ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о командном программном проекте на тему:

Проект Крошечная стратегическая игра вдохновленная Civilization и The Settlers of Catan

Выполнили студенты:

группы #БПМИ229, 2 курса	Горбацевич Анна Дмитриевна
группы #БПМИ227, 2 курса	Энгель Данила Александрович
группы #БПМИ225, 2 курса	Ахмадуллин Тимур Рафаэлевич

Принял руководитель проекта:

Колесниченко Елена Юрьевна
Научный сотрудник
Факультет компьютерных наук НИУ ВШЭ

Соруководитель:

Чумаков Олег Олегович
Генеральный директор
ООО “Нивал ВР"

Москва 2024

Содержание

Аннотация

Наш проект заключатся в создании стратегической компьютерной игры в формате локального pass-and-play, то есть игроки смогут играть за одним устройством, передавая его следующему игроку по окончанию своего хода. Средой для разработки игры был выбран Unreal Engine 5 [1]. Исходный код проекта лежит в открытом GitLab репозитории.

Игровое поле состоит из шестиугольных клеток разного типа: вода, плоский ледник, рельефный ледник, заснеженные равнины, заснеженные горы, равнины, горы, пустошь и пустыня. Основным методом взаимодействия игрока с игровым миром являются юниты и здания. Главным юнитом каждого игрока является Командующий, при его уничтожении игрок проигрывает, цель игры — стать единственным не проигравшим игроком. Юниты разделяются на наземные и воздушные, они отличаются набором клеток, по которым могут перемещаться, а также возможностью атаковать разные типы юнитов. Также юнит может быть элитным — элитные юниты умеют взаимодействовать со временем, у каждого юнита есть уникальные поле зрения, мобильность, здоровье и область и сила атаки. Здания не имеют возможности передвигаться и напрямую атаковать, в их числе есть телепорт, станции и заводы. Станции могут добывать ресурсы в зависимости от типа станции и расположения на поле, а заводы из этих ресурсов создают юнитов. Заводы делятся на наземные, воздушные и элитные, производящие соответствующие типы юнитов.

Главной особенностью нашей игры является возможность перемещения во времени и другое подобное взаимодействие со временем.

В игре присутствует динамический туман войны: каждый игрок видит клетки, которые находятся в поле зрения его юнитов, а также остаточные изображения клеток, которые ранее были в поле зрения.

Ключевые слова

игра, разработка игры, стратегическая игра, путешествие во времени, генерация игрового поля, объектно-ориентированное программирование, пикселизирующая постобработка

1 Введение

Настольные игры — одно из самых популярных хобби. Однако многие игровые механики не могут быть реализованы в реальном мире. Для решения данной проблемы появляется все больше компьютерных версий знаменитых настолок. Именно эта идея вдохновила нас на создание данного проекта.

Цель нашего проекта — создание пошаговой стратегической компьютерной игры, вдохновленной компьютерной игрой Supreme Commander [2], в которой все активные игровые элементы — это здания и юниты, а игровое поле состоит из шестиугольных клеток разных типов, как в серии компьютерных игр Civilization [3] или в настольной игре The Settlers of Catan [4].

Главной отличительной особенностью нашей игры является возможность взаимодействия со временем (отправлять в прошлое, откатывать во времени, замораживать во времени и т. д.).

Задачей нашего проекта является создание уникальной игры, которая не будет путать игроков, и провести несколько удачных плейтестов.

Средой для разработки игры был выбран Unreal Engine 5 [1]. Исходный код проекта лежит в открытом GitLab репозитории.

Наша команда решила, что мы хотим уделить больше времени уникальным механикам и полировке игры, поэтому мы выбрали самый простой способ создания игры, где игроки играют против друг друга, передавая игровое устройство — pass and play. Кроме того, при игре на одном устройстве сохраняются атмосфера настольных игр и необходимость собраться с друзьями в одном месте.

1.1 Деление задач между участниками проекта

Горбацевич Анна Дмитриевна

- Освоение среды Unreal Engine 5
- Создание документации для баланса игры
- Разработка всей ветки строителей `Builder`
- Разработка всех финальных юнитов
- Создание черновика промежуточного отчета

- Создание 3D моделей клеток
- Дизайн пользовательского интерфейса
- Проведение плейтестов
- Работа над отчетом

Энгель Данила Александрович

- Освоение среды Unreal Engine 5
- Создание 3D моделей клеток
- Написание обзора литературы
- Создание 3D моделей Командующего, Разведчика и Станции
- Работа над отчетом

Ахмадуллин Тимур Рафаэлевич

- Организационная работа, ревью документации и кода
- Освоение среды Unreal Engine 5
- Внутреннее представление, иерархия и интерфейсы элементов игры
- Алгоритм генерации игрового поля
- Разработка всех абстрактных классов, кроме `Builder`
- Разработка всей ветки зданий `Building`
- Имплементация 3D моделей
- Имплементация тумана войны
- Имплементация игрового контроллера и камеры
- Имплементация пользовательского интерфейса
- Имплементация логики игры, взаимодействия игрока с игрой
- Доработка и сведение промежуточного отчета

- Создание документации для баланса игры
- Организация и проведение плейтестов
- Создание и имплементация анимаций, визуальных эффектов и звуков
- Имплементация воспроизведения событий последнего хода
- Создание главного меню и меню паузы
- Создание 3D моделей клеток, юнитов и зданий
- Имплементация постобработки
- Работа над отчетом

Обзор литературы

В данной проектной работе очень много внимания следовало уделить алгоритму генерации карты. В связи с этим наша команда нашла ряд релевантных работ, в которых описываются различные подходы к решению этой задачи на практике.

Generating multi player maps through multi objective evolution [5]

В третьей главе этой статьи описывается подход к генерации карты в игре Civilization VI [3]. Разработчики этой игры сначала создавали линии хребтов, образующие сушу, с помощью алгоритма генерации фрактальной карты. Одним из способов является метод смещения средней точки — такой подход полезен для генерации двумерных карт. Заполнение игрового поля различными объектами начинается после его отрисовки, для этого команда разработчиков выбрала для своего проекта эволюционый алгоритм — он отталкивается от того, насколько «хорошее» текущее решение и модифицирует свое поведение.

Наша команда посчитала, что такой подход хоть и эффективно решает поставленные задачи, но в то же время требует гораздо больших ресурсов на его реализацию, которыми мы, в рамках данного проекта, не обладаем.

Using Wave Function Collapse algorithm for 2D and 3D level generation [6]

В данной работе описываются способы использования алгоритма коллапса волновой функции (Wave Function Collapse algorithm) [7], данный алгоритм способен генерировать битовые изображения, игровые поля и т. д. на основе паттернов во входных данных или вручную написанных ограничений. Он работает за счет случайных выборов (коллапсов волновой функции), воздействие которых распространяется на соседние клетки, ограничивая их возможные выборы.

В нашем проекте WFC алгоритм исключает случаи соседства несовместимых типов клеток с использованием вручную написанных ограничений.

Procedural terrain generation using Perlin noise [8]

Этот документ рассказывает о том, как используется алгоритм шума Перлина для генерации местности в различных продуктах, для его реализации нам нужны функции шума и интерполяции. Для двумерного случая функция шума принимает пару целых чисел в

качестве параметра и возвращает случайный единичный вектор на основе этого параметра. Если вы передаете тот же параметр два раза, она выдаст тот же единичный вектор дважды. Для каждой промежуточной точки используется функция интерполяции — она высчитывает скалярное произведение векторов четырех ближайших целых точек с вектором смещения промежуточной точки относительно соответствующей целой точки и на их основе высчитывает промежуточное значение с использованием какой-то интерполирующей функции (часто smoothstep).

Данный метод был выбран нашей командой для генерации «высоты» и «температуры», потому что он достаточно эффективен и отлично подходил для этой цели.

2 Главы

2.1 Игровое поле

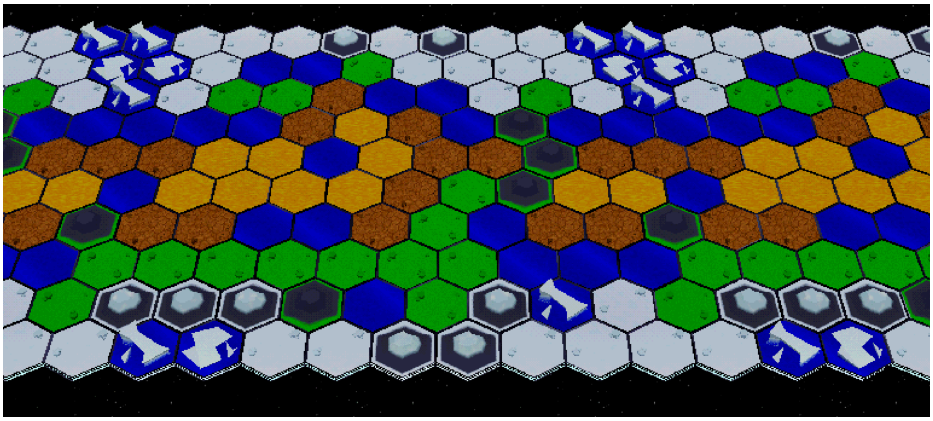

Рисунок 2.1: Сгенерированное игровое поле

Игровое поле состоит из шестиугольных клеток, вместе образующих прямоугольник. В начале игры рельеф генерируется (Рис. 2.1) с использованием смеси алгоритмов Perlin noise [9] и Wave Function Collapse (WFC) [7], а потом на поле равномерно распределяются Командующие.

2.1.1 Алгоритм генерации игрового поля

В первую очередь генерируются три поля градиентных векторов для вычисления трех промежуточных эвристик с помощью Perlin noise: «локальная высота», «глобальная высота» и «локальная температура». Далее с их помощью вводятся еще две составные эвристики: «высота» и «температура». «Высота» в точке (x, y) вычисляется по формуле:

$$h(x, y) = .75 \cdot h_g(x, y) + .25 \cdot h_l(x, y)$$

$h_g(x, y)$ — «глобальная высота с большим размером клетки», $h_l(x, y)$ — «локальная высота»

«Температура» в точке (x, y) вычисляется по формуле:

$$t(x, y) = .75 \cdot t_g(.5 - |y \,/\, \overbrace{height}^{\text{высота поля}} - .5|) + .25 \cdot t_l(x, y)$$

$$t_g(y) = 192y^5 - 240y^4 + 80y^3, \quad t_l(x, y) \text{ — «локальная температура»}$$

«Температура» использует Perlin noise только локально для того, чтобы получить эффект повышения «температуры» ближе к середине поля по вертикали. От «высоты» и

«температуры» зависят вероятности генерации различных типов клеток в последующей фазе WFC, так при увеличении «температуры» вероятность генерации Ледника понижается, а вероятность генерации Пустыни повышается.

В фазе WFC выбирается «неопределенная» клетка, у которой наименьшая «энтропия», которая вычисляется по формуле:

$$ent(c) = -\max_{t \in T} p_t(c) / \sum_{t \in T} p_t(c),$$

где T — возможные типы клеток, а $p_t(c)$ — вероятность клетки c стать типа t согласно «высоте», «температуре» и предикатам, которые используют соседние клетки для определения возможности принятия этой клеткой какого-то типа, так если у «неопределенной» клетки по соседству есть клетка, которая может быть только Пустошью или Пустыней, то эта клетка не может быть Ледником, после выбора клетки она «определяется» — ей присваивается возможный согласно предикатам и вероятностям тип, обновляются возможные типы всех соседних клеток, а также всех клеток, соседних с теми, у которых изменились возможные типы. Алгоритм продолжается, пока все клетки не будут «определены».

2.1.2 Клетка

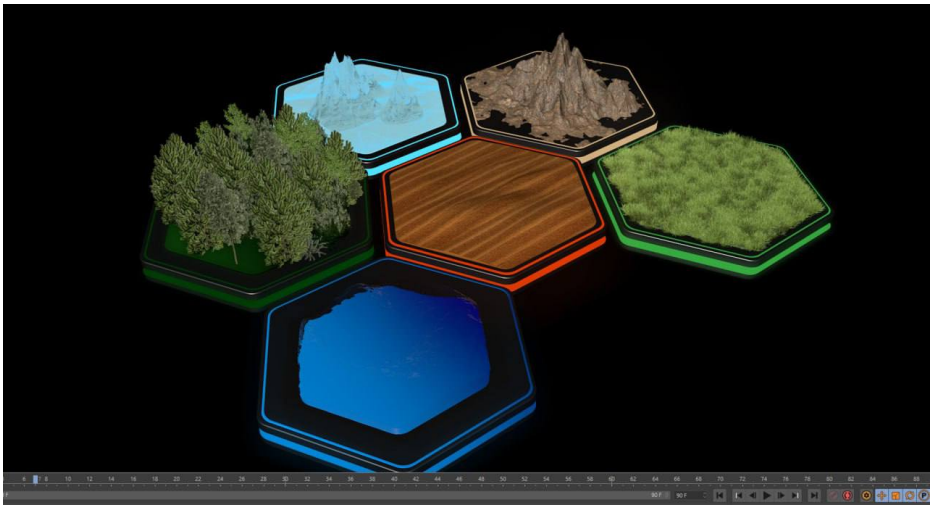

Рисунок 2.2: Первая версия моделей для разных типов клеток

Клетка представляет собой шестиугольную плитку, клетки, которые пересекаются с ней ребром, считаются соседними. На одной клетке может находится один наземный юнит, один воздушный юнит и одно здание. Клетка может быть одним из девяти типов: Вода, Ледник, Рельефный Ледник, Заснеженная равнина, Заснеженная гора, Равнина, Гора, Пустошь и Пустыня (Рис. 2.2) — тип клетки определяет, какие ресурсы добываются на этой клетке,

и как по ней передвигаются юниты.

2.1.3 Туман войны

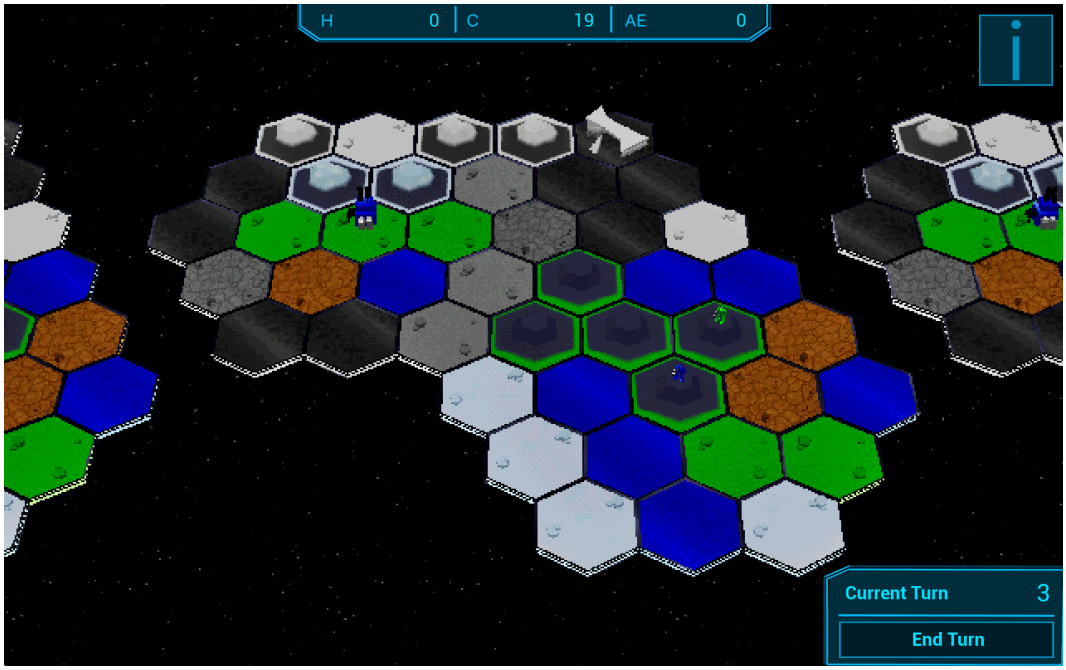

Рисунок 2.3: Клетки в поле зрения цветные, вне поля зрения серые

Игроки видят клетки, находящиеся в поле зрения их юнитов, а также юнитов и здания, которые находятся на этих клетках. Если в данный момент клетка не находится в поле зрения юнитов игрока, но когда-то находилась, то игрок видит серую версию клетки, но не видит юнитов и здания на этой клетке (Рис. 2.3).

2.2 Ход игры

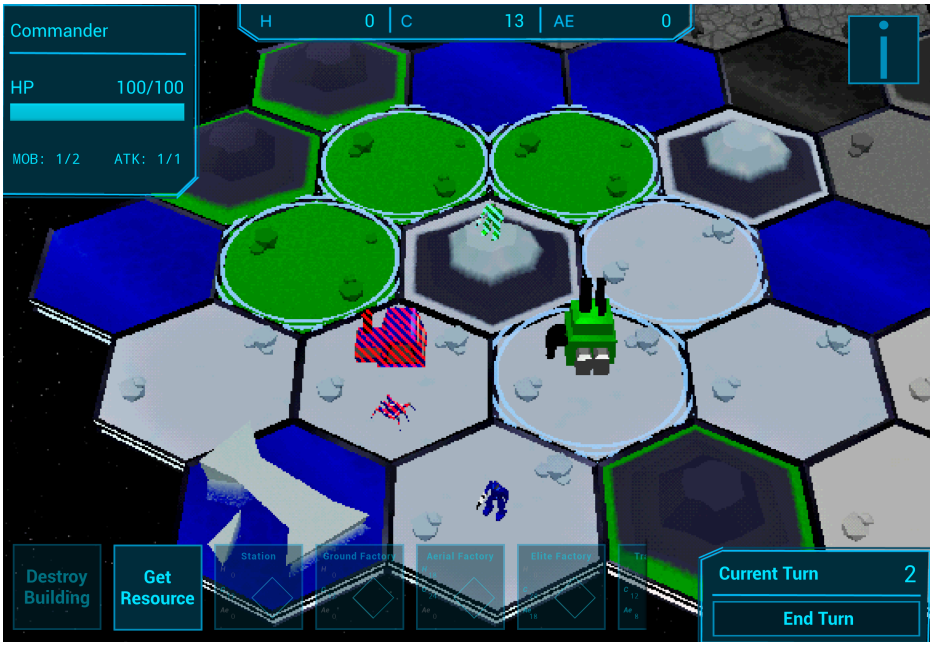

Рисунок 2.4: Клетки доступные для перемещения подсвечиваются голубым, юниты и здания, которые можно атаковать, красным

За один ход игрок может переместиться, атаковать и выполнить уникальную функцию юнита каждым юнитом и выполнить функцию каждого здания. После выбора юнита голубым подсвечиваются клетки, на которые он может переместиться, а юниты и здания, которых он может атаковать, подсвечиваются красным (Рис. 2.4).

Юниты. Все юниты обладают какими-то здоровьем, силой атаки, областью атаки, полем зрения и мобильностью (сколько раз юнит может переместиться за ход). Юниты делятся на наземные (их перемещение ограничено рельефом) и воздушные (могут передвигаться по любым клеткам). И воздушные, и наземные юниты могут быть элитными — их уникальное свойство в том, что они умеют разными способами влиять на время.

В игре реализовано 19 юнитов: 12 наземных, 7 воздушных. Для каждого юнита создана уникальная 3D модель.

Командующий. Командующий — это достаточно мобильный наземный юнит, у него много здоровья и большой урон, он может строить и разрушать здания. Игрок должен защищать своего Командующего любой ценой, потому что, если он погибает, игрок проигрывает.

2.2.1 Здания

Здания строятся Командующим или одним из Инженеров за ресурсы, они не могут перемещаться и не могут напрямую атаковать, они обладают каким-то здоровьем и в их поле зрения попадает только клетка, на которой они стоят. Здания можно построить только на плоской клетке. Реализованы заводы, станции и телепорты, для всех созданы уникальные 3D модели.

Рисунок 2.5: Данная таблица показывает какие ресурсы добывают разные станции на разных клетках

Станции. В начале каждого хода станции добывают ресурсы в зависимости от типа клетки, на которой они стоят. (Рис. 2.5).

Заводы. Заводы бывают 3 разных типов: наземные, воздушные и элитные — они используют ресурсы, чтобы производить юниты; у каждого завода есть какое-то множество юнитов, которых он может создавать.

Транспортер. Юнит, находящийся на клетке с дружественным Транспортером может переместиться на любую другую клетку с дружественным Транспортером, независимо от расстояния между этими клетками.

2.3 Пользовательский интерфейс

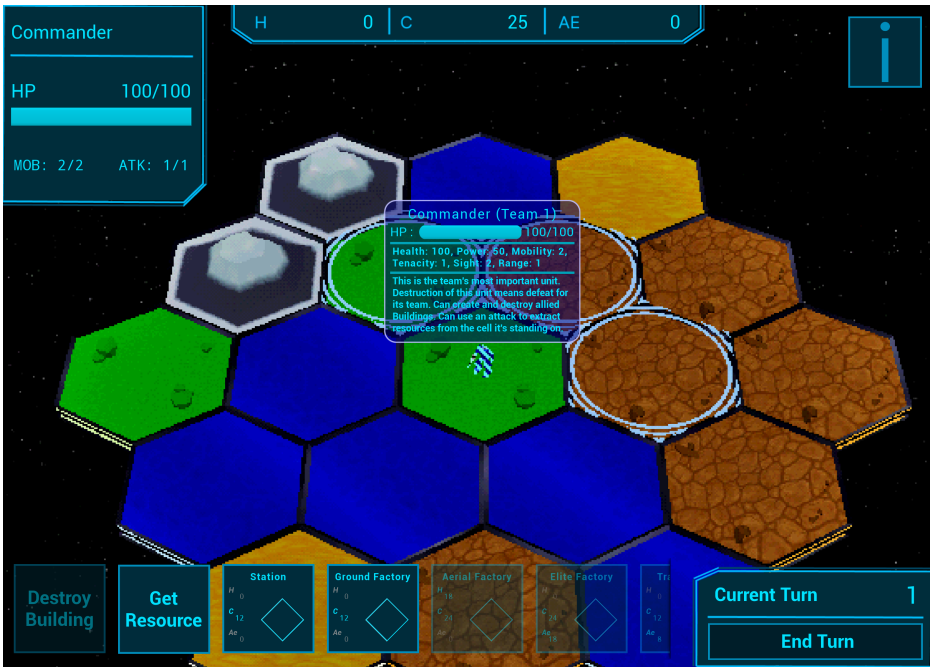

Рисунок 2.6: Пользовательский интерфейс

В игре реализован динамический пользовательский интерфейс (Рис. 2.6) — элементы интерфейса изменяются, появляются и пропадают из-за действий игрока.

Пользовательский интерфейс состоит из 6 основных элементов:

- **Кнопка окончания хода и счетчик хода.**

  При нажатии на эту кнопку в правом нижнем углу ход передается следующему игроку, около нее располагается счетчик, который отображает номер текущего хода.

- **Счетчики ресурсов.**

  Эти счетчики вдоль верхнего края экрана отображают количество каждого ресурса игрока.

- **Информация о выбранном объекте.**

  Этот блок в левом верхнем углу показывает информацию об объекте, который выбран игроком в данный момент.

- **Кнопки действий объекта.**

  Эти кнопки в левом нижнем углу привязаны к действиям выбранного объекта.

- **Информация об объекте под курсором.**

  Этот блок показывает информацию об объекте, на который игрок навелся курсором.

При зажатии правой кнопки мыши, отображается дополнительная информация.

- **Вкладка с информацией**

  При нажатии на кнопку «i», расположенную в правом верхнем углу, появляется окно с подсказками по игре (Рис. 2.7): информация об управлении, правила игры и таблица, показывающая какие ресурсы добывает станция в зависимости от типа клетки.

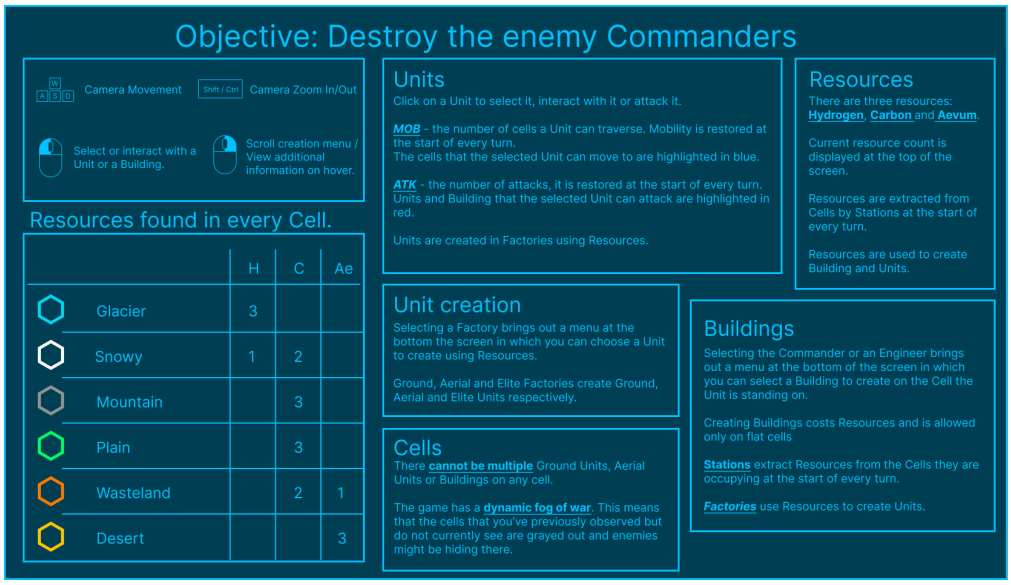

Рисунок 2.7: Окно информации

2.4 Реализация

В этой секции будут описаны методы реализации различных элементов игры, к которым наша команда пришла во время работы над проектом.

2.4.1 Иерархия классов

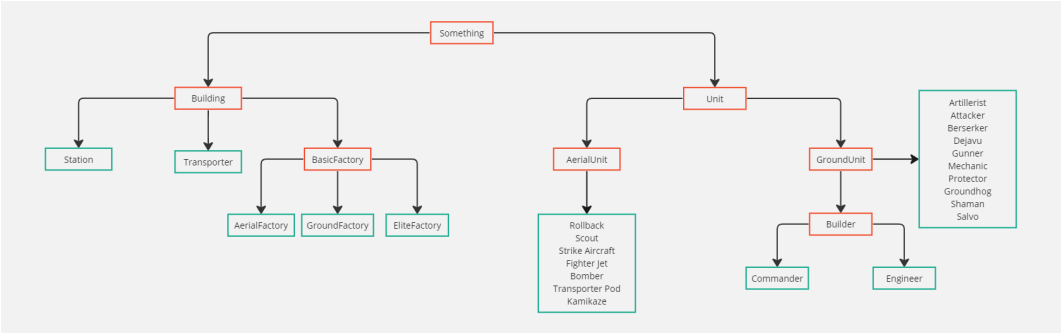

Рисунок 2.8: Иерархия классов. Абстрактные классы красные, финальные — зеленые

Для реализации зданий и юнитов мы активно пользовались наследованием — если два класса что-то объединяло, то мы еще до написания кода выносили этот общий функционал в какой-то абстрактный класс, от которого они впоследствии наследовались, что также позволяло реализовывать полиморфные функции и виртуальные методы, так, например, в классе, который контролирует логику игры, хранятся и передаются в функции указатели только на `ASomething`, который является общим предком всех юнитов и зданий (Рис. 2.8).

2.4.2 Сохранение временной линии

Для обеспечения возможности взаимодействия со временем необходимо иметь возможность сохранять и обращаться ко всем состояниям каждого динамического элемента игры на протяжении всей жизни этого элемента, эта задача относится к классу частично персистентных структур данных, однако из-за легковесности данных ее можно реализовать через обычный стек. Клетки хранят все свои данные (какие юниты и здания находятся на клетке, у скольких юнитов каждой команды эта клетка находится в поле зрения и т. д.) для каждого хода, а юниты и здания хранят данные (здоровье и т. д.) от хода создания до хода разрушения.

Возможно, что подобная реализация использует слишком много памяти, в случае возникновения проблем с памятью можно вместо этого хранить состояния в куче и при изменении состояния добавлять пару (номер хода, новое состояние) в кучу.

2.4.3 Горизонтально «бесконечное» поле

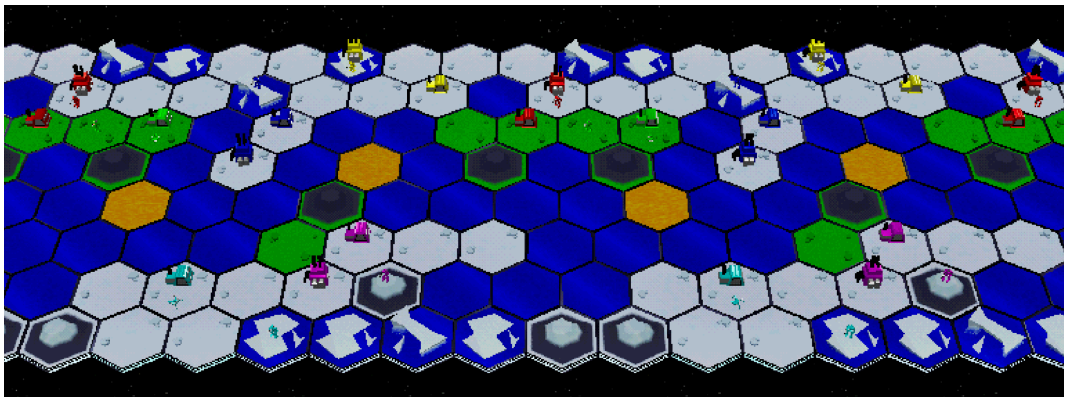

Рисунок 2.9: Можно увидеть одни и те же клетки, здания и юниты несколько раз

Для того, чтобы погрузить игрока в происходящее, мы решили взять возможность обогнуть всю карту горизонтально из серии игр Civilization, для ее реализации мы создаем копию модели каждой клетки, юнита и здания справа и слева от изначальной модели со сдвигом равным длине всего поля, а при преодолении камерой границ центрального поля, телепортируем ее в другой конец этого поля.

Благодаря этому достигается эффект «зацикленности» игрового поля (Рис. 2.9).

2.4.4 Пикселизирующая постобработка

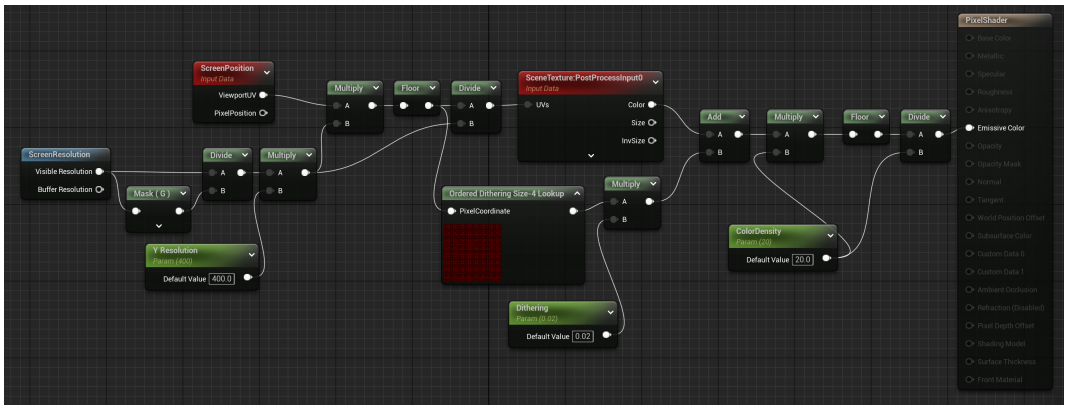

Рисунок 2.10: Схема материала постобработки в Unreal Engine 5

Наша команда решила добавить пикселизирующую постобработку, напоминающую старые игры, где пискелизация была вынужденной мерой для повышения производительности, чтобы создать уникальный художественный стиль и повысить визуальную идентичность игры. Пикселизация была реализована с помощью встроенных в среду Unreal Engine 5 материалов постобработки (Рис. 2.10). На первом этапе постобработки UV-координаты

растягиваются на размер экрана, а потом округляются до вектора кратного размеру пикселя, таким образом создается эффект пикселизации. На втором этапе к пиксилизированной картинке применяется Ordered dithering [10] размера 4, имплементированный в HLSL [11] через таблицу поиска, этот эффект помогает избегать появления резких переходов из цвета в цвет на последнем этапе — квантовании цветов, на котором количество возможных цветов ограничивается до 20^3, как часто делали в старых играх.

2.4.5 3D модели

Для каждой клетки мы создали 3D модели в программе Blender [12].

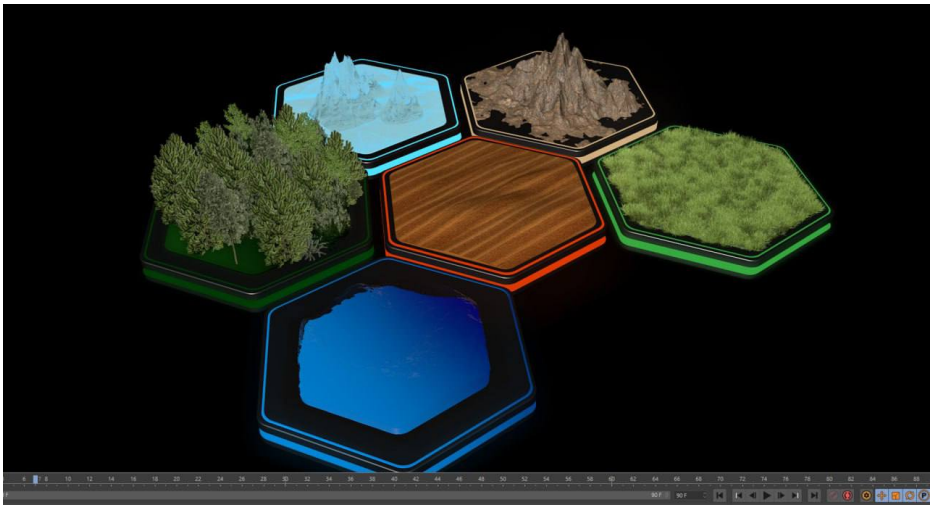

Рисунок 2.11: Первая версия моделей для разных типов клеток

В первой версии нашей игры было 6 типов клеток (Рис. 2.11). и два вида станций: «горячие» и «холодные». В результате плейтестов мы поняли, что «горячие» и «холодные» станции отличаются только типами клеток, на которых они могут быть построены, поэтому мы решили оставить только одну станцию и увеличить количество типов клеток.

Так мы пришли к игре с 15 типами клеток: Пустыня, Пустошь, Джунгли (Рис. 2.12), Равнина, Лес, Заснеженная тундра, Заснеженная тайга, Горы, Горы повыше со снежной шапкой, Заснеженные горы, Заснеженные горы повыше, Плоский ледник, Рельефный ледник и Вода (Рис. 2.13).

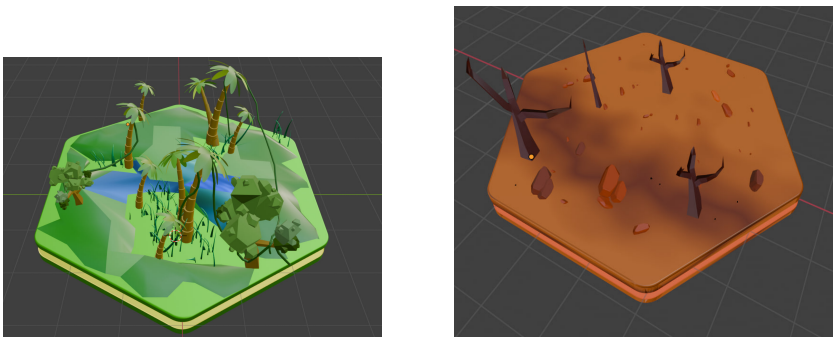

Рисунок 2.12: Джунгли и пустошь

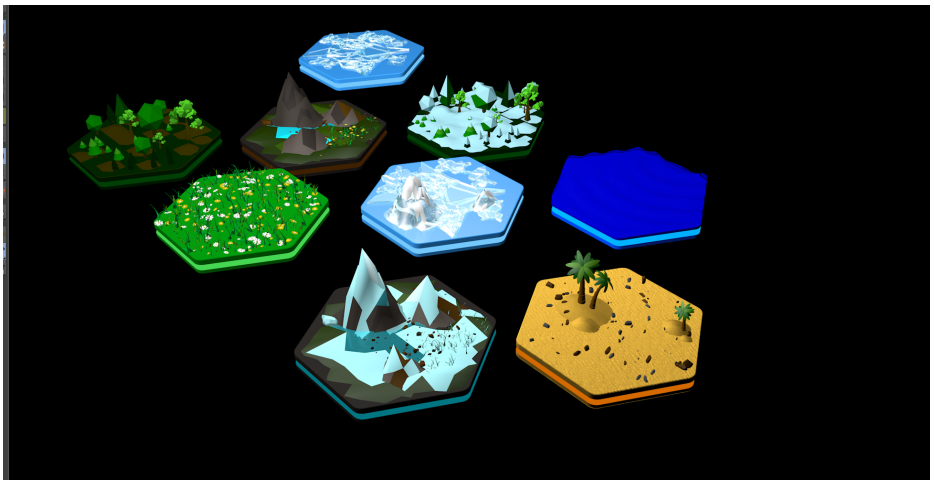

Рисунок 2.13: Вторая версия моделей для разных типов клеток

Появилась проблема с деревьями — они перекрывали соседние клетки, поэтому мы снова изменили типы клеток. Так мы пришли к финальной версии с девятью типами клеток: Вода, Ледник, Рельефный ледник, Заснеженная равнина, Заснеженная гора, Равнина, Гора, Пустошь и Пустыня (Рис. 2.14).

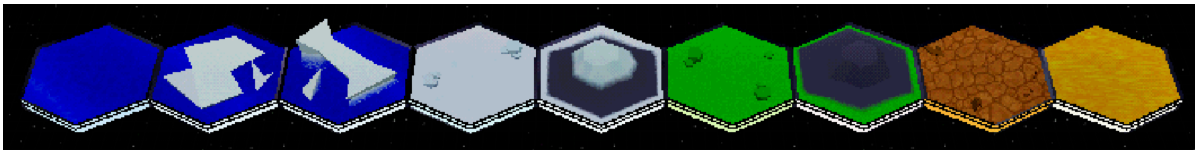

Рисунок 2.14: Итоговые клетки в игре

Также мы создали уникальные 3D модели для зданий и юнитов в Blender, которые принимают цвет команды.

2.4.6 Анимации

В игре реализована система анимаций через вспомогательный класс `AAnimator`, который хранит ассоциативный массив `{id: animation}`. Каждый кадр `AAnimator` пробегает

по этому массиву, обновляя состояние анимаций и, если необходимо, воспроизводя эффекты через встроенную в среду систему Niagara. Если анимация закончилась, `AAnimator` помечает ее законченной и удаляет. Также можно останавливать анимации раньше времени, сообщая `AAnimator id` анимации, которую нужно остановить.

Также реализован класс `AReplayer`, который повторяет все анимации, которые проигрывались во время ходов других игроков, направляя повторные запросы анимаций `AAnimator`. `AReplayer` сохраняет все анимации, которые проигрывает `AAnimator`, в стэк и для каждого хода хранится индекс последней анимации в этом стэке, так, когда начинается новый ход одной из команд, `AReplayer` поступает запрос повторить все анимации с индекса последней анимации последнего хода этой команды.

2.5 Плейтесты

Наша команда провела два плейтеста, чтобы исправить недочеты. Ниже приведены их результаты.

- **Игроки не видят подсказки**

  Изначально подсказки по игре появлялись только в начале, игроки их читали невнимательно и не могли разобраться в интерфейсе игры. После плейтеста мы пересмотрели систему подсказок, и вынесли всю нужную информацию в отдельное окно, к которому игрок может обратиться в любой момент игры

- **Непонятна цель игры**

  Изначально цель игры была указана неявно, и игроки могли только угадать ее. После плейтеста информацию о цели игры мы вынесли в окно с информацией.

- **Игроки не могут добраться друг до друга**

  При каждом запуске игры генирируются различные карты. На плейтесте карта сгенирировалась так, что игроки были отделены друг от друга водой. Они не могли добрать друг до друга наземными юнитами, что значительно усложняло ход игры. Мы изменили генерацию карты, чтобы игроки могли добраться друг до друга.

- **Не понятно какие ресурсы на клетках**

  Игрокам было сложно запомнить какие ресурсы добывают станции на разных клетках. Мы добавили таблицу во вкладку с информацией, которая отображает ресурсы в зависимости от клетки.

- **Игроки не знают где посмотреть характеристики юнитов**

  У всех юнитов разные характеристики. Изначально их можно было посмотреть только в меню завода, который производит этого юнита. Это не было удобно игрокам, поэтому мы добавили возможность смотреть информацию о юните при наведении на него курсором.

- **Баги**

  Благодаря плейтестам мы нашли и исправили следующие баги: исчезновение юнитов при перемещении за границу карты, приложение теряет фокус при нажатии кнопки, сломанный UI.

3 Заключение

Главный результат для нас, конечно, игра, которую мы создали. Однако, по ходу выполнения работы были достигнуты некоторые важные промежуточные результаты:

- Наша команда выбрала профессиональную среду для реализации проекта, поэтому первый месяц из времени, предоставленного на выполнение курсового проекта, был использован на ее освоение.
- Для создания 3D моделей мы выбрали программу Blender, которая тоже требовала изучения.
- Были придуманы несколько вариантов устройства игры и был избран самый лучший, учитывая отзывы игроков.
- Генерация карты использует несколько популярных алгоритмов, которые мы связали и доработали под свои задачи.
- Был приобретен опыт работы в команде относительно разработки игры: отчетность руководителю, созвоны для определения предстоящих дедлайнов и контрольных точек, проведение плейтестов и сбор отзывов от реальных игроков.

Наша команда, наверное, продолжит развивать игру. Вероятно, в будущем мы выложим ее в открытый доступ, будем поддерживать баланс в ней, а также разрабатывать новые обновления (например, добавлять новые станции/юнитов/поля), чтобы удерживать игроков. Кроме того, мы можем добавить мультиплеер, чтобы была возможность играть в любое время со своими друзьями или со случайными игроками.

Список литературы

[1] *Unreal Engine — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Unreal_Engine (дата обр. 26.04.2024).

[2] *Supreme Commander (video game) — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Supreme_Commander_(video_game) (дата обр. 26.04.2024).

[3] *Civilization (series) — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Civilization_(series) (дата обр. 26.04.2024).

[4] *Catan — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Catan (дата обр. 26.04.2024).

[5] Tómas Gudmundsson. «Generating multi player maps through multi objective evolution». В: (2012). URL: https://www2.imm.dtu.dk/pubdb/edoc/imm6474.pdf.

[6] Tristan Wauthier. «Using Wave Function Collapse algorithm for 2D and 3D level generation». В: (2023). URL: https://tristanwauthier.com/PDF/GW_2223_Tristan_Wauthier_EN_Paper.pdf.

[7] Maxim Gumin. *GitHub репозиторий Wave Function Collapse*. URL: https://github.com/mxgmn/WaveFunctionCollapse (дата обр. 26.04.2024).

[8] Thanh Le. «Procedural terrain generation using Perlin noise». В: (2023). URL: https://scholarworks.calstate.edu/downloads/m900p2577.

[9] Ken Perlin. *Noise and Turbulence*. URL: https://mrl.cs.nyu.edu/~perlin/doc/oscar.html#noise (дата обр. 26.04.2024).

[10] *Ordered dithering — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Ordered_dithering (дата обр. 26.04.2024).

[11] *High-Level Shader Language — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/High-Level_Shader_Language (дата обр. 26.04.2024).

[12] *Blender (software) — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Blender_(software) (дата обр. 26.04.2024).