использования.

-­ Управление памятью: в *C++* программистам необходимо самостоятельно управлять памятью, что может привести к ошибкам, таким как утечки памяти и повреждение данных.

-­ Отсутствие сборщика мусора: в отличие от некоторых других языков, в *C++* нет автоматического управления памятью, что увеличивает риск ошибок.

-­ Сложность синтаксиса: синтаксис *C++* может быть перегруженным и сложным для чтения, что затрудняет отладку и поддержку кода.

-­ Медленное время компиляции: из-за сложности языка и обилия функций время компиляции программ на *C++* может быть довольно длительным.

Эти недостатки могут сделать разработку на *C++* более трудоемкой и подверженной ошибкам, особенно для тех, кто только начинает работать с языком.

Язык *C++* часто используется для написания алгоритмов генерации игровых карт в играх по нескольким причинам:

**-­ Производительность**: *C++* известен своей высокой производительностью и близостью к аппаратному обеспечению, что позволяет разработчикам максимально оптимизировать код для быстрой обработки данных, что критически важно для алгоритмов генерации карт, требующих большого количества вычислений.

**-­ Контроль над памятью**: *C++* предоставляет разработчикам глубокий контроль над управлением памятью, что позволяет создавать эффективные и оптимизированные алгоритмы без лишнего потребления ресурсов.

**-­ Объектно-ориентированное программирование (ООП)**: ООП позволяет структурировать сложные алгоритмы и данные, что упрощает разработку и поддержку кода.

**-­ Богатая стандартная библиотека**: стандартная библиотека *C++* предоставляет множество готовых решений для общих задач программирования, что ускоряет процесс разработки.

Эти преимущества делают *C++* предпочтительным выбором для разработки игр, особенно когда требуется высокая производительность и контроль над ресурсами системы.

Язык программирования C++ используется во многих областях благодаря своей высокой производительности и гибкости. Вот некоторые из основных сфер применения *C++*:

-­ Операционные системы: *C++* часто используется для разработки операционных систем, так как он позволяет работать с низкоуровневым кодом и управлять системными ресурсами.

-­ Программирование встроенных систем: язык идеально подходит для разработки встроенного ПО, например, для беспилотных автомобилей, умных часов и устройств Интернета вещей.

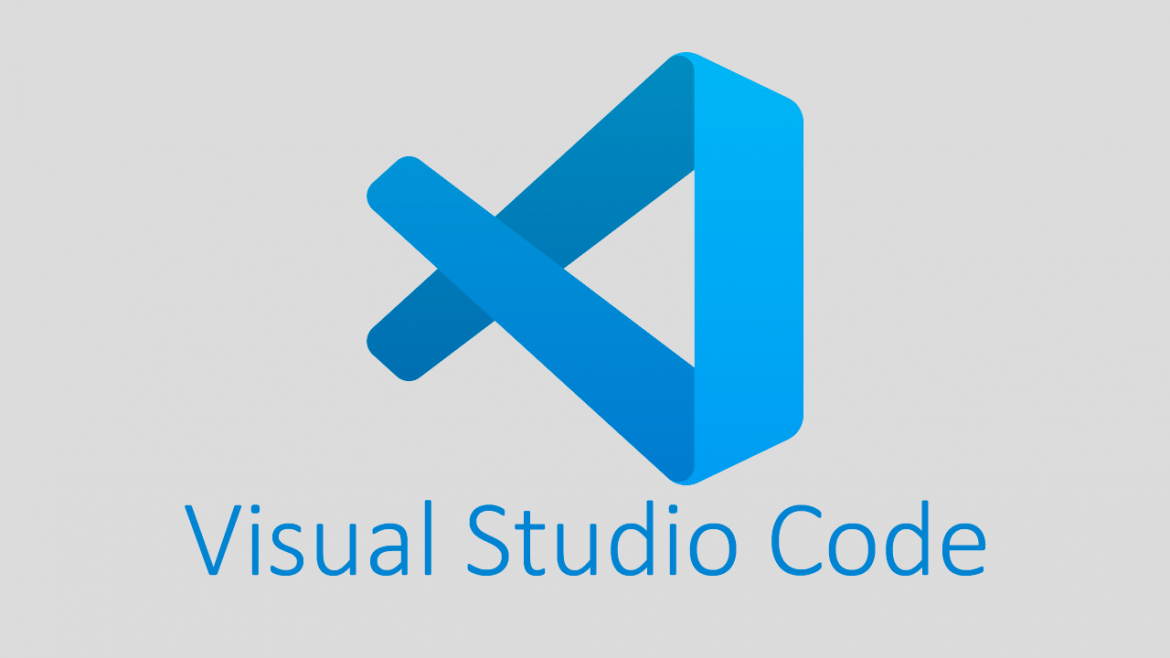
-­ Разработка игр и игровых движков: *C++* используется для создания высокопроизводительных игр и игровых движков, таких как *Unity*.

-­ Разработка баз данных: благодаря своей способности к управлению памятью и скорости, *C++* применяется для создания сложных баз данных.

Эти примеры демонстрируют универсальность *C++* и его способность справляться с различными задачами программирования.

**2 СРЕДА РАЗРАБОТКИ И КРОССПЛАТФОРМЕННАЯ СРЕДА РАЗРАБОТКИ КОМПЬЮТЕРНЫХ ИГР**

## 2.1 *Visual Studio Code*

 *Visual Studio Code (VS Code)* — это мощный редактор исходного кода, разработанный *Microsoft* для *Windows*, *Linux* и *macOS*. Он представляет собой “лёгкий” инструмент для кроссплатформенной разработки веб- и облачных приложений, включающий в себя отладчик, инструменты для работы с *Git*, подсветку синтаксиса, *IntelliSense* и средства для рефакторинга.

|  |
| --- |
| Рисунок 2.1 – Логотип *Visual Studio Code* |

История появления: *Visual Studio Code* был анонсирован 29 апреля 2015 года на конференции *Build* компанией *Microsoft*. Вскоре после этого была выпущена бета-версия. 18 ноября 2015 года *VS Code* был выпущен под лицензией *MIT*, и исходный код был опубликован на *GitHub*. Это было значительным шагом, так как позволило сообществу разработчиков вносить свой вклад в проект. 14 апреля 2016 года *Visual Studio Code* вышел из стадии бета-тестирования и стал доступен для широкой публики. *VS Code* основан на *Electron* и использует веб-редактор *Monaco*, разработанный для *Visual Studio Online*. Это обеспечивает высокую скорость работы и возможность использования редактора на различных платформах.

*VS Code* быстро завоевал популярность среди разработчиков благодаря своей гибкости, мощным функциям и активному сообществу, которое постоянно работает над улучшением и расширением его возможностей. *Visual Studio Code (VS Code)* обладает множеством преимуществ, которые делают его популярным выбором среди разработчиков:

-­ Легкость освоения: *VS Code* имеет интуитивно понятный интерфейс, его легко установить и настроить.

-­ Поддержка множества языков программирования: редактор поддерживает широкий спектр языков, включая *HTML/CSS, JavaScript, PHP, Python, Go, Ruby, C#, TypeScript* и многие другие.

-­ Расширяемость: благодаря маркетплейсу с более чем 9000 расширений, пользователи могут настраивать редактор под свои нужды, добавляя поддержку различных фреймворков и инструментов.

-­ Интеграция с *Git*: *VS Code* предлагает встроенные инструменты для работы с системой контроля версий *Git*.

-­ IntelliSense: технология автодополнения кода *IntelliSense* ускоряет написание кода благодаря предложениям и информации о типах данных, функциях и т. д.

-­ Отладка: встроенные средства профилирования и отладки помогают выявлять низкопроизводительные места кода.

-­ Бесплатное использование: *VS Code* распространяется бесплатно, что делает его доступным для широкого круга разработчиков.

-­ Кроссплатформенность: поддерживает *Windows*, *Linux* и *macOS*, что позволяет использовать его на любой операционной системе.

-­ Активное сообщество: большое комьюнити разработчиков и регулярные обновления от *Microsoft* обеспечивают постоянное улучшение и расширение возможностей редактора.

*Visual Studio Code (VS Code)* является одним из самых популярных редакторов кода, но, как и любой инструмент, он имеет свои недостатки:

-­ Ограниченные возможности рефакторинга: по сравнению с полноценными интегрированными средами разработки (*IDE*), такими как *PyCharm, VS Code* может предложить менее мощные инструменты для рефакторинга.

-­ Отладка: хотя *VS Code* предлагает встроенные средства отладки, они могут быть менее продвинутыми по сравнению с теми, что доступны в специализированных *IDE*.

-­ Требуются расширения для специфичных функций: для некоторых языков программирования и фреймворков необходимо устанавливать дополнительные расширения, чтобы обеспечить полноценную поддержку.

-­ Производительность: на слабых компьютерах *VS Code* может работать медленнее, особенно при работе с большими проектами.

-­ Безопасность: были выявлены серьезные недостатки безопасности в некоторых популярных расширениях, которые могут создать уязвимости для кибератак.

-­ Специфические баги: пользователи иногда сталкиваются со специфическими багами на некоторых языках программирования.

*Visual Studio Code (VS Code)* — это мощный и гибкий редактор кода, который можно использовать в различных сферах программирования:

-­ Разработка веб-приложений: *VS Code* поддерживает *HTML, CSS, JavaScript* и множество других языков и фреймворков, что делает его идеальным инструментом для фронтенд и бэкенд разработки.

-­ Разработка мобильных приложений: с помощью плагинов и расширений, *VS Code* может быть использован для разработки мобильных приложений на платформах, таких как *Android* и *iOS*.

-­ Работа с базами данных: редактор предоставляет инструменты для работы с *SQL* и другими системами управления базами данных.

-­ Кроссплатформенная разработка: *VS Code* работает на *Windows, macOS* и *Linux*, что позволяет разработчикам использовать один и тот же инструмент независимо от операционной системы.

-­ Разработка консольных и *GUI* приложений: *VS Code* поддерживает разработку как консольных приложений, так и приложений с графическим интерфейсом, включая поддержку технологии *Windows Forms*.

-­ Интеграция с системами контроля версий: *VS Code* обладает встроенной поддержкой *Git*, что упрощает управление версиями и совместную работу над проектами.

## 2.2 *U******nreal Engine*****

***Unreal Engine*** — это игровой движок, разрабатываемый и поддерживаемый компанией ***Epic Games***. Первой игрой, созданной на этом движке, был шутер от первого лица ***Unreal***, выпущенный в **1998 году**. Хотя движок изначально предназначался для разработки шутеров от первого лица, его последующие версии успешно применялись в играх самых различных жанров, включая стелс-игры, файтинги, массовые многопользовательские ролевые онлайн-игры и многие другие.



|  |
| --- |
| Рисунок 2.1 – Логотип Unreal Engine |

В прошлом *Unreal Engine* распространялся на условиях оплаты ежемесячной подписки. С 2015 года он стал бесплатным, но разработчики, использующие его приложения, обязаны перечислять **5% роялти** от общемирового дохода с некоторыми условиями.

Интересно, что *Unreal Engine* был адаптирован и для применения в играх жанра *MMORPG*, таких как ***Lineage II****.* Этот игровой движок оказал огромное влияние на историю видеоигр и продолжает использоваться в культовых проектах.

**Плюсы *Unreal Engine*:**

**-­ Графика высокого уровня**: *Unreal Engine* обладает мощными возможностями для создания впечатляющей графики. Он часто используется для разработки высокобюджетных игр класса *Triple-A.*

**­- Портативность**: *Unreal Engine* написан на *C++* и обладает высокой степенью переносимости. Множество разработчиков используют его в своих проектах.

**-­ Множество платформ**: *Unreal Engine* поддерживает множество платформ, включая *Windows, macOS, Linux, iOS, Android, Nintendo Switch, PlayStation, Xbox, Magic Leap One* и виртуальную реальность.

**-­ Награды и признание**: *Unreal Engine* получил множество наград, включая рекорд *Guinness World Records* как “самый успешный игровой движок” в 2014 году.

**Минусы *Unreal Engine:***

**-­ Крутой кривой обучения**: *Unreal Engine* имеет крутой кривой обучения, особенно для новичков.

**-­ Ресурсоемкость**: работа с *Unreal Engine* может потребовать значительных ресурсов, включая высокопроизводительное оборудование.

**-­ Лицензионные расходы**: для коммерческих проектов необходимо уплачивать 5% роялти от дохода.

**Сфера применения *Unreal Engine:***

**-­ Игровая индустрия**: *Unreal Engine* широко используется для разработки видеоигр разных жанров, от шутеров до *RPG* и симуляторов.

**-­ Архитектурная визуализация**: *Unreal Engine* помогает создавать реалистичные визуализации архитектурных проектов.

**-­ Фильмы и телевидение**: движок используется для создания спецэффектов, анимации и виртуальных съемок.

3 АЛГОРИТМИЗАЦИЯ

## 3.1 Понятие Алгоритмизация и история

Алгоритмизация — это процесс разработки и представления алгоритмов, то есть последовательности инструкций, которые выполняются для решения определенной задачи или достижения определенной цели. Она включает в себя выражение алгоритмов в языке, понятном компьютеру или другому исполнителю, который может *U****nreal Engine*** выполнить эти инструкции.

История алгоритмизации тесно связана с развитием вычислительной техники и информатики в целом. Вот несколько ключевых моментов в истории алгоритмизации:

-­ Древнейшие алгоритмы: уже в древних цивилизациях люди использовали алгоритмы для решения различных задач. Например, древние египтяне использовали алгоритмы для выполнения арифметических операций и построения пирамид.

-­ Алгоритмы в математике: с развитием математики появились формальные методы записи алгоритмов. Известные математики, такие как Ал-Хорезми и Леонардо Пизанский (Фибоначчи), разработали алгоритмы для решения математических задач и описали их в своих трудах.

-­ Машины Тьюринга: в 1936 году Алан Тьюринг предложил концепцию универсальной вычислительной машины, известной сейчас как машина Тьюринга. Он показал, что любой алгоритм может быть представлен в виде последовательности действий, выполняемых такой машиной. Это сформировало основу для развития теории алгоритмов и компьютерных наук.

-­ Развитие компьютеров: в середине XX века появились первые электронные цифровые компьютеры. С развитием вычислительной техники стало возможным создание и выполнение сложных алгоритмов. Программирование стало процессом разработки и записи алгоритмов на языке, понятном компьютеру.

-­ Современная алгоритмизация: с развитием информационных технологий и компьютерных наук алгоритмизация стала ключевым понятием в различных областях, таких как программирование, искусственный интеллект, анализ данных и другие. Современные языки программирования и среды разработки предоставляют различные инструменты для разработки и представления алгоритмов.

Сегодня алгоритмизация играет важную роль во многих сферах, и ее применение становится все более широким и разнообразным.

Алгоритмизация обладает рядом достоинств и широко применяется в различных сферах. Вот некоторые из ее преимуществ и областей применения:

-­ Эффективность: алгоритмизация позволяет систематизировать и автоматизировать выполнение задач. Хорошо разработанные алгоритмы могут ускорять процессы, оптимизировать ресурсы и повышать общую эффективность работы.

-­ Точность: алгоритмы могут быть разработаны с высокой степенью точности и надежности. Они позволяют минимизировать ошибки, связанные с человеческим фактором, и обеспечивают консистентность в выполнении задач.

-­ Автоматизация: алгоритмизация является основой автоматизации процессов. Она позволяет создавать программы и системы, которые могут выполнять задачи без прямого участия человека. Примеры включают автоматизацию производственных процессов, управление базами данных, робототехнику и автономные системы.

-­ Масштабируемость: хорошо разработанные алгоритмы могут быть масштабированы для работы с большими объемами данных или выполнения сложных операций. Они позволяют обрабатывать и анализировать большие массивы информации, что важно в сферах, связанных с анализом данных, исследованиями и вычислительной наукой.

-­ Искусственный интеллект: алгоритмизация играет ключевую роль в разработке и реализации систем искусственного интеллекта. Алгоритмы машинного обучения, нейронные сети и другие техники основаны на алгоритмизации и позволяют компьютерам обучаться, анализировать данные и принимать решения на основе опыта.

-­ Криптография и безопасность: алгоритмизация играет важную роль в области криптографии и обеспечения безопасности информации. Разработка безопасных алгоритмов шифрования и протоколов защиты позволяет обеспечить конфиденциальность и целостность данных в сетях и системах обмена информацией.

4 ПРОЦЕДУРНАЯ ГЕНЕРАЦИЯ: РАЗРАБОТКА И ОПИСАНИЕ АЛГОРИТМОВ

## 4.1 Процедурная генерация

Процедурная генерация уровней (или ***Procedural Level Generation***) — это метод создания игрового контента алгоритмически, в отличие от ручного создания. Он использует комбинацию контента, созданного человеком, и алгоритмов, сопряженных с компьютерной случайностью и вычислительной мощностью. Процедурная генерация применяется в различных областях разработки игр, включая создание текстур, *3D*-моделей и генерацию игровых уровней.

Общий подход к процедурной генерации уровней:

**-­ Определение параметров уровня**: сначала определяются параметры уровня, такие как размер, сложность, структура и цель уровня.

**-­ Алгоритм генерации**: затем создается алгоритм, который будет генерировать уровни. Например, можно начать с прямоугольника, представляющего всю игровую область, и разделить его на две части (горизонтально или вертикально) случайным образом.

**-­ Инстанциация объектов**: алгоритм создает игровые объекты, такие как платформы, враги, предметы и декорации, на основе заданных параметров. Это может включать в себя случайное размещение объектов, чтобы каждый уровень был уникальным.

**-­ Тестирование и балансировка**: сгенерированные уровни тестируются на играбельность и баланс. Если уровни слишком сложные или слишком простые, алгоритм может быть настроен для более оптимальных результатов.

Процедурная генерация уровней имеет несколько **преимуществ**:

**-­ Переиграбельность**: благодаря процедурной генерации каждый уровень может быть уникальным. Игроки не будут скучать, так как каждый раз они сталкиваются с разными вызовами и ситуациями.

**-­ Экономия времени и ресурсов**: ручное создание уровней требует много времени и усилий. Процедурная генерация позволяет сэкономить время разработчиков, освобождая их для других задач.

**-­ Адаптивность к разным устройствам**: процедурно сгенерированные уровни могут легко адаптироваться к разным платформам и разрешениям экрана.

**-­ Сложность и баланс**: алгоритмы могут автоматически настраивать сложность уровней, чтобы они были интересными и вызывали у игроков желание продолжать играть.

**­- Новые идеи и вариации**: процедурная генерация может привести к неожиданным комбинациям элементов уровней, что способствует творчеству и разнообразию.

Каждый алгоритм имеет как свои преимущества, так и недостатки. Далее представлены некоторые минусы алгоритма процедурной генерации уровней:

**-­ Потенциальная непредсказуемость**: процедурно сгенерированные уровни могут иногда быть несбалансированными или даже непроходимыми. Это может вызвать разочарование у игроков, особенно если они сталкиваются с нечестными ситуациями.

**-­ Ограниченность в дизайне**: алгоритмы не всегда способны создать уровни с такой же художественной проработкой, как ручное создание. Процедурная генерация может ограничивать вариативность и креативность дизайна.

**-­ Сложность балансировки**: хотя алгоритмы могут настраивать сложность, достижение идеального баланса между интересными вызовами и уровнем сложности может быть сложной задачей.

**-­ Потеря ручного влияния**: процедурная генерация уровней убирает ручное влияние разработчиков, что может быть нежелательным в некоторых случаях.

**-­ Сложность в создании сюжетных уровней**: процедурная генерация хорошо работает для аркадных игр, но создание сюжетных уровней с определенными событиями и задачами может быть сложнее.

Процедурная генерация уровней позволяет создавать разнообразные и интересные игровые миры, обеспечивая высокую переиграбельность и экономию ресурсов разработчиков.

## 4.2 Описание алгоритма Шум Перлина (*Perlin Noise*)

*Perlin noise* (Шум Перлина, также иногда Классический шум Перлина) — математический алгоритм по генерированию процедурной текстуры псевдослучайным методом.

Используется в компьютерной графике для увеличения реализма или графической сложности поверхности геометрических объектов. Также может использоваться для генерации эффектов дыма, тумана и так далее.

Алгоритм Шума Перлина был разработан Кеном Перлином в 1983 году. Кен Перлин в *Mathematical Applications Group, Inc.* и создал этот алгоритм, который позднее был назван в его честь.

Ниже алгоритм рассмотрен более подробно:

-­ Шум Перлина представляет собой **градиентный шум**, состоящий из набора псевдослучайных единичных векторов (направлений градиента), которые расположены в определенных точках пространства и интерполированы функцией сглаживания между этими точками.

-­ Для генерации шума Перлина в одномерном пространстве необходимо для каждой точки этого пространства вычислить значение шумовой функции, используя направление градиента (или наклон) в указанной точке.

-­ Алгоритм можно масштабировать в одно-, двух- и трехмерном виде, а также ввести четвертое временное измерение, позволяя динамически изменять текстуры во времени.

**Алгоритм Шума Перлина** обладает несколькими преимуществами:

**1. Процедурная генерация:**

A) Шум Перлина является процедурной текстурой, что означает, что он не требует большого объема памяти для хранения текстурных данных.

Б) Вместо этого алгоритм генерирует текстуры на лету, используя математические вычисления. Это особенно полезно при работе с большими объемами данных или в реальном времени.

**2. Натуральные узоры:**

А) Шум Перлина создает текстуры, которые имеют натуральный вид. Он может использоваться для имитации различных природных явлений, таких как облака, волны, камни, листва и т. д.

Б) Градиентный характер шума Перлина позволяет создавать плавные переходы между разными уровнями интенсивности.

**3. Контроль параметров:**

А) Алгоритм Шума Перлина легко настраивается. Вы можете изменять параметры, такие как частота, амплитуда и октавы, чтобы получить разнообразные текстуры.

Б) Это позволяет адаптировать алгоритм к конкретным потребностям и задачам.

**4. Использование в компьютерной графике:**

А) Шум Перлина широко применяется в компьютерной графике для создания реалистичных и красочных визуальных эффектов.

Б) Он используется в играх, анимации, фильмах и других мультимедийных проектах.

В целом, алгоритм Шума Перлина предоставляет мощный инструмент для генерации процедурных текстур, и его гибкость делает его популярным среди разработчиков и художников.

Даже имея такой перечень плюсов, у Шума Перлина есть и минусы:

**1. Артефакты и регулярность:**

А) Шум Перлина может создавать артефакты, такие как видимые сетки или регулярные узоры, особенно при низкой частоте или недостаточной амплитуде.

Б) Это может привести к нежелательным визуальным эффектам, особенно если алгоритм используется для создания больших поверхностей.

**2. Ограниченная разнообразность:**

А) Шум Перлина, хотя и гибкий, имеет свои ограничения в создании разнообразных текстур.

Б) Например, он может имитировать некоторые природные узоры, но не всегда может точно воспроизвести сложные детали или необычные формы.

**2. Сложность настройки:**

А) Настройка параметров алгоритма может быть сложной задачей.

Б) Выбор правильных значений для частоты, амплитуды и октав может потребовать опыта и экспериментов.

**3. Вычислительная нагрузка:**

А) Шум Перлина требует вычислительных ресурсов для генерации текстур.

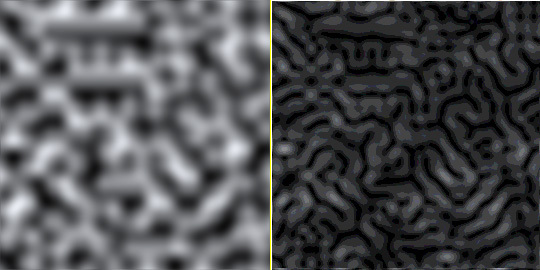
Б) В некоторых случаях это может быть проблемой, особенно при работе в реальном времени или на мобильных устройствах.

**4. Отсутствие глобальной координации:**

А) Шум Перлина генерируется локально для каждой точки пространства.

Б) Это означает, что он не учитывает глобальные свойства текстуры, такие как ее общая структура или симмети.

|  |
| --- |
|  |
|  |
| Рисунок 4.1 – Двумерный срез трёхмерной разновидности шума Перлина. |

Этот алгоритм широко используется в двухмерной и трехмерной компьютерной графике для создания визуальных эффектов, таких как дым, облака, туман, огонь и многое другое. Он также часто используется как простая текстура, покрывающая геометрические модели. В отличие от растровых текстур, шум Перлина является процедурной текстурой, не занимающей память, но требующей вычислительных ресурсов для выполнения. Неудивительно, что он нашел свое применение в разработке игр, демосцене и других областях.

## 4.3 Реализация а****лгоритма Шум Перлина (*Perlin Noise*)****

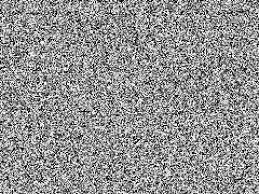
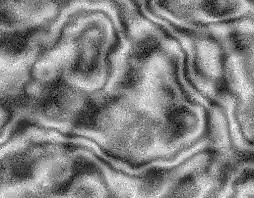
Алгоритм «*Perlin noise*» проводит генерацию текстур методом генерации псевдослучайных чисел, однако все визуальные детали текстуры имеют одинаковый размер. Это свойство делает шум Перлина легко управляемым; множества масштабированных копий шума Перлина могут быть вставленными в математические выражения для создания самых разнообразных процедурных текстур.

Перлин дал следующее определение шума: шум — это аппроксимация к белому шуму, ограниченная по диапазону одной октавой. Формальное определение шума имеет следующий вид:



**Рисунок 4.2 – Формальное определение шума.**

Где *N* — функция шума. Шум Перлина — это процедурный шум. «Прилагательное процедурный используется в компьютерных науках, чтобы отделить сущности, описываемые программным кодом, от описываемых структурами данных». То есть нечто сгенерированное, а не заданное жёстко.

 На рисунке 3 показано отличие белого шума от шума Перлина:

|  |  |
| --- | --- |
| **Белый шум** | **Шум Перлина** |
|  |  |

**Рисунок 4.3 – Белый шум и шум Перлина.**

Основные этапы работы алгоритма:

**1. Инициализация**:

А) Начнем с задания seed (начального числа), которое определяет структуру ландшафта.

Б) Создаем сетку точек в многомерном пространстве (одно-, двух- или трехмерном).

**2. Генерация градиентов**:

А) В каждой точке сетки генерируем псевдослучайный вектор (градиент).

Б) Градиенты могут указывать направление изменения функции в этой точке.

**3. Интерполяция градиентов**:

А) Между ближайшими точками интерполируем градиенты, чтобы получить плавные переходы.

Б) Это делается с помощью функции сглаживания, такой как Body Textкубическая интерполяция.

**4. Генерация шума**:

А) В каждой точке сетки вычисляем шум, используя интерполированные градиенты.

Б) Шум может быть сглаженным или детализированным в зависимости от параметров алгоритма.

**5.Применение к текстурам и рельефам**:

А) Полученный шум можно использовать для создания текстур, рельефов, карт, вершин и других структур.

## 4.4 Описание а****лгоритма *Diamond-Square (DS)*****

Алгоритм *Diamond-Square* — это метод генерации карт высот для компьютерной графики, который используется для создания фрактальных ландшафтов. Он был впервые представлен на конференции *SIGGRAPH* в 1982 году.

Далее показаны шаги выполнения алгоритма:

Шаг 1. Инициализация угловых точек. Присваивание им значений высот (например, выбором случайных чисел).

Шаг 2. Шаг *diamond*. Нахождение срединной точки, присваивание ей значения, на основе среднего от угловых, плюс случайное число.

Шаг 3. Шаг *square*. Нахождение срединных точек для ромбов, отмеченных черными точками (на этом шаге, по одной точке каждого ромба выходят за пределы массива). Плюс случайное число.

Шаг 4. Шаг *diamond*. Для каждого квадрата (на этом шаге их 4), повторяем шаг № 2.

Шаг 5. Шаг *square*. Повторяем шаг № 3 для каждого ромба. У ромбов, имеющих точки на краю массива, одна из точек выходит за пределы массива.

На рисунке ниже показаны шаги, проходимые алгоритмом *diamond-square* на примере массива 5х5.

|  |
| --- |
| Рисунок 4.4 – Шаги, проходимые алгоритмом *diamond-square* на примере массива 5х5. |

Эти два шага повторяются, уменьшая случайное значение с каждой итерацией, что создаёт более мелкие детали ландшафта. Случайное число обычно выбирается в промежутке *([-R\_i, R\_i])*, где (*R*) — это фактор неровности, а (*i*) — номер итерации. При последовательном использовании с указанием общих начальных значений, этот метод позволяет «сшивать» генерируемые карты высот.

Этот алгоритм может быть использован для генерации реалистичных ландшафтов в различных компьютерных графических программах, например, *Terragen*.

## 4.5 Описание а****лгоритма** **клеточные автоматы (*Cellular Automata, CA*)****

**Клеточные автоматы (*Cellular Automata, CA*) — это модель системы, состоящей из объектов-ячеек, которые обладают следующими характеристиками:**

-­ Ячейки расположены на сетке (гриде). Эта сетка может быть одномерной, двумерной или даже многомерной.

-­ Каждая ячейка имеет состояние, которое может изменяться со временем. Количество возможных состояний обычно конечно. Простейший пример — это два состояния: 1 и 0 (или включено и выключено, живое и мертвое).

-­ У каждой ячейки есть “соседство” — это могут быть все соседние ячейки, окружающие данную ячейку. Определение соседства может варьироваться, но часто это все смежные ячейки.

Алгоритм работы клеточного автомата заключается в следующем:

**-­ Инициализация**: сначала задается начальное состояние сетки, где каждая ячейка может быть случайно установлена в одно из возможных состояний.

**-­ Обновление**: на каждом шаге алгоритма (или поколении) состояние каждой ячейки обновляется одновременно в соответствии с набором правил, которые определяют новое состояние ячейки на основе ее текущего состояния и состояний ее соседей.

**-­ Правила**: Правила могут быть очень простыми или очень сложными и могут включать в себя различные факторы, такие как количество “живых” соседей вокруг ячейки.

Примером клеточного автомата является знаменитая “Игра «Жизнь»” (*Conway’s Game of Life)*, где каждая ячейка на двумерной сетке может быть живой или мертвой, и ее состояние на следующем шаге определяется количеством живых соседей вокруг нее.

В программировании клеточные автоматы могут быть реализованы различными способами, в зависимости от требований и предпочтений разработчика. Они могут использоваться для моделирования сложных систем, создания генеративного искусства, разработки игр и многих других задач, где требуется моделирование динамических систем с большим количеством взаимодействующих элементов.

## **4.6 **Описание алгоритма *Voronoi Diagrams* (Диаграммы Вороного)****

**Диаграммы Вороного — это математический инструмент, который может быть использован для создания игровых карт, особенно в жанре процедурной генерации. Они позволяют создавать карты с естественно выглядящими разделениями территорий, что идеально подходит для игр, где требуется реалистичное и непредсказуемое распределение ресурсов, зон влияния или биомов.**

Вот базовое описание алгоритма создания диаграмм Вороного для игровых карт:

**1. Генерация точек**: сначала генерируются случайные точки на карте, которые будут служить “семенами” для будущих ячеек Вороного.

**2. Расчет диаграммы**: для каждой точки на карте вычисляется ближайшая “семенная” точка. Регион, состоящий из всех точек, ближайших к одной “семенной” точке, формирует ячейку Вороного.

**3. Применение биомов и текстур**: после формирования ячеек к ним применяются различные текстуры и характеристики, соответствующие разным биомам или типам местности.

**4. Оптимизация и детализация**: дополнительные алгоритмы могут быть использованы для добавления деталей, таких как реки, дороги и другие структуры, чтобы карта выглядела более естественно и интересно.

В контексте игровой разработки, диаграммы Вороного могут быть использованы для создания уникальных и интересных карт, которые каждый раз будут отличаться друг от друга, предоставляя игрокам новый опыт при каждом новом запуске игры.

**ЗАКЛЮЧЕНИЕ**

Работа с алгоритмами создания карт — это увлекательный и творческий процесс, который позволяет разработчикам создавать уникальные и захватывающие игровые миры.

Данная курсовая работа представляет собой значительный шаг вперед в самообучении в области разработки игровых карт и алгоритмов процедурной генерации. Разработанные и проанализированные методы позволяют не только улучшить качество и разнообразие игровых миров, но и предоставляют разработчикам инструменты для более эффективной и творческой работы. Использование языка программирования *C++* демонстрирует их гибкость и мощь в решении сложных задач генерации контента, что делает их идеальными для использования в современной индустрии видеоигр.

Применение алгоритмов процедурной генерации открывает новые горизонты для создания динамичных и живых игровых миров, которые могут адаптироваться и изменяться в ответ на действия игроков. Это не только повышает уровень погружения и интерактивности, но и способствует созданию более глубоких и запоминающихся игровых опытов.

В целом, результаты исследования подчеркивают важность непрерывного изучения и развития новых технологий в области игрового дизайна. Они также подтверждают, что с помощью правильно выбранных алгоритмов и инструментов разработчики могут преодолеть технические ограничения и реализовать самые смелые творческие замыслы, создавая уникальные игровые миры, которые будут радовать игроков на протяжении многих лет.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Введение в *C++* – [Электронный ресурс]. – Режим доступа: *https://metanit.com/cpp/tutorial/1.1.php* – Дата доступа: 26.04.2024.

2. *Visual Studio Code for Education* - [Электронный ресурс]. – Режим доступа: https://visualstudio.microsoft.com/ru/ – Дата доступа: 26.04.2024.

3. *Unreal Engine* - Википедия – [Электронный ресурс]. – Режим доступа: *https://www.unrealengine.com/en-US* – Дата доступа: 26.04.2024.

4. Репозиторий БГУИР – [Электронный ресурс]. – Режим доступа: *https://libeldoc.bsuir.by/handle/123456789/52232* – Дата доступа: 16.05.2024.

5. Простая процедурная генерация мира, или Шумы Перлина на *Python* – [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/companies/selectel/articles/731506/* – Дата доступа: 26.04.2024.

6. MMO с нуля. Часть 2. Наращивание функционала + алгоритм *Diamond Square* – [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/articles/334786/* – Дата доступа: 26.04.2024.

7. Простейшие клеточные автоматы и их практическое применение – [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/articles/273393/* – Дата доступа: 26.04.2024.

8. Диаграмма Вороного - Википедия – [Электронный ресурс]. – Режим доступа:*https://habr.com/ru/articles/309252/*– Дата доступа: 26.04.2024.

**Приложение А**

**Aлгоритм Шума Перлина**

**#include <iostream>**

**#include <cmath>**

**#include <vector>**

**#include <random>**

**class PerlinNoise {**

**// Перестановки для генерации шума**

**std::vector<int> p;**

**public:**

**// Конструктор**

**PerlinNoise() {**

**// Инициализация вектора перестановок p**

**p = {**

**//Значения**

**};**

**// Перемешивание массива**

**std::shuffle(p.begin(), p.begin() + 256, std::default\_random\_engine());**

**p.insert(p.end(), p.begin(), p.begin() + 256);**

**}**

**// Функция для интерполяции**

**double fade(double t) {**

**return t \* t \* t \* (t \* (t \* 6 - 15) + 10);**

**}**

**// Линейная интерполяция**

**double lerp(double t, double a, double b) {**

**return a + t \* (b - a);**

**}**

**// Градиент**

**double grad(int hash, double x, double y, double z) {**

**int h = hash & 15;**

**double u = h < 8 ? x : y,**

**v = h < 4 ? y : h == 12 || h == 14 ? x : z;**

**return ((h & 1) == 0 ? u : -u) + ((h & 2) == 0 ? v : -v);**

**}**

**// Генерация шума Перлина**

**double noise(double x, double y, double z) {**

**// Определение координат ячейки**

**int X = (int)floor(x) & 255;**

**int Y = (int)floor(y) & 255;**

**int Z = (int)floor(z) & 255;**

**// Нахождение относительных координат в ячейке**

**x -= floor(x);**

**y -= floor(y);**

**z -= floor(z);**

**// Вычисление функции сглаживания**

**double u = fade(x);**

**double v = fade(y);**

**double w = fade(z);**

**// Хеширование координат углов ячейки**

**int A = p[X] + Y, AA = p[A] + Z, AB = p[A + 1] + Z,**

**B = p[X + 1] + Y, BA = p[B] + Z, BB = p[B + 1] + Z;**

**// Интерполяция**

**return lerp(w, lerp(v, lerp(u, grad(p[AA], x, y, z),**

**grad(p[BA], x - 1, y, z)),**

**lerp(u, grad(p[AB], x, y - 1, z),**

**grad(p[BB], x - 1, y - 1, z))),**

**lerp(v, lerp(u, grad(p[AA + 1], x, y, z - 1),**

**grad(p[BA + 1], x - 1, y, z - 1)),**

**lerp(u, grad(p[AB + 1], x, y - 1, z - 1),**

**grad(p[BB + 1], x - 1, y - 1, z - 1))));**

**}**

**};**

**int main() {**

**PerlinNoise perlin;**

**// Генерация шума для 2D точки**

**double x = 0.5;**

**double y = 0.5;**

**std::cout << "Perlin Noise Value at (0.5, 0.5): " << perlin.noise(x, y, 0.0) << std::endl;**

**return 0;**

**}**

**Приложение Б**

**Aлгоритм Diamond-Square**

**#include <iostream>**

**#include <vector>**

**#include <random>**

**#include <cmath>**

**// Функция для генерации случайного числа в заданном диапазоне**

**float randomFloat(float min, float max) {**

**static std::default\_random\_engine generator;**

**std::uniform\_real\_distribution<float> distribution(min, max);**

**return distribution(generator);**

**}**

**// Функция для выполнения "шага квадрата"**

**void squareStep(std::vector<std::vector<float>>& field, int x, int y, int step, float roughness) {**

**int hs = step / 2;**

**float a = field[(x - hs + field.size()) % field.size()][y];**

**float b = field[(x + hs) % field.size()][y];**

**float c = field[x][(y - hs + field[0].size()) % field[0].size()];**

**float d = field[x][(y + hs) % field[0].size()];**

**field[x][y] = (a + b + c + d) / 4 + randomFloat(-roughness, roughness);**

**}**

**// Функция для выполнения "шага алмаза"**

**void diamondStep(std::vector<std::vector<float>>& field, int x, int y, int step, float roughness) {**

**int hs = step / 2;**

**float a = field[(x - hs + field.size()) % field.size()][(y - hs + field[0].size()) % field[0].size()];**

**float b = field[(x - hs + field.size()) % field.size()][(y + hs) % field[0].size()];**

**float c = field[(x + hs) % field.size()][(y - hs + field[0].size()) % field[0].size()];**

**float d = field[(x + hs) % field.size()][(y + hs) % field[0].size()];**

**field[x][y] = (a + b + c + d) / 4 + randomFloat(-roughness, roughness);**

**}**

**// Основная функция алгоритма Diamond-Square**

**void diamondSquare(std::vector<std::vector<float>>& field, int size, float roughness) {**

**int half = size / 2;**

**if (half < 1) return;**

**for (int y = half; y < field.size(); y += size) {**

**for (int x = half; x < field[0].size(); x += size) {**

**squareStep(field, x % field[0].size(), y % field.size(), size, roughness);**

**}**

**}**

**for (int x = 0; x < field.size(); x += half) {**

**for (int y = (x + half) % size; y < field[0].size(); y += size) {**

**diamondStep(field, x % field.size(), y % field[0].size(), size, roughness);**

**}**

**}**

**diamondSquare(field, size / 2, roughness / 2);**

**}****СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Введение в *C++* – [Электронный ресурс]. – Режим доступа: https://metanit.com/cpp/tutorial/1.1.php – Дата доступа: 26.04.2024.

2. Visual Studio Code for Education - [Электронный ресурс]. – Режим доступа: https://visualstudio.microsoft.com/ru/ – Дата доступа: 26.04.2024.

3. Unreal Engine - Википедия – [Электронный ресурс]. – Режим доступа: https://www.unrealengine.com/en-US – Дата доступа: 26.04.2024.

4. Репозиторий БГУИР – [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/handle/123456789/52232 – Дата доступа: 16.05.2024.

5. Простая процедурная генерация мира, или Шумы Перлина на Python – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/selectel/articles/731506/ – Дата доступа: 26.04.2024.

6. MMO с нуля. Часть 2. Наращивание функционала + алгоритм Diamond Square – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/334786/ – Дата доступа: 26.04.2024.

7. Простейшие клеточные автоматы и их практическое применение – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/273393/ – Дата доступа: 26.04.2024.

**8.** **Диаграмма Вороного - Википедия – [Электронный ресурс]. – Режим доступа:https://habr.com/ru/articles/309252/– Дата доступа: 26.04.2024.**

**Приложение В**

**Алгоритм клеточные автоматы (Cellular Automata, CA)**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

// Определение размера сетки

const int width = 10;

const int height = 10;

// Функция для инициализации сетки

std::vector<std::vector<int>> initializeGrid() {

std::srand(std::time(nullptr)); // Использование текущего времени для инициализации генератора случайных чисел

std::vector<std::vector<int>> grid(height, std::vector<int>(width));

for (int y = 0; y < height; ++y) {

for (int x = 0; x < width; ++x) {

grid[y][x] = std::rand() % 2; // Случайное заполнение сетки 0 или 1

}

}

return grid;

}

// Функция для вывода сетки

void printGrid(const std::vector<std::vector<int>>& grid) {

for (const auto& row : grid) {

for (int cell : row) {

std::cout << (cell ? "█" : " ") << " ";

}

std::cout << '\n';

}

}

// Функция для обновления сетки

void updateGrid(std::vector<std::vector<int>>& grid) {

std::vector<std::vector<int>> newGrid = grid;

for (int y = 0; y < height; ++y) {

for (int x = 0; x < width; ++x) {

int aliveNeighbors = 0;

// Проверка всех восьми соседей

for (int i = -1; i <= 1; ++i) {

for (int j = -1; j <= 1; ++j) {

if (i == 0 && j == 0) continue; // Пропускаем саму клетку

int newY = (y + i + height) % height;

int newX = (x + j + width) % width;

aliveNeighbors += grid[newY][newX];

}

}

// Правило: клетка оживает, если у неё ровно три живых соседа

if (aliveNeighbors == 3) {

newGrid[y][x] = 1;

} else if (aliveNeighbors < 2 || aliveNeighbors > 3) {

newGrid[y][x] = 0;

}

}

}

grid = newGrid;

}

int main() {

std::vector<std::vector<int>> grid = initializeGrid();

printGrid(grid); // Вывод начальной сетки

for (int step = 0; step < 10; ++step) {

updateGrid(grid);

printGrid(grid); // Вывод сетки после каждого обновления

std::cout << "---\n";

}

return 0;

}

**Приложение Г**

**Aлгоритм Voronoi Diagrams (Диаграммы Вороного)**

**#include <vector>**

**#include <iostream>**

**#include "voronoi.h" // Предполагается, что вы используете библиотеку для работы с диаграммами Вороного**

**// Структура для представления точки**

**struct Point {**

**double x, y;**

**};**

**// Функция для генерации диаграммы Вороного**

**void generateVoronoiDiagram(const std::vector<Point>& points) {**

**// Инициализация диаграммы Вороного**

**voronoi::VoronoiDiagram diagram(points.size());**

**// Добавление точек в диаграмму**

**for (const auto& point : points) {**

**diagram.addPoint(voronoi::Point(point.x, point.y));**

**}**

**// Генерация диаграммы**

**diagram.construct();**

**// Вывод результата**

**for (const auto& cell : diagram.getCells()) {**

**std::cout << "Ячейка для точки (" << cell.site.x << ", " << cell.site.y << "):" << std::endl;**

**for (const auto& edge : cell.edges) {**

**std::cout << " Ребро от (" << edge.start.x << ", " << edge.start.y << ")";**

**std::cout << " до (" << edge.end.x << ", " << edge.end.y << ")" << std::endl;**

**}**

**}**

**}**

**int main() {**

**// Пример использования**

**std::vector<Point> points = {{0, 0}, {100, 100}, {200, 200}, {300, 300}};**

**generateVoronoiDiagram(points);**

**return 0;**

**}**

**Приложение Д**

**Блок-схема к алгоритму Шум Перлина**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | |  | | | | | | | *ГУИР.6-05-0612-02 002 П3* |
|  |  | |  |  | |  | | | | | | | |
|  |  |  |  |  | *ГУИР.6-05-0612-02 002 П3* | | | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | *Блок-схема Шума Перлина* | | *Лит.* | | | *Масса* | | *Масштаб* | |
| *Изм.* | *Лист* | *№ докум.* | *Подпись* | *Дата* |  |  |  |  | | *1:1* | |
| *Разраб*. | | *Слюсарь С.Ю.* |  |  |
| *Проверил* | | *Чайкин И.С.* |  |  |
|  | |  |  |  | *Лист 1* | | | | *Листов 1* | | |
|  | |  |  |  |  | | *Кафедра информатики гр. 353501* | | | | | | |
|  | |  |  |  |
|  | |  |  |  |

[PAGEREF \_Toc168314117 \h Error: Reference source not found](#_Toc168314117)Error: Reference source not foundError: Reference source not foundError: Reference source not foundError: Reference source not found

**Приложение Е**

**Блок-схема к алгоритму Diamond-Square**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | |  | | | | | | | *ГУИР.6-05-0612-02 003 П3* |
|  |  | |  |  | |  | | | | | | | |
|  |  |  |  |  | *ГУИР.6-05-0612-02 003 П3* | | | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | *Блок-схема алгоритма Diamond-Square* | | *Лит.* | | | *Масса* | | *Масштаб* | |
| *Изм.* | *Лист* | *№ докум.* | *Подпись* | *Дата* |  |  |  |  | | *1:1* | |
| *Разраб*. | | *Слюсарь С.Ю.* |  |  |
| *Проверил* | | *Чайкин И.С.* |  |  |
|  | |  |  |  | *Лист 1* | | | | *Листов 1* | | |
|  | |  |  |  |  | | *Кафедра информатики гр. 353501* | | | | | | |
|  | |  |  |  |
|  | |  |  |  |

[PAGEREF \_Toc168314119 \h Error: Reference source not found](#_Toc168314119)Error: Reference source not foundError: Reference source not foundError: Reference source not foundError: Reference source not found

**Приложение Ё**

**Блок-схема к алгоритму клеточные автоматы**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | |  | | | | | | | *ГУИР.6-05-0612-02 004 П3* |
|  |  | |  |  | |  | | | | | | | |
|  |  |  |  |  | *ГУИР.6-05-0612-02 004 П3* | | | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | *Блок-схема алгоритма клеточные автоматы* | | *Лит.* | | | *Масса* | | *Масштаб* | |
| *Изм.* | *Лист* | *№ докум.* | *Подпись* | *Дата* |  |  |  |  | | *1:1* | |
| *Разраб*. | | *Слюсарь С.Ю.* |  |  |
| *Проверил* | | *Чайкин И.С.* |  |  |
|  | |  |  |  | *Лист 1* | | | | *Листов 1* | | |
|  | |  |  |  |  | | *Кафедра информатики гр. 353501* | | | | | | |
|  | |  |  |  |
|  | |  |  |  |

**Приложение Ж**

**Блок-схема к алгоритму Voronoi Diagrams**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | |  | | | | | | | *ГУИР.6-05-0612-02 ПЗ* |
|  |  | |  |  | |  | | | | | | | |
|  |  |  |  |  | *ГУИР.6-05-0612-02 005 П3* | | | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | *Блок-схема алгоритма Voronoi Diagrams* | | *Лит.* | | | *Масса* | | *Масштаб* | |
| *Изм.* | *Лист* | *№ докум.* | *Подпись* | *Дата* |  |  |  |  | | *1:1* | |
| *Разраб*. | | *Слюсарь С.Ю.* |  |  |
| *Проверил* | | *Чайкин И.С.* |  |  |
|  | |  |  |  | *Лист 1* | | | | *Листов 1* | | |
|  | |  |  |  |  | | *Кафедра информатики гр. 353501* | | | | | | |
|  | |  |  |  |
|  | |  |  |  |

[PAGEREF \_Toc168314121 \h Error: Reference source not found](#_Toc168314121)Error: Reference source not foundError: Reference source not foundError: Reference source not foundError: Reference source not found

**Приложение З**

**Функциональная блок-схема**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | |  | | | | | | | *ГУИР.6-05-0612-02 006 П* |
|  |  | |  |  | |  | | | | | | | |
|  |  |  |  |  | *ГУИР.6-05-0612-02 006 П3* | | | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | *Функциональная блок-схема* | | *Лит.* | | | *Масса* | | *Масштаб* | |
| *Изм.* | *Лист* | *№ докум.* | *Подпись* | *Дата* |  |  |  |  | | *1:1* | |
| *Разраб*. | | *Слюсарь С.Ю.* |  |  |
| *Проверил* | | *Чайкин И.С.* |  |  |
|  | |  |  |  | *Лист 1* | | | | *Листов 1* | | |
|  | |  |  |  |  | | *Кафедра информатики гр. 353501* | | | | | | |
|  | |  |  |  |
|  | |  |  |  |