

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Работа допущена к защите

Руководитель

« 25 » 12 2019 г.

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

на тему: «Проектирование и реализация алгоритмов блока редактора
погодных условий имитатора закабинного пространства»

Студент Шорин В.Д.

Шифр 171406

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71-ПГ

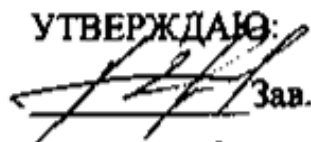
Руководитель Фролов А.И.

Оценка: « отлично » Дата 25.12.19

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»


Кафедра программной инженерии

УТВЕРЖДАЮ:

 Зав. кафедрой
«22» 10 2019 г.

ЗАДАНИЕ
на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Студент  Шорин В.Д.

Шифр 171406

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71ПГ

1.Тема курсовой работы

«Проектирование и реализация алгоритмов блока редактора погодных условий имитатора закабинного пространства»

2.Срок сдачи студентом законченной работы «23» декабря 2019

3. Исходные данные

Техническое задание на «Программный имитатор закабинного пространства»
от 30.08.2018.

Программный имитатор закабинного пространства: Пояснительная записка
УРКТ. 04.08.01-01 81 01 ЛУ

4. Содержание курсовой работы

Назначение и область применения ИЗП

Основные требования к программному обеспечению

Входные и выходные данные ИЗП

Проектирование программной системы имитатора местности

Проектирование блока погодных условий

Диаграмма состояний интерфейса

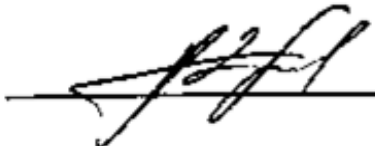
Проектирование алгоритмов блока редактора погодных условий

Реализация Алгоритмов блока редактора погодных условий

Примеры функционирования программного обеспечения

5. Отчетный материал курсовой работы

Пояснительная записка курсовой работы

Руководитель  Фролов А.И.

Задание принял к исполнению: «22» октября 2019

Подпись студента  _____

Содержание

ВВЕДЕНИЕ	5
1 ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	6
1.1 Назначение и область применения	6
1.2 Основные требования к программному обеспечению	7
1.3 Входные и выходные данные.....	10
1.4 Частное задание на разработку	11
2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ ИМИТАТОРА МЕСТНОСТИ	12
2.1 Общая структура	12
2.2 Проектирование блока погодных условий	14
2.3 Диаграмма состояний интерфейса.....	26
3 ПРОЕКТИРОВАНИЕ АЛГОРИТМОВ БЛОКА РЕДАКТОРА ПОГОДНЫХ УСЛОВИЙ	29
3.1 Модуль настройки водной поверхности.....	29
3.2 Модуль настройки снежной поверхности	31
3.3 Модуль настройки погодных условий.....	35
4 РЕАЛИЗАЦИЯ АЛГОРИТМОВ БЛОКА РЕДАКТОРА ПОГОДНЫХ УСЛОВИЙ	38
4.1 Реализация алгоритмов класса WaterScript	38
4.2 Реализация алгоритмов класса SnowTerrainGenerator	39
4.3 Реализация алгоритмов класса weatherMenu	43
4.4 Примеры функционирования программного обеспечения	45
ЗАКЛЮЧЕНИЕ.....	52
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	53
ПРИЛОЖЕНИЕ А(обязательное)ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ.....	54

ВВЕДЕНИЕ

Разработка ведется в рамках составной части НИОКТР: «Разработка программного имитатора закабинного пространства для применения в составе технологического стенда комплексной настройки и проверки комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики».

Наименование изделия: «Программный имитатор закабинного пространства для применения в составе технологического стенда комплексной настройки и проверки комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики», далее – ИЗП.

Изделие является составной частью технологического стенда комплексной настройки и проверки (далее – ТСКН) комплекса для обеспечения поисково-спасательных операций (далее – КОПСО), проводимых с помощью летательных аппаратов в условиях Арктики.

Целью курсовой работы является проектирование и реализация алгоритмов блока редактора погодных условий ИЗП.

Задачами курсовой работы являются:

- 1) определение области применения и назначения ПО;
- 2) определение общих требований к разрабатываемой системе;
- 3) определение входных и выходных данных;
- 4) проектирование общей структуры реализуемого блока;
- 5) создание диаграмм, описывающих разрабатываемое ПО;
- 6) проектирование алгоритмов реализуемого блока;
- 7) определение особенностей реализации.

1 ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Назначение и область применения

В рамках научно-исследовательской, опытно-конструкторской и технологической работ ведётся разработка и создание всепогодного и всесезонного комплекса для обеспечения поисково-спасательных операций (КОПСО), проводимых с помощью летательных аппаратов в условиях Арктики.

КОПСО представляет собой комплекс информационно-измерительных средств, обеспечивающих повышение безопасности полётов вертолётов и информационного обеспечения поисково-спасательных операций, таких как:

- 1) поиск и обнаружение потерпевших бедствие в прибрежных морских районах и на побережье.;
- 2) наведение наземных поисково-спасательных сил на объекты поиска;
- 3) десантирование спасательных групп посадочным способом в сложных метеорологических условиях.

КОПСО включает в себя следующие составные части: автономный источник питания, лазерно-телевизионный модуль, радиолокационную станцию переднего обзора, радиолокационную станцию зондирования подстилающей поверхности и аппаратуру управления и комплексной (АУК) обработки информации.

Для обеспечения проведения комплексной отладки, предварительных и приемочных испытаний опытных образцов КОПСО, а также для проведения приемосдаточных испытаний серийных образцов используется технологический стенд комплексной настройки и проверки (ТСКН) [2].

Процессы комплексной отладки, предварительных и приемочных испытаний опытных образцов КОПСО, приемосдаточных испытаний серийных образцов с использованием ТСКН, сопряжены с необходимостью проведения большого количества тестовых запусков в части отладки и проверки корректности функционирования программного обеспечения

аппаратуры управления и комплексной обработки информации. В условиях отсутствия (до проведения большого количества летных испытаний) достаточного количества исходных данных, получаемых с измерительных блоков КОПСО, необходимо обеспечить генерацию подобных изображений в режиме имитации пролета в определенной местности с заданными параметрами [2].

С целью обеспечения такой возможности в состав ТСКН входит ИЗП (имитатор закабинного пространства или имитатор местности), предназначенный для моделирования измерительной информации (генерации файлов различного формата) от разноспектральных датчиков, входящих в состав аппаратуры КОПСО, при пролете летательного аппарата (ЛА) над участком местности с заданными характеристиками при заданных условиях.

1.2 Основные требования к программному обеспечению

На основе анализа требований к назначению и области применения разработана схема программной системы имитатора закабинного пространства (ИЗП), приведённая на рисунке 1.

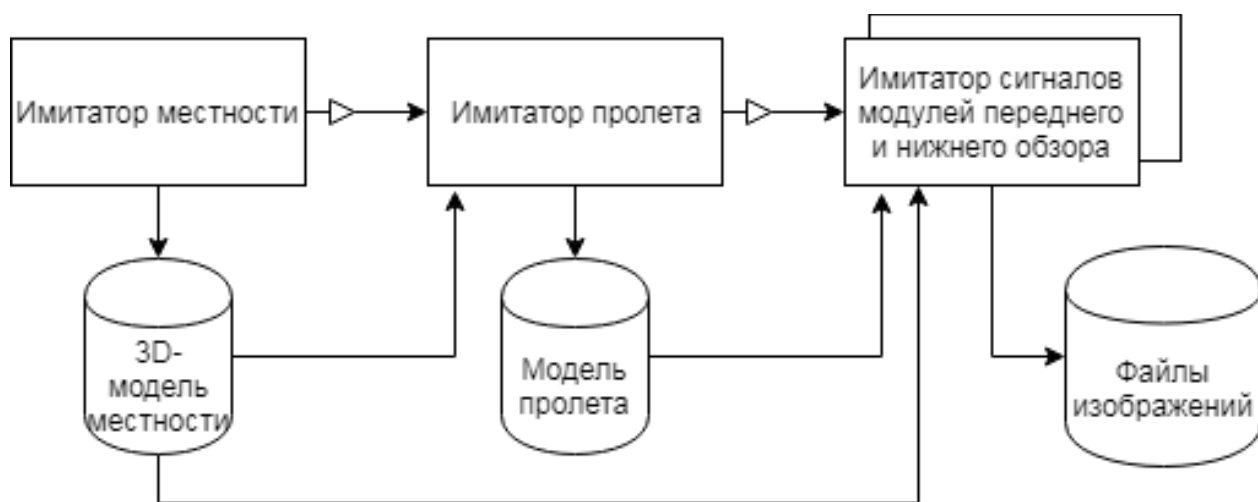


Рисунок 1 – Схема взаимодействия программ программной системы
ИЗП

Имитатор местности с объектами, расположенными не ней, реализующий 3D-модель окружающей обстановки в области наблюдения, должен обеспечить:

- 1) выбор в интерактивном режиме участка местности из набора готовых ландшафтов;
- 2) расположение на подстилающей поверхности и задание параметров различных объектов;
- 3) определение параметров окружающей среды (освещенности и погодных условий, включая температуру, туман, осадки);
- 4) генерацию трехмерной модели окружающей местности.

Исходя из этого, имитатор должен включать в себя редактор окружающей местности.

Редактор представляет собой компьютерную программу, в которой пользователю предоставляется возможность выбирать в интерактивном режиме участок местности из готовых ландшафтов и объекты с определёнными характеристиками [1].

Объект может представлять собой примитив либо композицию примитивов в случае моделирования сложного объекта, например, опоры ЛЭП. В специфике поставленной задачи, высокий уровень детализации не нужен, поэтому достаточно оперировать заданными топологиями объектов.

Основные характеристики примитива объекта:

- 1) форма;
- 2) размеры;
- 3) текстуры поверхности (граней).

Наложение текстур необходимо для дальнейшей генерации изображений (извлечение необходимых моделируемых физических параметров) в различных спектрах. Цвет текстуры (градиент интенсивности) для инфракрасного диапазона задаёт температуру определённой области объекта. Также текстура может иметь альфа-канал (степень прозрачности). Данная характеристика полезна для физических расчётов, связанных с освещённостью.

Объекты находятся на определённых типах подстилающих поверхностей:

- 1) поле;
- 2) лес;
- 3) кустарник;
- 4) болото;
- 5) скальный грунт.
- 6) водная поверхность (характеризуется волнением и глубиной);

Выделяется два вида покрытия подстилающей поверхности:

- 1) ледяная поверхность (характеризуется толщиной);
- 2) снежный покров (характеризуется толщиной).

Подстилающая поверхность тоже представляет собой примитив типа «поверхность» с определенной геометрией (топологией) и с заданной текстурой.

Также пользователь в редакторе может задавать температурные и погодные характеристики окружающего пространства.

Данные от имитатора местности поступают на вход имитатора пролета. Модель местности в имитаторе пролета используется для задания на ней полетного задания. На основании полетного задания и программной физической модели движения ЛА имитатор пролета генерирует модель пролета, содержащую необходимую полетную информацию.

Подсистемы имитации сигналов модулей переднего и нижнего обзора в качестве входной информации получают модель пролета и модель окружающей местности. На основании информации о характеристиках датчиков, их выходных изображениях, точках подвеса датчиков и модели пролета строятся двухмерные представления видимой области модели окружающей местности с учетом условий видимости, и параметров движения генерируются изображения в различных спектрах и сохраняются во внешнюю память.

Полученные серии файлов будут использоваться для настройки, проверки и отладки аппаратуры управления и комплексной обработки информации, а именно алгоритмов и программного обеспечения обработки

данных датчиков. В ходе выполнения работ по настройке, проверке и отладке полученные посредством ИЗП файлы будут последовательно подаваться на вычислительное устройство для обработки для оценки количества и качества входной информации [2].

1.3 Входные и выходные данные

Для блока имитатора местности входными данными являются заданные в интерактивном режиме или считанные из сохраненного файла данные описания сцены в формате XML:

- 1) путь к файлу с моделью ландшафта;
- 2) температуры подстилающих поверхностей:
 - лес: температура поверхности, °C;
 - поле: температура поверхности, °C;
 - кустарник: температура поверхности, °C;
 - болото: температура поверхности, °C;
 - скальный грунт: температура поверхности, °C;
 - водная поверхность: температура поверхности, °C;
 - снежная поверхность: температура поверхности, °C;
 - ледяная поверхность: температура поверхности, °C.
- 3) объекты на сцене:
 - название модели объекта;
 - координаты объекта (X, Z, Y);
 - температура объекта, °C;
 - поворот объекта по оси Y.
4. условия видимости:
 - а) освещение:
 - координаты источника освещения (X, Z);
 - интенсивность освещения, %;
 - тип источника освещения: солнце, луна;
 - б) погодные условия:

- тип осадков (без осадков, туман, дождь, снег);
- интенсивность осадков, %;
- скорость ветра, м/с.

Для остальных блоков ИЗП наборы входных данных также задаются в интерактивном режиме или считываются из сохраненных файлов различных форматов. Выходные данные ИЗП – файлы данных с модулей переднего и нижнего обзора, в том числе и наборы изображений [3].

Выходными данными для блока имитатора местности является сохраненный в формате xml файл сцены.

1.4 Частное задание на разработку

Курсовая работа, в рамках которой проектируются и разрабатываются алгоритмы подсистемы имитатора местности, является комплексной и содержит ряд отдельных подзадач.

В данной курсовой работе будет рассмотрено проектирование и реализация алгоритмов блока редактора погодных условий программной системы ИЗП, позволяющего пользователю программного обеспечения проводить настройку погодных условий и условий освещения требуемой для моделирования сцены.

Для выполнения поставленной задачи и реализации ПО и проектируемых алгоритмов будет использоваться среда Unity, являющаяся межплатформенная средой разработки компьютерных игр, так как она предоставляет множество функциональных возможностей для конечного программного продукта, в том числе моделирование физических сред, карты нормалей, преграждение окружающего света в экранном пространстве, динамические тени и многое другое [5].

В качестве языка программирования будет использован язык C# – один из двух официально существующих вариантов выбора для разработки в среде Unity и одновременно с этим наиболее широко используемый и поддерживаемый язык в указанной среде [4].

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ ИМИТАТОРА МЕСТНОСТИ

2.1 Общая структура

Основными составляющими компонентами имитатора местности являются:

- 1) редактор погодных условий;
- 2) редактор местности;
- 3) интерфейс.

Блок редактор местности позволяет пользователю выбрать требуемую для редактирования местность, произвести настройку расположенных на ней подстилающих поверхностей, выбрать объекты, наличие которых необходимо на моделируемой местности, и разместить их согласно особенностям ландшафта, а также при необходимости сохранить созданный прототип местности или загрузить для использования или дальнейшего редактирования уже имеющийся.

Блок редактора местности можно разделить на следующие модули согласно их функциям и назначению:

- 1) модуль загрузки ландшафта;
- 2) модуль настройки подстилающих поверхностей;
- 3) модуль настройки объектов;
- 4) модуль сохранения и загрузки сцены.

Блок редактора погодных условий предоставляет пользователю возможности по настройке необходимых для имитации условий видимости и погодных условий и внесению изменений в моделируемую местность путем установки параметров для водных поверхностей и определения участков ландшафта, на которых должна располагаться снежная поверхность.

Блок редактора погодных условий включает в себя следующие составляющие модули:

- 1) модуль настройки погодных условий;

- 2) модуль настройки водной поверхности;
- 3) модуль настройки снежной поверхности.

С помощью блока интерфейса пользователь получает доступ к возможностям редактора местности и погодных условий и осуществляет управление процессом редактирования. Блок интерфейса также обеспечивает взаимодействие программы с пользователем в форме диалога.

Основные составляющие блоки и модули программной системы имитатора местности представлены в виде диаграммы компонентов на рисунке 2.

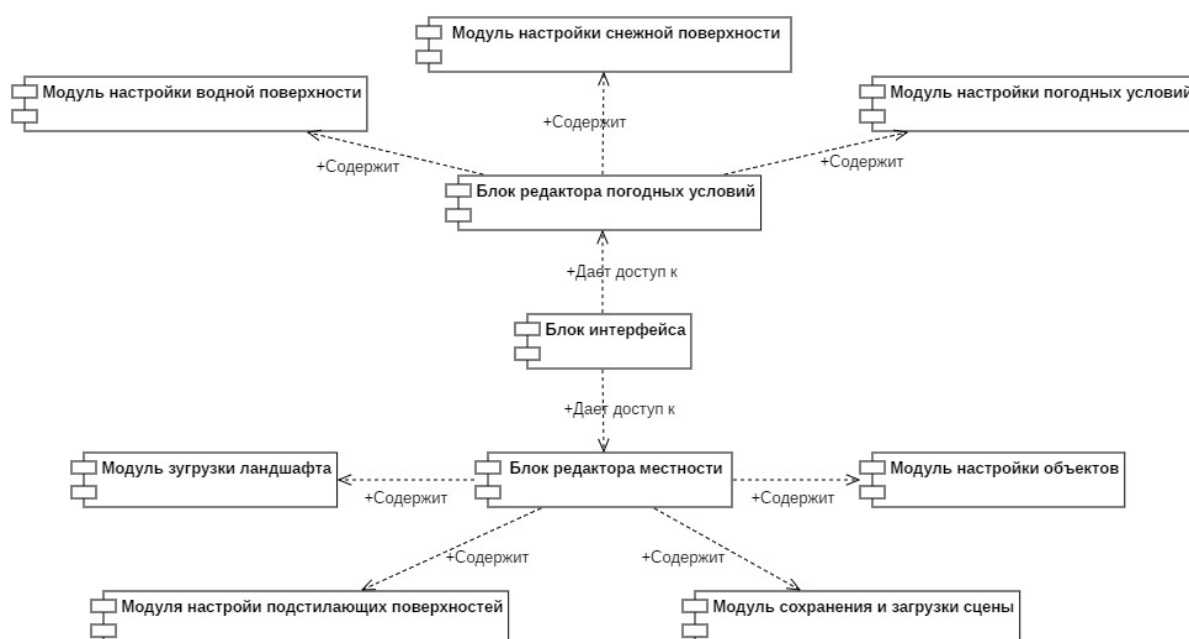


Рисунок 2 – Диаграмма компонентов имитатора местности

2.2 Проектирование блока погодных условий

Ниже представлены диаграммы классов с их описанием для соответствующих модулей блока редактора погодных условий.

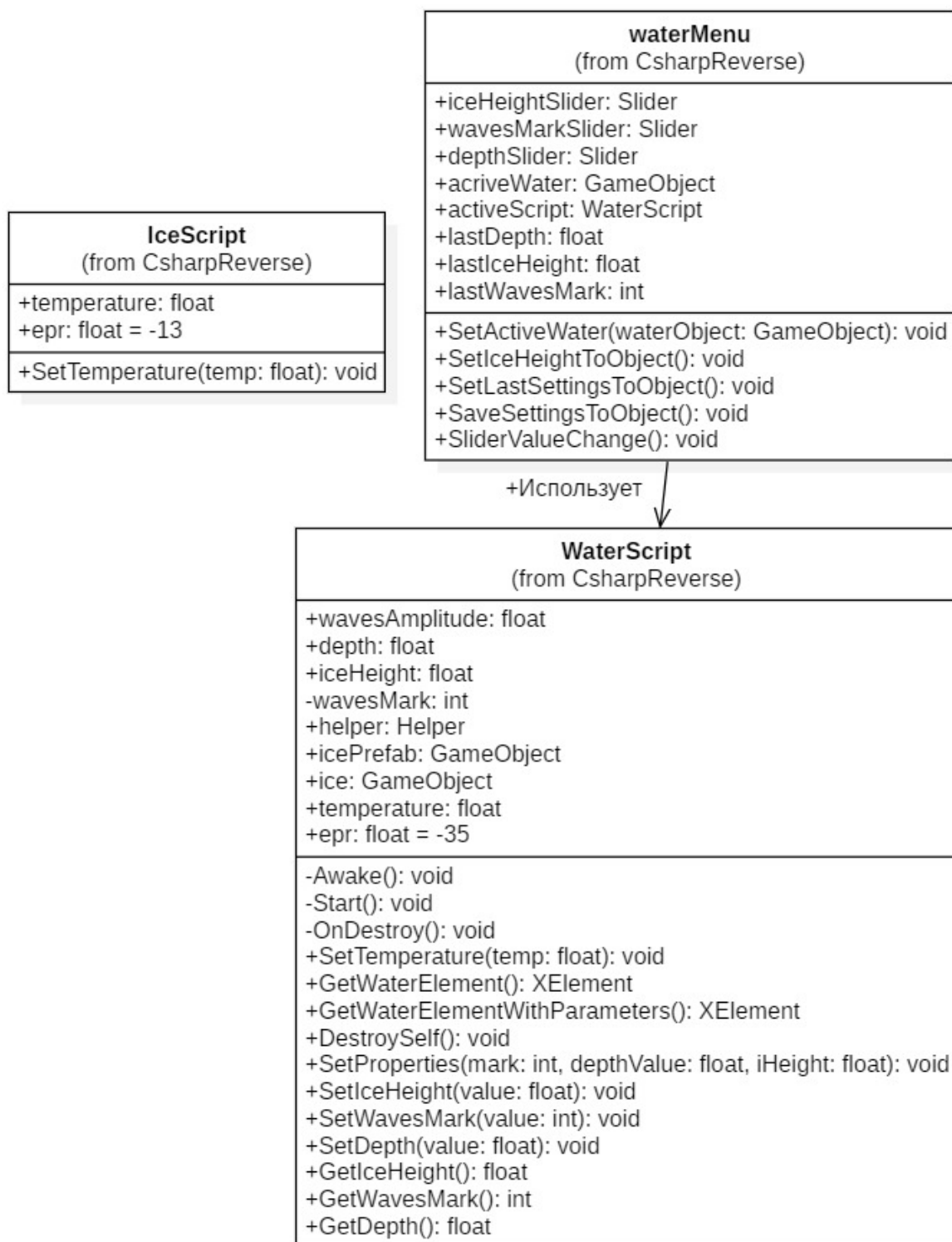


Рисунок 3 – Диаграмма классов модуля настройки водной поверхности

Класс «waterMenu» необходим для связи интерфейса и вводимых пользователем данных с их реализацией.

Содержит следующие поля:

— iceHeightSlider: Slider — содержит ссылку на ползунок, с помощью которого пользователь изменяет значение толщины льда;

— wavesMarkSlider: Slider — содержит ссылку на ползунок, с помощью которого пользователь изменяет значение волнения водного объекта;

— depthSlider: Slider — содержит ссылку на ползунок, с помощью которого пользователь изменяет значение глубины водного объекта;

— acriveWater: GameObject — содержит ссылку на текущий водный объект, с которым работает пользователь;

— activeScript: WaterScript — содержит ссылку на скрипт, связанный с редактируемым водным объектом;

— lastDepth: float — содержит значение глубины водного объекта;

— lastIceHeight: float — содержит значение толщины льда на водном объекте;

— lastWavesMark: int — содержит значение волнения водного объекта.

Содержит следующие методы:

— SetActiveWater(waterObject: GameObject) : void — получает на вход ссылку на текущий водный объект, с которым работает пользователь, получает ссылку на связанный с ним скрипт и устанавливает на интерфейсе в меню «Настройки воды» значения толщины льда, волнения и глубины водного объекта;

— SetIceHeightToObject(): void — устанавливает введенное пользователем значение толщины льда;

— SetLastSettingsToObject(): void — устанавливает старые значения толщины льда, волнения и глубины текущему водному объекту;

— `SaveSettingsToObject(): void` — устанавливает введенные пользователем значения толщины льда, волнения и глубины текущему водному объекту;

— `SliderValueChange(): void` — отслеживает изменения ползунков пользователем и устанавливает новые значения в поля.

Класс «`WaterScript`» служит для управления водным объектом. Содержит следующие поля:

— `wavesAmplitude: float` — хранит устанавливаемое значение амплитуды колебания волн для водного объекта;

— `depth: float` — хранит глубину водного объекта;

— `iceHeight: float` — хранит толщину льда водного объекта;

— `wavesMark: int` — хранит амплитуду колебания волн в баллах;

— `icePrefab: GameObject` — хранит ссылку на объект, который будет использоваться для создания льда;

— `ice: GameObject` — хранит ссылку на созданный объект льда;

— `temperature: float` — хранит значение температуры водного объекта, устанавливаемое пользователем;

— `epr: float = -35` — постоянное значение ЭПР водного объекта;

— `helper: Helper` — переменная для ссылки на скрипт `Helper`.

Содержит следующие методы:

— `Awake(): void` — выполняет действия сразу после инициализации префаба, присваивает полю `helper` ссылку на скрипт `Helper`;

— `Start(): void` — вызывается перед прорисовкой первого фрейма, добавляет текущий водный объект в массив водных объектов;

— `OnDestroy(): void` — Эта функция вызывается для последнего кадра существования объекта (объект может быть уничтожен в ответ на `Object.Destroy` или при закрытии сцены), удаляет текущий водный объект из массива объектов;

— `SetTemperature(temp: float): void` — устанавливает полученное от пользователя значение температуры `temp` в переменную `temperature`;

— `GetWaterElement(): XElement` — возвращает структуру типа `XElement` в которой хранятся данные о координатах и размерах водного объекта;

— `GetWaterElementWithParameters(): XElement` — возвращает структуру типа `XElement` в которой хранятся данные о координатах, толщине льда, волнении и глубине водного объекта;

— `DestroySelf(): void` — уничтожает выбранный водный объект;

— `SetProperties(mark: int, depthValue: float, iHeight: float): void` — устанавливает переменные `wavesAmplitude` (в зависимости от полученного значения `mark`), `depth` в `depthValue` и `iceHeight` в `iHeight`;

— `SetIceHeight(value: float): void` — в зависимости от полученного `value` либо создает объект льда с соответствующим значением толщины, либо уничтожает его (если значение равно 0);

— `SetWavesMark(value: int): void` — в зависимости от полученного значения `value` устанавливает значение переменной `wavesAmplitude`;

— `SetDepth(value: float): void` — в зависимости от полученного значения `value` устанавливает значение переменной `depth`;

— `GetIceHeight(): float` — возвращает текущее значение переменной `iceHeight`;

— `GetWavesMark(): int` — возвращает текущее значение переменной `wavesMark`;

— `GetDepth(): float` — возвращает текущее значение переменной `depth`.

Класс «IceScript» служит для управления параметрами льда.

Содержит следующие поля:

— `temperature: float` — хранит значение температуры льда;

— `ep: float = -13` — хранит значение ЭПР льда.

Содержит следующие методы:

— `SetTemperature(temp: float): void` — в зависимости от полученного значения `temp` устанавливает значение переменной `temperature`.

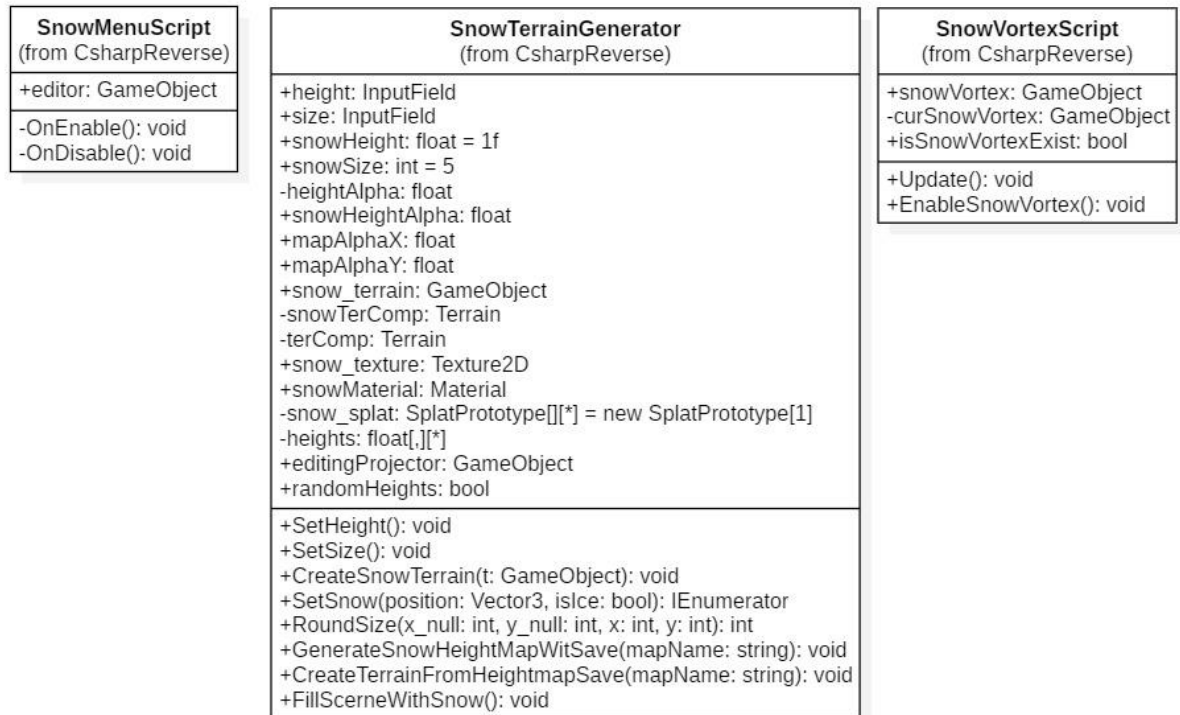


Рисунок 4 – Диаграмма классов модуля настройки снежной поверхности

Класс «SnowMenuScript» служит для связи интерфейса и реализации установки снега (если пользователь выбирает пункт меню «Снег», то начинается режим установки снега. Когда окно установки снега закрывается, возвращается режим редактирования.

Содержит следующие поля:

— `editor: GameObject` — хранит ссылку на объект, содержащий в себе скрипт Editor.

Содержит следующие методы:

— `OnEnable(): void` — Эта функция вызывается только после того, как меню установки снега будет выбрано и устанавливает в скрипте Editor режим установки снега;

— `OnDisable(): void` — Эта функция вызывается, когда меню установки снега закрывается и устанавливает в скрипте Editor режим редактирования.

Класс «SnowTerrainGenerator» служит для генерации и управления снежной поверхностью.

Содержит следующие поля:

— height: InputField — хранит ссылку на поле ввода значения толщины накладываемого снега;

— size: InputField — хранит ссылку на поле ввода значения размера кисти накладываемого снега;

— snowHeight: float = 1f — хранит текущее значение толщины накладываемого снега (по умолчанию 1);

— snowSize: int = 5 — хранит текущее значение размера кисти накладываемого снега (по умолчанию 5);

— heightAlpha: float — хранит отношение 1 к высоте редактируемого ландшафта;

— snowHeightAlpha: float — хранит отношение 1 к высоте ландшафта снега;

— mapAlphaX: float — хранит отношение разрешения карты высот ландшафта снега к его ширине;

— mapAlphaY: float — хранит отношение разрешения карты высот ландшафта снега к его высоте;

— snow_terrain: GameObject — хранит ссылку на ландшафт снега;

— snowTerComp: Terrain — хранит ссылку на компонент Terrain ландшафта снега;

— terComp: Terrain — хранит ссылку на компонент Terrain редактируемого ландшафта;

— snow_texture: Texture2D — хранит ссылку на текстуру снега;

— snowMaterial: Material — хранит ссылку на материал снега;

— snow_splat: SplatPrototype[][*] = new SplatPrototype[1] — массив текстур, используемых TerrainData;

— heights: float[][*] — массив высот ландшафта снега;

— `editingProjector: GameObject` — ссылка на объект, изображающий кисть нанесения снега;

— `randomHeights: bool` — проверка на наличие случайных высот.

Содержит следующие методы:

— `SetHeight(): void` — в зависимости от введенного пользователем значения высоты в поле `height` устанавливает значение переменной `snowHeight`;

— `SetSize(): void` — в зависимости от введенного пользователем значения размера кисти нанесения снега в поле `size` устанавливает значение переменной `snowSize` и размер кисти `editingProjector`;

— `CreateSnowTerrain (t: GameObject): void` — создает ландшафт снега;

— `SetSnow(position: Vector3, isIce: bool): IEnumerator` — функция, отвечающая за нанесение снега;

— `RoundSize (x_null: int, y_null: int, x: int, y: int): int` — возвращает размер кисти нанесения снега;

— `GenerateSnowHeightMapWitSave (mapName: string): void` — сохраняет карту высот снега для последующей загрузки;

— `CreateTerrainFromHeightmapSave (mapName: string): void` — создает ландшафт снега из сохраненного файла.

Класс «`SnowVortexScript`» служит для управления завихрением снега.

Содержит следующие поля:

— `snowVortex: GameObject` — хранит ссылку на объект, реализующий завихрение снега;

— `curSnowVortex: GameObject` — хранит ссылку на текущий созданный объект, реализующий завихрение снега;

— `isSnowVortexExist: bool` — служит для проверки на существование завихрения снега.

Методы:

— Update(): void — вызывается один раз за кадр и вызывает функцию EnableSnowVortex();

— EnableSnowVortex(): void — служит для создания, управления и удаления завихрения снега.

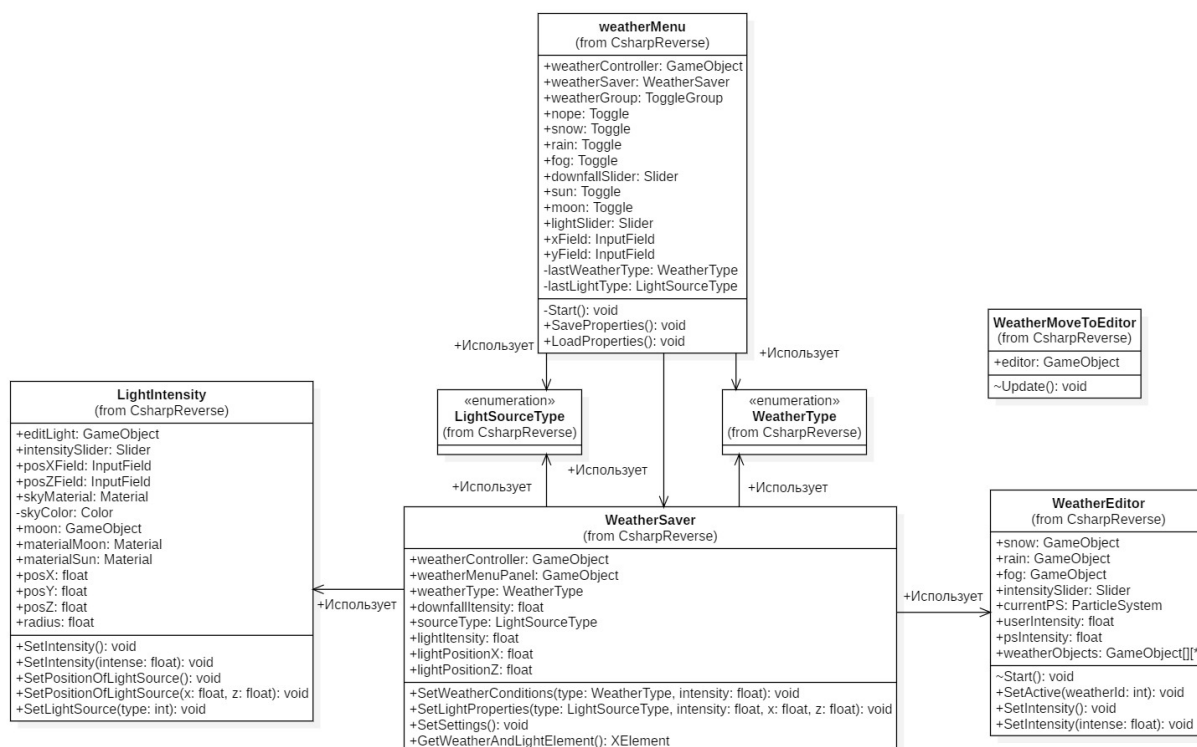


Рисунок 5 – Диаграмма классов модуля настройки погодных условий

Класс «weatherMenu» используется для управления меню настроек условий видимости. Позволяет загружать в поля меню значения, загруженные из файла и получать введенные пользователем новые значения.

Содержит поля:

— weatherController: GameObject — содержит ссылку на объект сцены, который хранит ссылку на скрипт WeatherSaver;

— weatherSaver: WeatherSaver — содержит ссылку на скрипт WeatherSaver;

— weatherGroup: ToggleGroup — содержит ссылку на группу переключателей управления погодными условиями;

— nope: Toggle — хранит ссылку на переключатель «Нет осадков»;

— snow: Toggle — хранит ссылку на переключатель «Снег»;

- rain: Toggle — хранит ссылку на переключатель «Дождь»;
- fog: Toggle — хранит ссылку на переключатель «Туман»;
- downfallSlider: Slider — хранит ссылку на ползунок «Интенсивность осадков»;
- sun: Toggle — хранит ссылку на переключатель «Солнце»;
- moon: Toggle — хранит ссылку на переключатель «Луна»;
- lightSlider: Slider — хранит ссылку на ползунок «Интенсивность освещения»;
- xField: InputField — хранит ссылку на X-координату источника света;
- yField: InputField — хранит ссылку на Y-координату;
- lastWeatherType: WeatherType — хранит значение типа погодных условий, загруженное по умолчанию;
- lastLightType: LightSourceType — хранит значение типа источника цвета, загруженное по умолчанию.

Содержит следующие методы:

- Start(): void — при начале работы скрипта получает ссылку на скрипт WeatherSaver и устанавливает ее в переменную weatherSaver, а также устанавливает переменную lastWeatherType в none (т.е. «Нет осадков») и lastLightType в «sun» («Солнце»);
- SaveProperties(): void — проверяет все элементы меню настройки условий видимости на изменение и записывает новые значения в переменные для последующего сохранения их в файл;
- LoadProperties(): void — загружает из файла значения переменных и переключателей в скрипт и устанавливает в соответствии с ними контекст интерфейса.

Класс «WeatherSaver» — получает переменные из скрипта weatherMenu и сохраняют их в файл, а также вызывает функции из скриптов WeatherEditor и LightIntensity для установки значений, переключателей и ползунков в соответствии с загруженными из файла параметрами.

Содержит поля:

- `weatherController: GameObject` — содержит ссылку на объект сцены, который хранит ссылку на скрипт `WeatherSaver`;
- `weatherMenuPanel: GameObject` — хранит ссылку на панель меню настройки условий видимости;
- `downfallIntensity: float` — хранит значение интенсивности осадков;
- `lightIntensity: float` — хранит значение интенсивности освещения;
- `lightPositionX: float` — хранит X-координату источника света;
- `lightPositionZ: float` — хранит Y-координату источника света;
- `weatherType: WeatherType` — хранит значение типа погодных условий;
- `sourceType: LightSourceType` — хранит значение типа источника света.

Содержит следующие методы:

- `SetWeatherConditions(type: WeatherType, intensity: float): void` — устанавливает значения типа погодных условий (`weatherType`) и интенсивности осадков (`downfallIntensity`) в соответствии с входными параметрами;
- `SetLightProperties(type: LightSourceType, intensity: float, x: float, z: float): void` — устанавливает значения типа источника света (`sourceType`), интенсивности освещения (`lightIntensity`) и координат X (`lightPositionX`) и Y (`lightPositionZ`) источника освещения соответственно;
- `SetSettings(): void` — вызывает функции установки параметров погодных условий и освещения из скриптов `WeatherEditor` и `LightIntensity`;
- `GetWeatherAndLightElement(): XElement` — сохраняет значения переменных, ползунков и переключателей в файл.

Класс «`LightIntensity`» служит для управления настройками освещения;

Содержит поля:

- `editLight: GameObject` — хранит ссылку на объект источника света;

— `intensitySlider: Slider` — хранит ссылку на ползунок настройки интенсивности освещения;

— `posXField: InputField` — хранит ссылку на поле ввода X-координаты источника освещения;

— `posZField: InputField` — хранит ссылку на поле ввода Y-координаты источника освещения;

— `skyMaterial: Material` — хранит ссылку на материал неба;

— `skyColor: Color` — хранит значение цвета неба;

— `moon: GameObject` — хранит ссылку на объект Луны;

— `materialMoon: Material` — хранит ссылку на материал Луны;

— `materialSun: Material` — хранит ссылку на материал Солнца;

— `posX: float` — хранит X-координату источника освещения;

— `posY: float` — хранит Y-координату источника освещения;

— `posZ: float` — хранит Z-координату источника освещения;

— `radius: float` — хранит радиус окружности расположения источника освещения.

Содержит следующие методы:

— `SetIntensity(): void` — устанавливает настройки освещения по умолчанию;

— `SetIntensity(intense: float): void` — устанавливает настройки освещения в соответствии с входным параметром `intense`;

— `SetPositionOfLightSource(): void` — вычисляет и устанавливает позицию источника освещения по умолчанию;

— `SetPositionOfLightSource(x:float, z:float): void` — вычисляет и устанавливает позицию источника освещения в соответствии с входными параметрами;

— `SetLightSource(type: int): void` — устанавливает тип источника освещения.

Класс «WeatherEditor» служит для управления настройками погодных условий.

Содержит поля:

— snow: GameObject — хранит ссылку на объект, реализующий снегопад;

— rain: GameObject — хранит ссылку на объект, реализующий дождь;

— fog: GameObject — хранит ссылку на объект, реализующий туман;

— intensitySlider: Slider — хранит ссылку на ползунок настройки интенсивности освещения;

— currentPS: ParticleSystem — хранит ссылку на погодные условия, установленные в данный момент;

— userIntensity: float — хранит значение интенсивности, введенное пользователем;

— psIntensity: float — хранит значение интенсивности объекты;

— weatherObjects: GameObject[] — хранит массив объектов погодных условий.

Содержит следующие методы:

— Start(): void — при старте работы скрипта определяет значения массива weatherObjects;

— SetActive(weatherId: int): void — в соответствии со входным параметром weatherId устанавливает тип источника освещения;

— SetIntensity(): void — устанавливает значение интенсивности по умолчанию;

— SetIntensity(intense: float): void — устанавливает значение интенсивности в зависимости от входного параметра intense.

Класс «WeatherMoveToEditor» служит для установки позиции погодных условий в соответствии с позицией игрока.

Содержит поля:

— editor: GameObject — хранит ссылку на объект редактора.

Содержит следующие методы:

— Update(): void — каждый кадр обновляет позицию погодных условий в соответствии с позицией редактора.

Перечисление «LightSourceType» содержит типы источника освещения (Солнце и Луна).

Перечисление «WeatherType» содержит типы погодных условий (Нет погодных условий, снег, дождь, туман).

2.3 Диаграмма состояний интерфейса

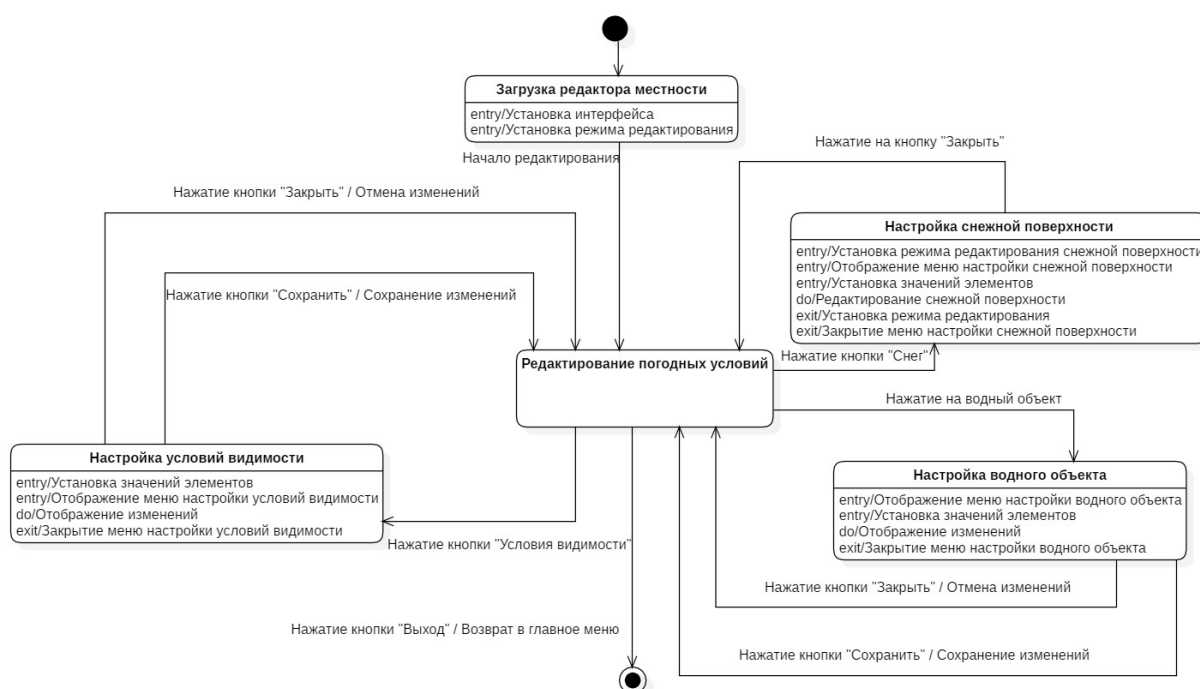


Рисунок 6 — Диаграмма состояний интерфейса

На рисунке 6 представлена диаграмма состояний интерфейса. Представим описание данной схемы.

При выборе пункта «Редактор местности» главного меню система попадает в состояние загрузки редактора местности, в котором происходит установка интерфейса, где пользователь может выбрать нужные ему действия, щелкнув по соответствующему пункту меню, и установка режима редактирования.

При нажатии кнопки «Снег» система попадает в состояние настройки снежной поверхности. При входе в это состояние происходит отображение меню настройки снежной поверхности, установка значений элементов, а также режим редактирования заменяется на режим редактирования снежной поверхности, в котором пользователь может изменять снежную поверхность посредством изменения значений толщины слоя и радиуса кисти в соответствующих пунктах меню. При нажатии кнопки «Заккрыть» происходит установка режима редактирования вместо режима редактирования снега, а также закрытие меню настройки снежной поверхности.

При нажатии на любой водный объект система переходит в состояние настройки водного объекта и происходит отображение меню настройки водного объекта и установка значений элементов. В данном состоянии пользователь может изменять такие параметры водного объекта, как: волнение, глубина и толщина льда. При нажатии кнопки «Сохранить» система переходит в состояние редактирования погодных условий, происходит сохранение всех измененных параметров и закрытие меню настройки водного объекта. При нажатии на кнопку «Заккрыть» система переходит в состояние редактирования погодных условий, происходит отмена всех изменений, возврат их в исходное состояние и закрытие меню настройки водного объекта.

При нажатии на кнопку «Условия видимости» система переходит в состояние настройки условий видимости и происходит отображение меню настройки условий видимости и установка значений элементов. Пользователь может изменять такие параметры, как: тип погодных условий, интенсивность осадков, тип источника света, интенсивность освещения и координаты источника света. При нажатии кнопки «Сохранить» система переходит в состояние редактирования погодных условий, происходит сохранение всех измененных параметров и закрытие меню настройки условий видимости. При нажатии на кнопку «Заккрыть» система переходит в состояние редактирования погодных условий, происходит отмена всех изменений, возврат их в исходное состояние и закрытие меню настройки условий видимости.

При нажатии кнопки «Выход» происходит закрытие сцены редактирования и возврат в главное меню.

3 ПРОЕКТИРОВАНИЕ АЛГОРИТМОВ БЛОКА РЕДАКТОРА ПОГОДНЫХ УСЛОВИЙ

Блок редактора погодных условий состоит из следующих модулей:

- Модуль настройки водной поверхности;
- Модуль настройки снежной поверхности;
- Модуль настройки погодных условий.

Рассмотрим подробнее процесс проектирования каждого из них.

3.1 Модуль настройки водной поверхности

В данном модуле наибольший интерес при проектировании представляют методы «SetIceHeight» и «SetProperties» класса «WaterScript».

Метод «SetIceHeight» реализует алгоритм установки высоты ледяного слоя на водном объекте. Суть алгоритма в следующем: на вход алгоритму подается значение устанавливаемой высоты льда и, в зависимости от него, устанавливается или удаляется объект льда. Общая схема данного алгоритма представлена на рисунке 7.



Рисунок 7 — Схема установки высоты льда

Метод «SetProperties» реализует алгоритм установки свойств водного объекта. Суть алгоритма в следующем: на вход алгоритму подаются все

свойства водного объекта (волнение (в баллах), глубина и высота льда). Далее, в зависимости от значения волнения, устанавливается амплитуда колебания волн. Общая схема данного алгоритма представлена на рисунке 8.

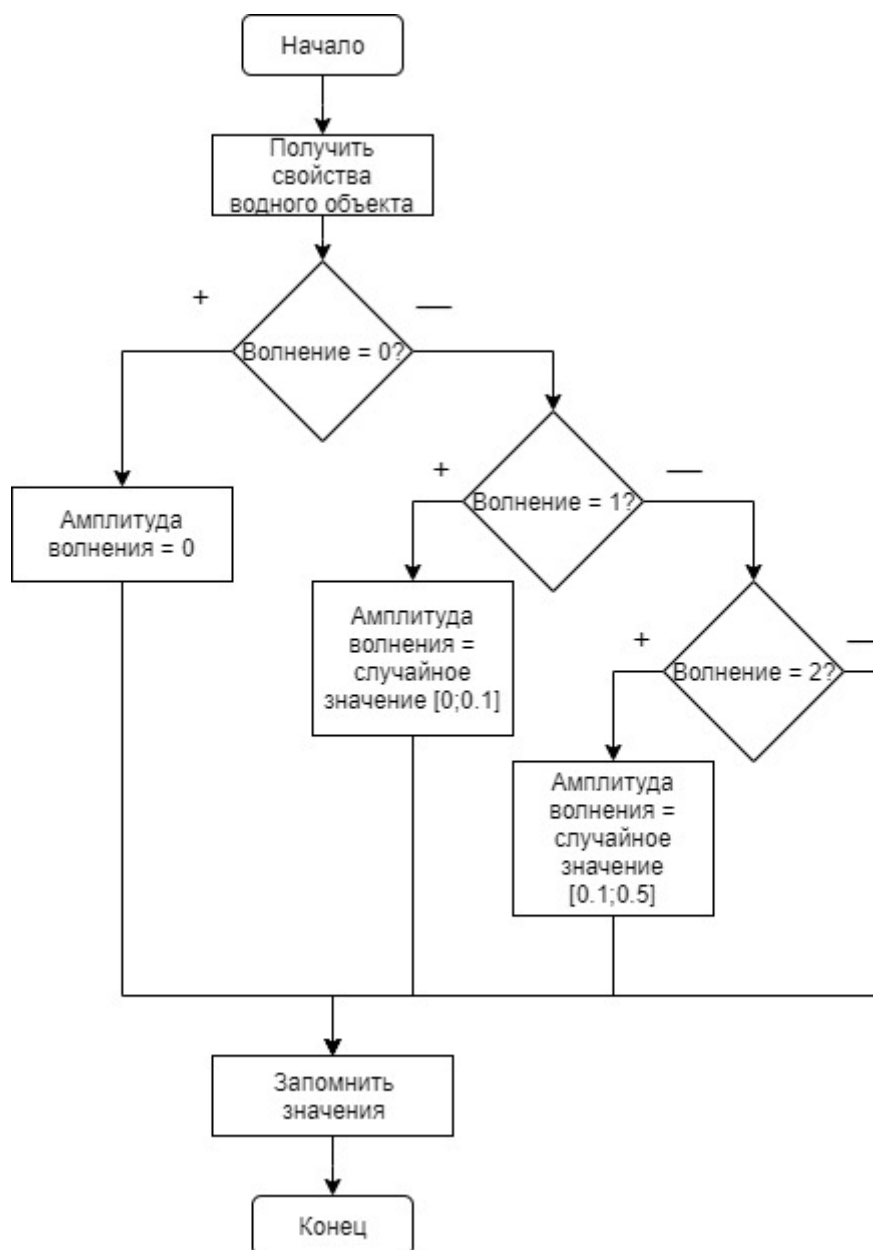


Рисунок 8 — Схема установки свойств водного объекта

Остальные методы не нуждаются в подробном проектировании в силу своей тривиальности (их суть сводится либо к простому установлению переданных параметров в свойства текущего класса, либо к возврату данных свойств, либо к другим простым действиям).

3.2 Модуль настройки снежной поверхности

В данном модуле наибольший интерес при проектировании представляют методы «SetSnow», «GenerateSnowHeightMapWitSave» и «CreateTerrainFromHeightmapSave» скрипта «SnowTerrainGenerator».

Метод «SetSnow» реализует алгоритм установки снега на поверхность. В общем виде данный алгоритм описывает следующая последовательность действий:

1. Установка размеров кисти нанесения снега и позиции.
2. Создать пустую карту высот снега.
3. В двойном вложенном цикле, соответствующем квадратной карте

высот, выполнять:

3.1. Если результат суммы квадратов разности значений итераторов и изначальных значений меньше квадрата введенной высоты снега, выполнять:

3.1.1. Вычислить очередную точку, в которой необходимо изменить высоту снега.

3.1.2. Если введенное значение высоты снега больше нуля:

3.1.2.1. Нанести снег.

3.1.3. Если введенное значение высоты снега равно нулю:

3.1.3.1. Удалить снег.

4. Установить карту высот снега.

Общая схема данного алгоритма представлена на рисунке 9.

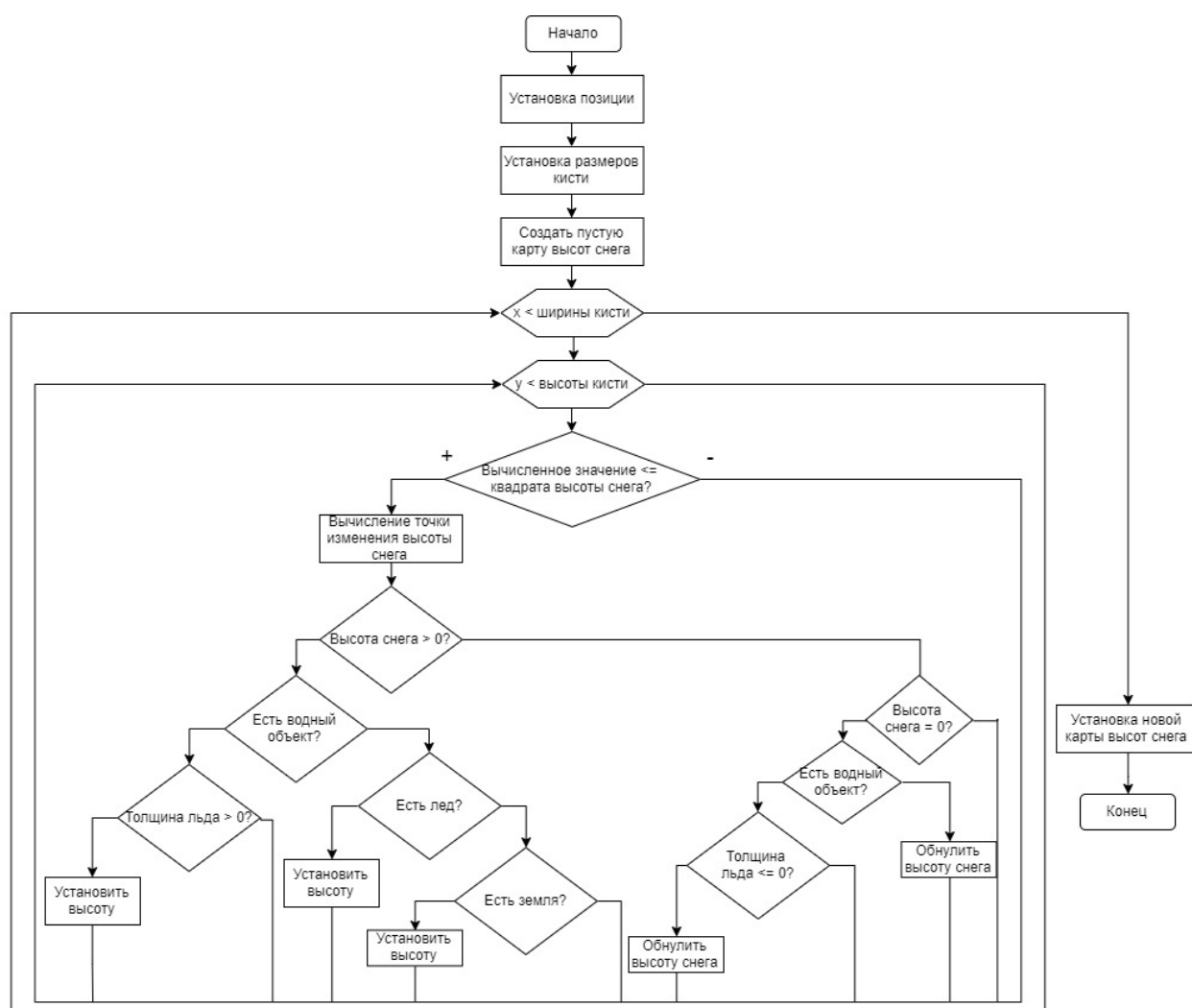


Рисунок 9 — Схема установки снега на поверхность

Метод «GenerateSnowHeightMapWitSave» реализует алгоритм генерации карты высот с последующим ее сохранением. В общем виде данный алгоритм описывается следующей последовательностью действий:

1. Установка значений высоты и ширины карты высот снега.
2. Создание пустой текстуры для карты высот снега.
3. Создание пустой карты высот снега.
4. В двойном вложенном цикле, соответствующем ширине и высоте квадратной карты высот снега, выполнять:
 - 4.1. Получение разницы между высотой снега и высотой ландшафта в текущей точке.
 - 4.2. Если разница меньше или равна нулю:
 - 4.2.1. Установить разницу в ноль.

- 4.2.2. Установить соответствующий текстель.
 - 4.3. Если разница больше нуля:
 - 4.3.1. Установить соответствующий текстель.
 5. Создать png-файл, соответствующий карте высот снега.
 6. Сохранить его в системе.
- Общая схема данного алгоритма представлена на рисунке 10.

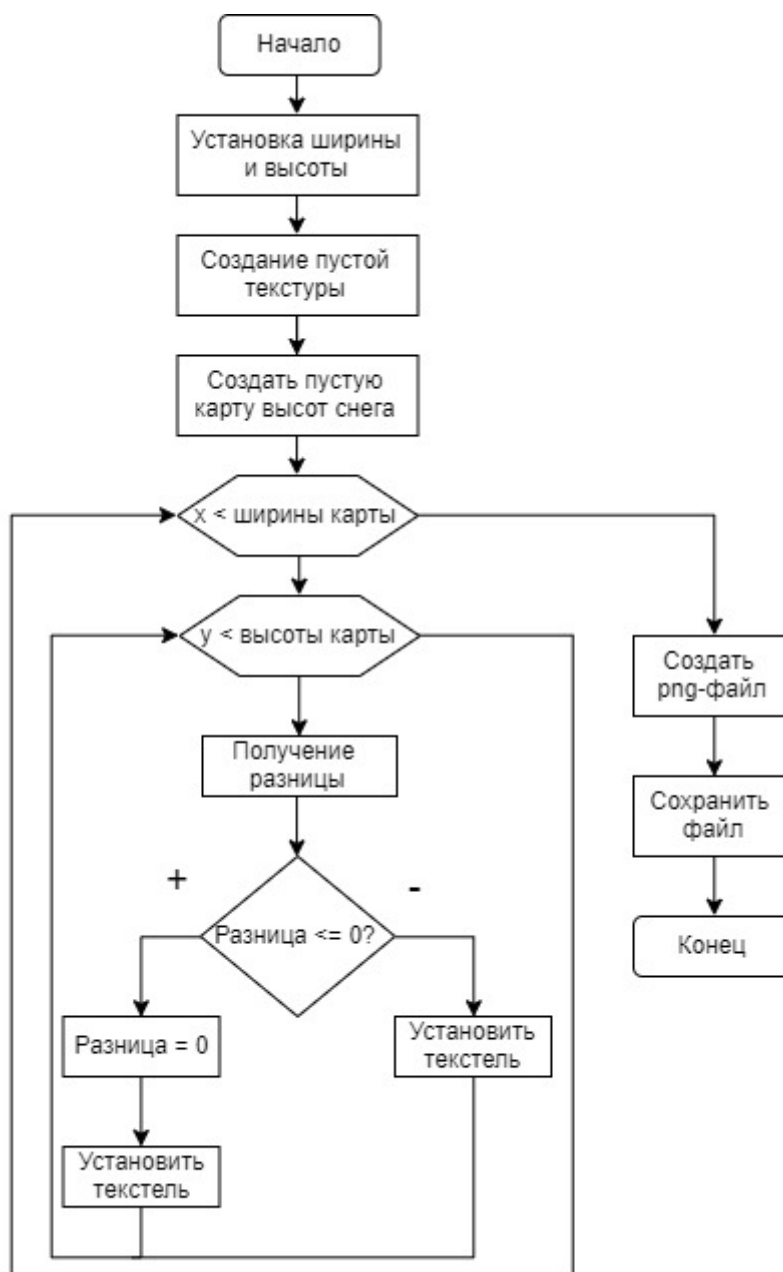


Рисунок 10 — Схема генерации карты высот с последующим ее сохранением

Метод «CreateTerrainFromHeightmapSave» реализует алгоритм создания карты высот снега и наложения ее на ландшафт из изображения карты высот.

В общем виде данный алгоритм описывает следующая последовательность действий:

1. Получить ссылку на ландшафт снега.
2. Получить изображение карты высот снега.
3. Создать пустую карту высот снега.
4. В двойном вложенном цикле, соответствующем ширине и высоте

квадратной карты высот снега, выполнять:

- 4.1. Если разница равна нулю, присвоить карте высот ноль.

Иначе, присвоить вычислить новую высоту с учетом этой разницы.

5. Установить загруженную карту высот.

Общая схема данного алгоритма представлена на рисунке 11.

Остальные методы не нуждаются в подробном проектировании в силу своей тривиальности (их суть сводится либо к простому установлению переданных параметров в свойства текущего класса, либо к возврату данных свойств, либо к другим простым действиям).

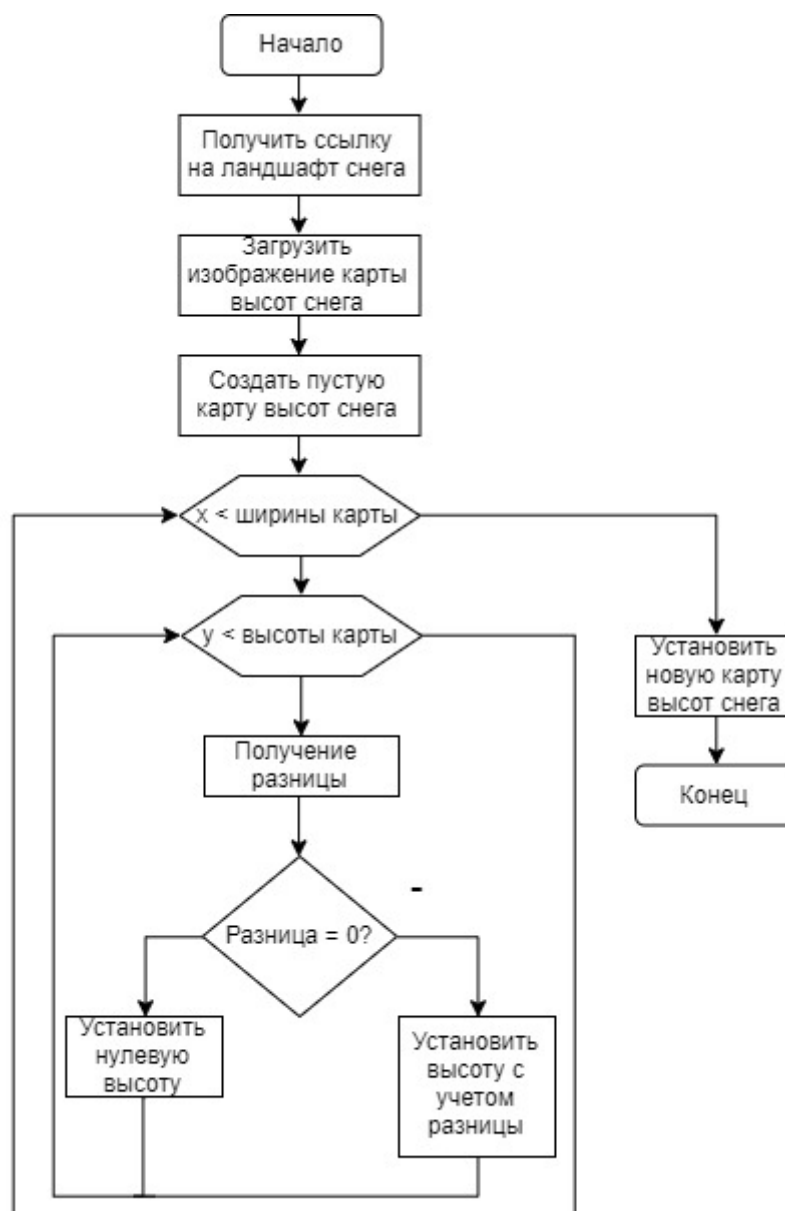


Рисунок 11 — Схема создания ландшафта из карты высот

3.3 Модуль настройки погодных условий

В данном модуле наибольший интерес при проектировании представляет метод «LoadProperties» скрипта «weatherMenu».

Метод «LoadProperties» реализует алгоритм загрузки и установления погодных условий. В общем виде данный алгоритм описывает следующая последовательность действий:

1. Получение загруженных значений.
2. Установка типа осадков в соответствии с загруженным значением.

3. Установка значений интенсивности осадков и освещения в соответствии с загруженными значениями.

4. Установка типа источника освещения в соответствии с загруженным значением.

5. Установка координат источника освещения в соответствии с загруженным значением.

Общая схема данного алгоритма представлена на рисунке 12.

Остальные методы не нуждаются в подробном проектировании в силу своей тривиальности (их суть сводится либо к простому установлению переданных параметров в свойства текущего класса, либо к возврату данных свойств, либо к другим простым действиям).

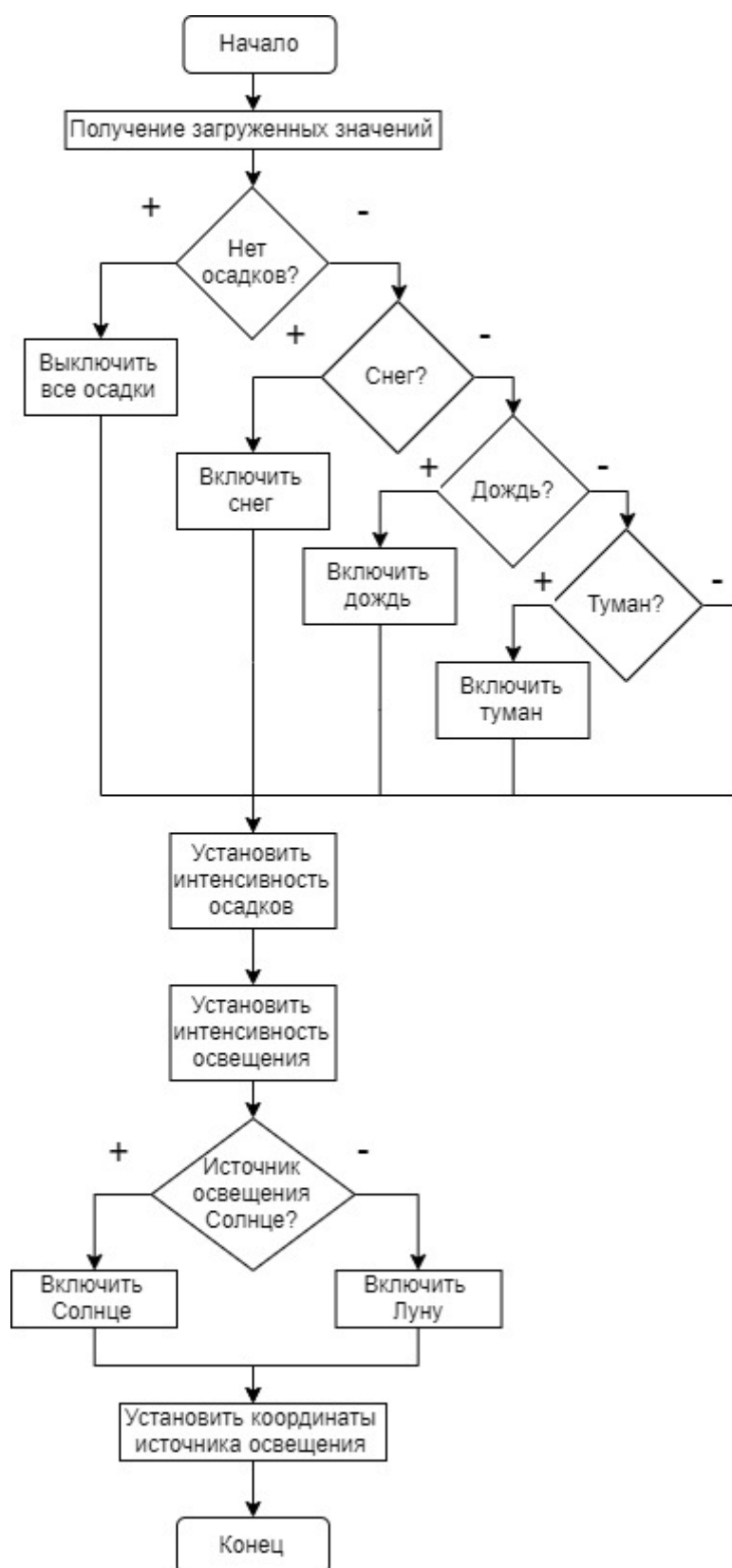


Рисунок 12 — Схема загрузки и установления погодных условий

4 РЕАЛИЗАЦИЯ АЛГОРИТМОВ БЛОКА РЕДАКТОРА ПОГОДНЫХ УСЛОВИЙ

4.1 Реализация алгоритмов класса WaterScript

Класс WaterScript служит для управления водным объектом: установки его температуры, сохранения информации о нем в XML-файл, установки значений свойств (волнения, глубины и толщины льда).

Ниже представлена реализация метода «SetProperties».

```
public void SetProperties(int mark, float depthValue, float iHeight) {  
    wavesMark = mark;  
    switch (wavesMark) {  
        case 0:  
            wavesAmplitude = 0;  
            break;  
        case 1:  
            wavesAmplitude = Random.Range(0f, 0.1f);  
            break;  
        case 2:  
            wavesAmplitude = Random.Range(0.1f, 0.5f);  
            break;  
    }  
  
    depth = depthValue;  
    iceHeight = iHeight;  
}
```

На вход методу подаются три параметра:

- mark: int — целочисленное значение, соответствующие выбранному пользователем значению волнения в баллах.
- depthValue: float — значение глубины водного объекта, введенное пользователем.
- iHeight: float — значение толщины льда водного объекта, введенное пользователем.

В зависимости от полученного значения mark, значению амплитуды волнения присваивается соответствующее число. В конце оставшиеся

полученные параметры запоминаются в системе для последующего использования.

Ниже представлена реализация метода SetIceHeight.

```
public void SetIceHeight(float value) {
    if (value > 0) {
        if (ice == null) {
            ice = Instantiate(icePrefab, transform.position + new Vector3(0, 1f, 0),
Quaternion.identity, transform) as GameObject;
        }
    }
    else {
        if (ice != null) {
            Destroy(ice);
        }
    }
    iceHeight = value;
}
```

Данный метод получает при вызове параметр value типа float, обозначающий значение толщины льда, введенное пользователем. Если это значение больше нуля и льда еще не существует на данном водном объекте, то методом Instantiate, которому передаются ссылка на шаблон льда, его позиция, поворот и данные о родителе, к которому прикрепляется данный объект льда, создается объект льда. Если переданное значение равно нулю и объект льда существует на водном объекте, то он уничтожается. В конце в переменную iceHeight заносится переданное значение.

4.2 Реализация алгоритмов класса SnowTerrainGenerator

Класс SnowTerrainGenerator служит для управления снежной поверхностью, а именно: создание ландшафта снега, нанесение снега, сохранения карты мест нанесения снега, как карты высот и построения ландшафта снега по этой карте.

Для создания возможности нанесения снега на поверхность была реализована идея создания отдельного ландшафта, копирующего собой основной ландшафт, но с одной текстурой снега, и располагающийся на 0.001м ниже основного(чтобы не перекрывать его), который пользователь

будет редактировать. В соответствии с полученными с введенными пользователем параметрами, на поверхности, соответствующей размеру кисти, толщина ландшафта снега увеличивается или уменьшается на соответствующую величину. Если пользователь вводит значение толщины снега, равное 0, то ландшафт в данной местности приобретает изначальную высоту, т.е. скрывается под основным ландшафтом.

Метод `CreateSnowTerrain` создает ландшафт, который реализует снежную поверхность. Получает на вход ссылку на основной ландшафт, создает сначала простой ландшафт, который на 0.001м ниже основного:

```
snow_terrain.transform.position -= new Vector3(t.transform.position.x,
0.001f, t.transform.position.z);
```

создает копию его карты высот:

```
heights = terComp.terrainData.GetHeights(0, 0,
terComp.terrainData.heightmapWidth, terComp.terrainData.heightmapHeight);
```

```
float[,] snow_heights = snowTerComp.terrainData.GetHeights(0, 0,
snowTerComp.terrainData.heightmapWidth,
snowTerComp.terrainData.heightmapHeight);
```

```
for (int x = 0; x < terComp.terrainData.heightmapWidth; x++) {
    for (int y = 0; y < terComp.terrainData.heightmapHeight; y++) {
        snow_heights[y, x] = 0;
    }
}
```

и присваивает ее ландшафту снега:

```
snowTerComp.terrainData.SetHeights(0, 0, snow_heights);
```

`SetSnow` является сопрограммой, которая реализует нанесение снега на поверхность. Принцип работы состоит в изменении карты высот ландшафта снега. Для начала определяем координаты области изменения высоты снега. Далее во вложенном цикле проходим по массиву, соответствующему карте высот. Если значение `snowHeight` высоты снега, введенное пользователем, больше 0, то пускаем луч:


```
Physics.Raycast(heightPosition, Vector3.down, out downRayHit,
Mathf.Infinity, layerMask);
```

и проверяем на какой поверхности мы находимся:

Если на водном объекте и на нем есть лед:

```
if (ws != null) {
    if (ws.GetIceHeight() > 0) {
        snow_heights[y, x] = (snowHeight + downRayHit.point.y) /
snowTerComp.terrainData.size.y;
    }
}
```

Если на льду:

```
if (pointTransform.GetComponent<IceScript>()) {
    snow_heights[y, x] = (snowHeight + downRayHit.point.y) /
snowTerComp.terrainData.size.y;
}
```

Если на редактируемом ландшафте:

```
if (pointTransform.GetComponent<Terrain>()) {
    snow_heights[y, x] = heights[y, x] + snowHeight /
snowTerComp.terrainData.size.y;
}
```

Иначе, если значение высоты снега равно нулю, то устанавливает карту высот в первоначальное значение и вычитаем 0.1, чтобы ландшафты не перекрывались.

В конце присваиваем карте высот ландшафта снега отредактированную карту высот.

```
snowTerComp.terrainData.SetHeights(0, 0, snow_heights);
```

Полную реализацию алгоритма SetSnow можно посмотреть в Приложении А.

Метод `GenerateSnowHeightMapWitSave` служит для сохранения карты нанесенного снега в изображение, являющееся картой высот снежного ландшафта. Сначала создается текстура с размерами ландшафта:

```
Texture2D mapTexture = new Texture2D (width, height);
```

Затем во вложенном цикле по разнице между стандартными высотами и высотами, полученными с ландшафта определяется цвет пикселя, соответствующий высоте снега в данной точке — чем больше разница между высотами (т.е. фактическая высота снега), тем светлее пиксель.

```
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        float delta = snow_heights[y, x] - heights[y, x];
        if (delta <= 0) {
            delta = 0;
            mapTexture.SetPixel(x, y, new Color(0f, 0f, 0f));
        }
        else {
            mapTexture.SetPixel(x, y, new Color(delta * 100, delta * 100, delta *
100));
        }
    }
}
```

В конце перекодировываем созданную текстуру в изображение формата png и сохраняем в соответствующую папку:

```
byte[] bytes = mapTexture.EncodeToPNG();
string filename = Application.streamingAssetsPath + "/SnowMaps/" +
mapName + ".png";
File.WriteAllBytes(filename, bytes);
```

Полную реализацию алгоритма `GenerateSnowHeightMapWitSave` можно посмотреть в Приложении А.

Метод `CreateTerrainFromHeightmapSave` является обратным методу `GenerateSnowHeightMapWitSave`, и из созданного изображения карты высот снега строит ландшафт снега по тому же принципу — чем светлее пиксель, тем больше толщина снега.

Полную реализацию алгоритма `CreateTerrainFromHeightmapSave` можно посмотреть в Приложении А.

4.3 Реализация алгоритмов класса `weatherMenu`

Данный класс служит для сохранения и загрузки значений всех условий видимости (как погоды, так и освещения). Ниже представлена реализация метода «`LoadProperties`».

Метод получает загруженные значения и сохраняет их в переменную `weatherSaver`. Далее, в зависимости от загруженного типа погодных условий, устанавливается соответствующий.

```
public void LoadProperties() {
    weatherSaver = weatherController.GetComponent<WeatherSaver>();
    switch (weatherSaver.weatherType) {
```

В данном случае ничего не устанавливается («Нет осадков»):

```
    case WeatherType.none: {
        snow.isOn = false; rain.isOn = false;
        fog.isOn = false;   nope.isOn = true;
        break;
    }
```

Устанавливается снег:

```
    case WeatherType.snow: {
        nope.isOn = false; rain.isOn = false;
        fog.isOn = false;  snow.isOn = true;
        break;
    }
```

Устанавливается дождь:

```
    case WeatherType.rain: {
```

```

    nope.isOn = false; snow.isOn = false;
    fog.isOn = false; rain.isOn = true;
    break;
}

```

Устанавливается туман:

```

case WeatherType.fog: {
    nope.isOn = false; snow.isOn = false;
    rain.isOn = false; fog.isOn = true;
    break;
}
}

```

Далее устанавливаются значения интенсивностей осадков и освещения:

```

downfallSlider.value = weatherSaver.downfallIntensity;
lightSlider.value = weatherSaver.lightIntensity;

```

После этого, в зависимости от загруженного источника освещения, устанавливается соответствующий:

```

switch (weatherSaver.sourceType) {

```

В данном случае, устанавливается Солнце:

```

case LightSourceType.sun: {
    moon.isOn = false; sun.isOn = true;
    break;
}

```

В данном случае, устанавливается Луна:

```

case LightSourceType.moon: {
    sun.isOn = false; moon.isOn = true;
    break;
}
}

```

В конце устанавливаются загруженные координаты источника освещения:

```

xField.text = weatherSaver.lightPositionX.ToString();
yField.text = weatherSaver.lightPositionZ.ToString();
}

```

4.4 Примеры функционирования программного обеспечения

При запуске программной системы ИЗП пользователю предоставляется главное меню приложения, изображенное на рисунке 13, с возможностью перехода в основные блоки для дальнейшего использования их функций. Переход в блок редактора сцены осуществляется при помощи клика мыши на кнопке «Редактор местности».



Рисунок 13 – Главное меню программы

После выбора данного блока и его загрузки пользователю для работы с приложением предоставляется интерфейс, основной частью которого является расположенной в верхней части экрана главное меню, дающее доступ к основным функциям по настройке погодных условий. К разрабатываемому блоку из главного меню, представленного на рисунке 14, относятся кнопки «Условия видимости» и «Снег».

При нажатии мышью по кнопке «Снег» пользователю становится доступно меню настройки снега, где он может указать толщину наносимого снега и размер кисти (Рисунок 15). Для реализации нанесения снега пользователю достаточно водить мышью по участкам поверхности, на

которые необходимо нанести снег. Также, у пользователя имеется возможность сразу заполнить всю местность снегом на указанную толщину при нажатии на кнопку «Заполнение местности».

Все изменения принимаются и сохраняются в процессе нанесения снега. При нажатии кнопки «Закрыть» (красный крест в правом верхнем углу панели настройки снега) происходит закрытие панели, отмена установленных изменений и установка значений по умолчанию.

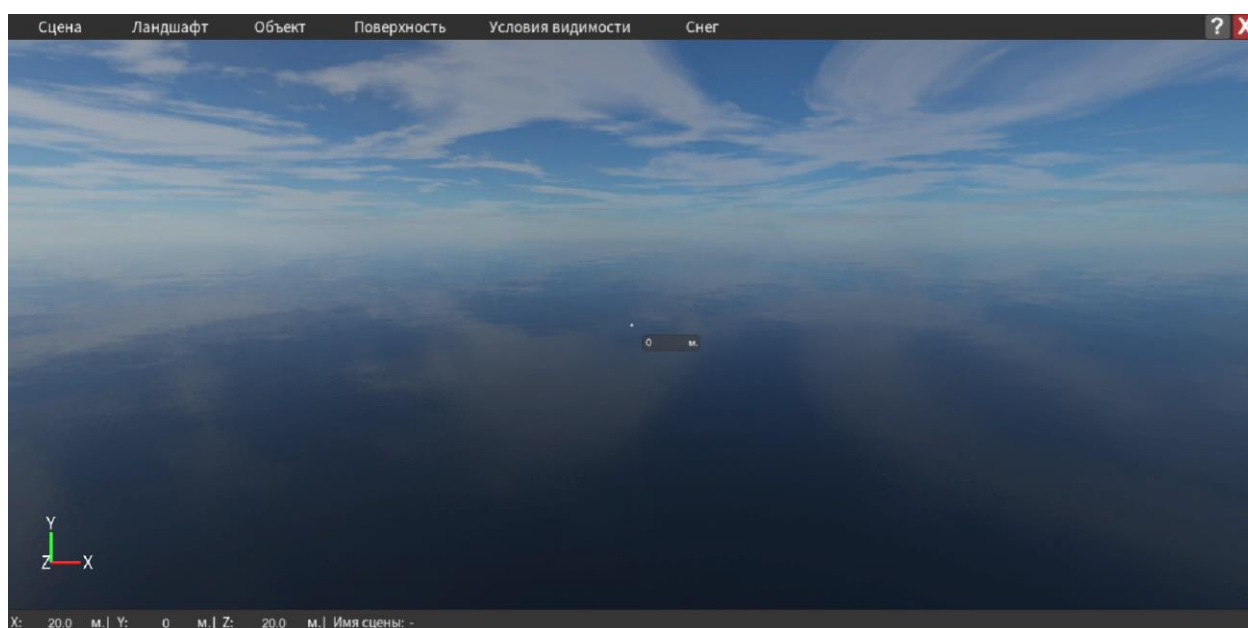


Рисунок 14 – Главное меню редактора сцены

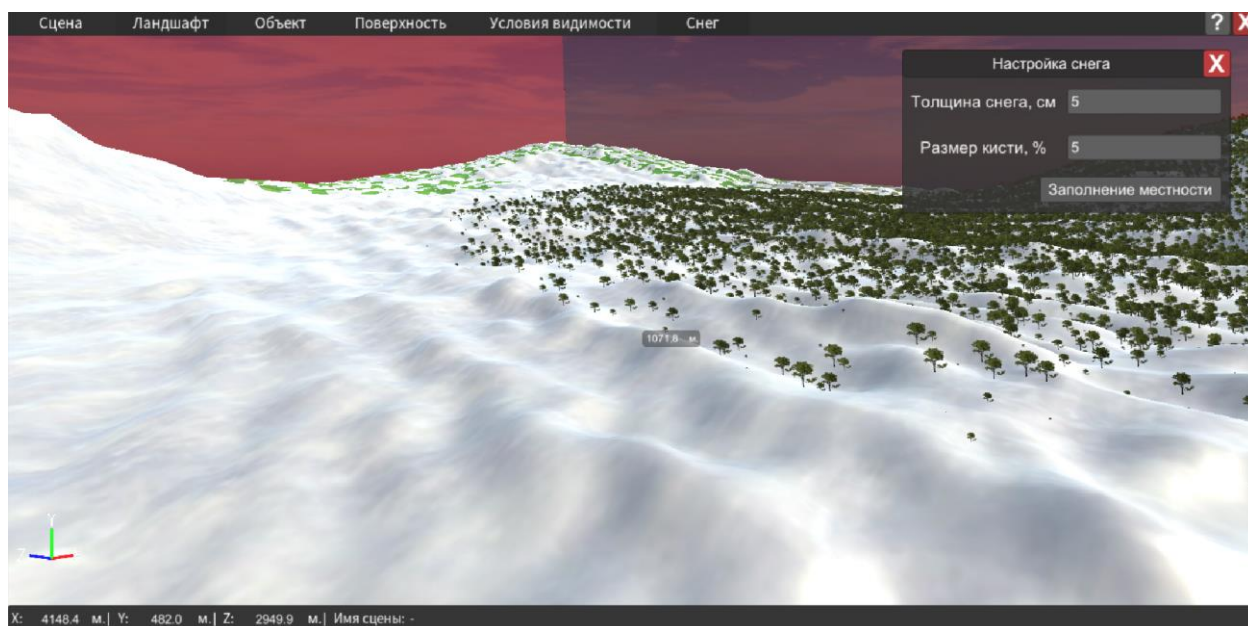


Рисунок 15 — Пример нанесения снега на поверхность (заполнение местности)

При нажатии мышью на пункте меню «Условия видимости» пользователю становится доступна панель настройки условий видимости, где можно:

- выбрать тип погодных условий (один из переключателей «Нет осадков», «Туман», «Снег» и «Дождь») (Рисунки 16 – 18);
- указать интенсивность осадков (Рисунок 19);
- выбрать тип источника освещения (один из переключателей «Солнце» и «Луна») (Рисунок 20);
- указать интенсивность освещения (Рисунок 21);
- указать координаты источника света.

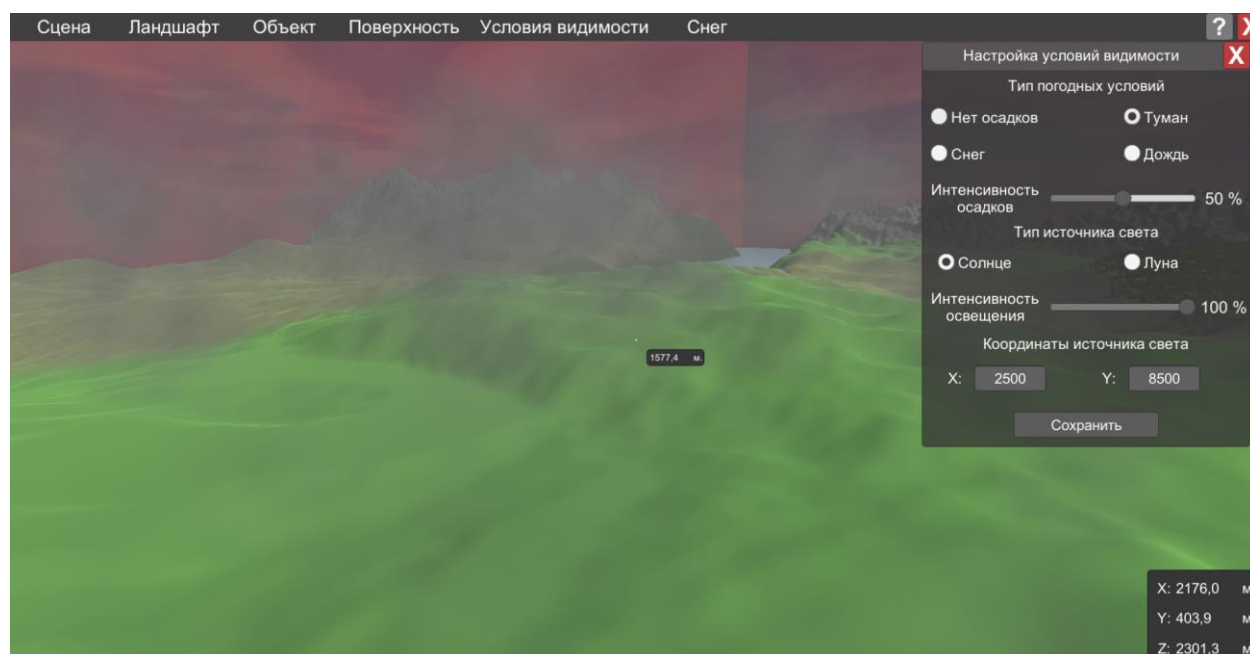


Рисунок 16 — Пример работы тумана

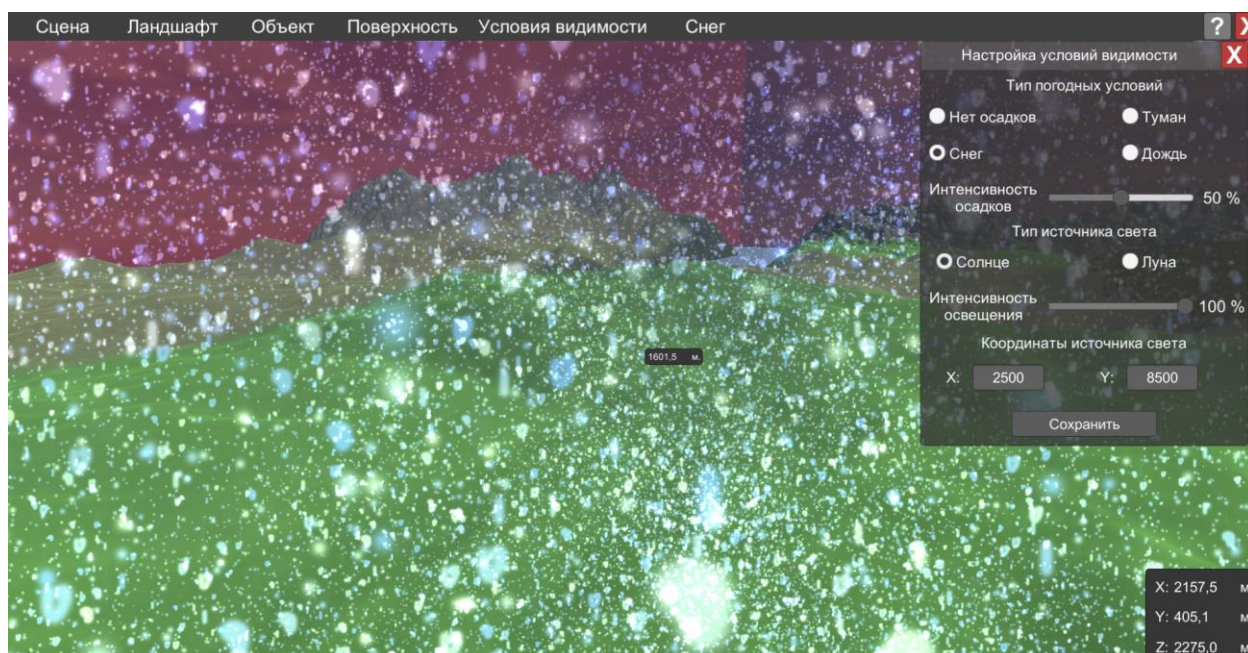


Рисунок 17 — Пример работы снега

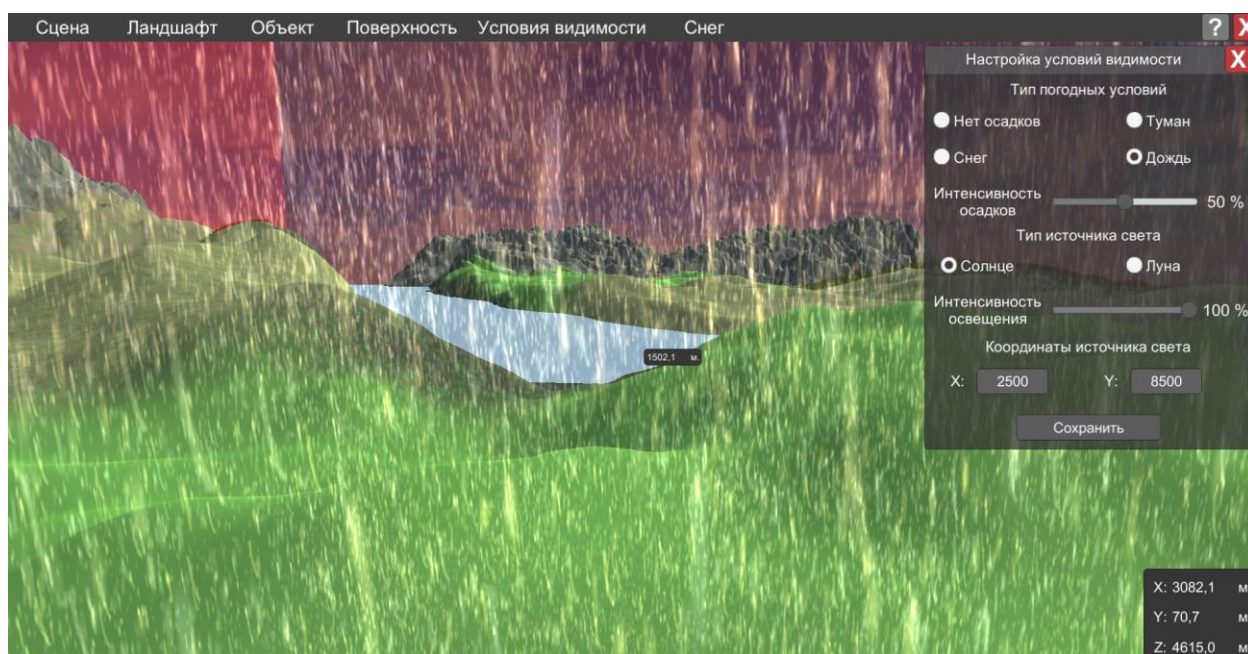


Рисунок 18 — Пример работы дождя



Рисунок 19 — Пример изменения интенсивности погодных условий
(на примере снега)

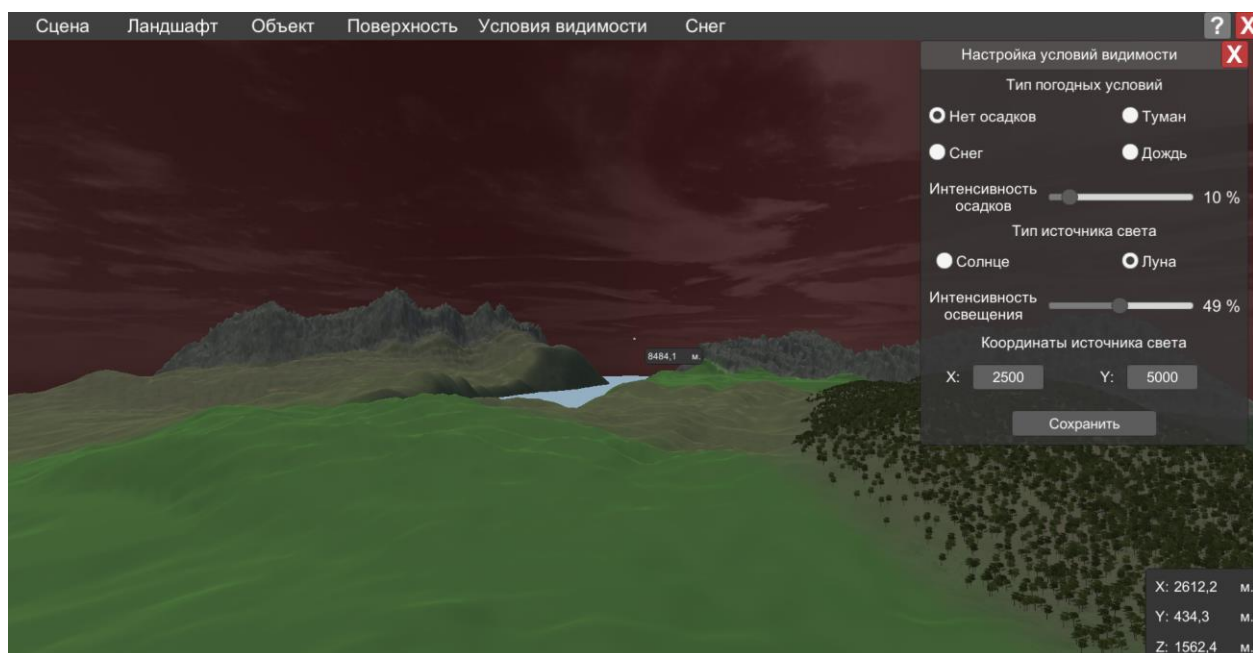


Рисунок 20 — Пример работы Луны

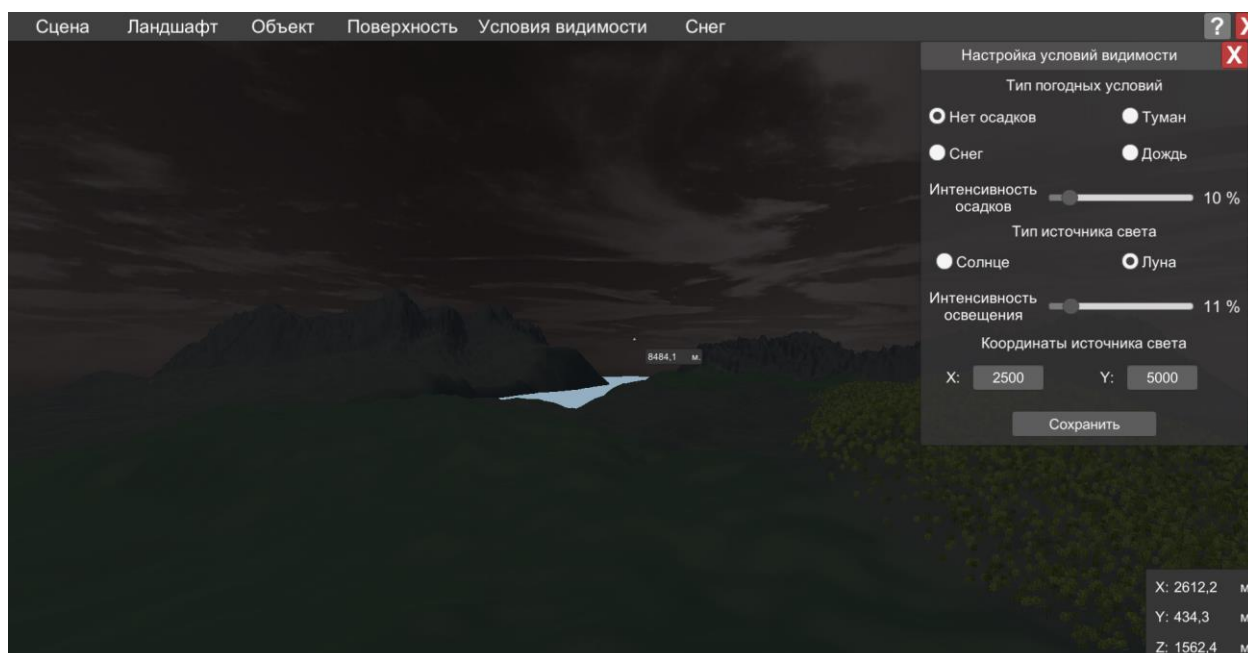


Рисунок 21 — Пример изменения интенсивности освещения

При нажатии кнопки «Сохранить» все изменения принимаются и сохраняются. При нажатии кнопки «Закрыть» (красный крест в правом верхнем углу панели настройки условий видимости) происходит закрытие панели, отмена установленных изменений и установка значений по умолчанию.

При клике правой кнопкой мыши по водному объекту пользователю становится доступна панель настройки водного объекта (Рисунок 22). С помощью нее пользователь может:

- изменить значение толщины льда (Рисунок 23);
- изменить значение волнения водного объекта;
- изменить значение глубины водного объекта.

При нажатии кнопки «Сохранить» все изменения принимаются и сохраняются. При нажатии кнопки «Закрыть» (красный крест в правом верхнем углу панели настройки водного объекта) происходит закрытие панели, отмена установленных изменений и установка значений по умолчанию.

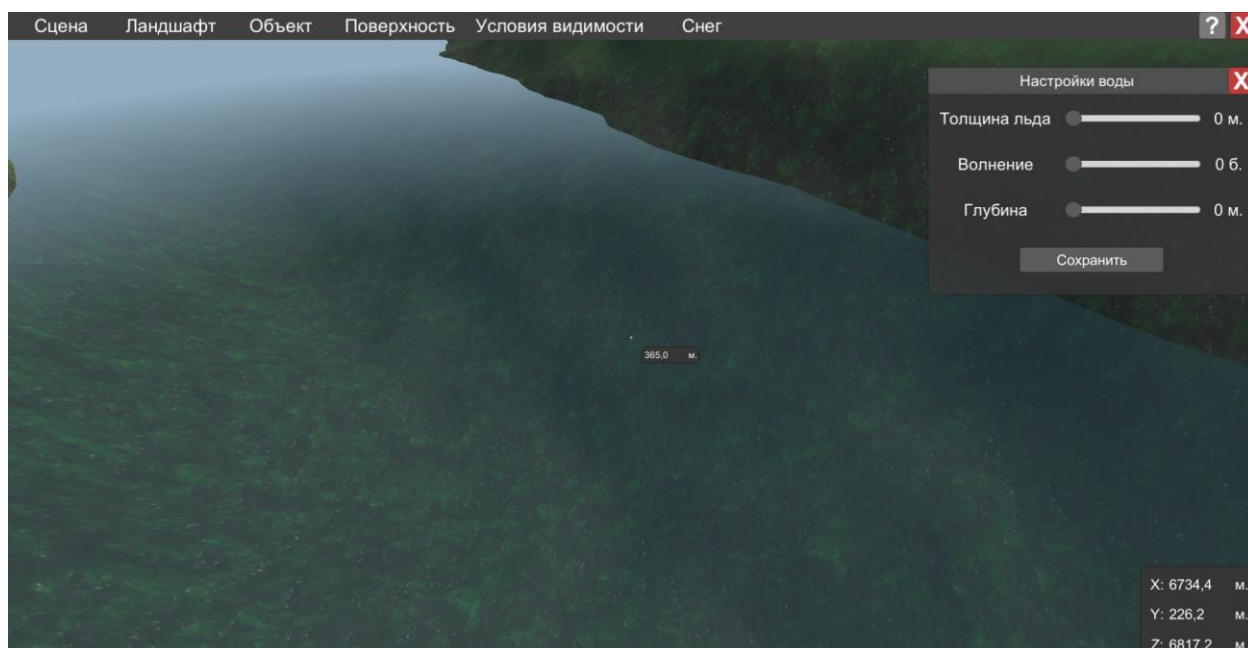


Рисунок 22 — Панель настройки водного объекта

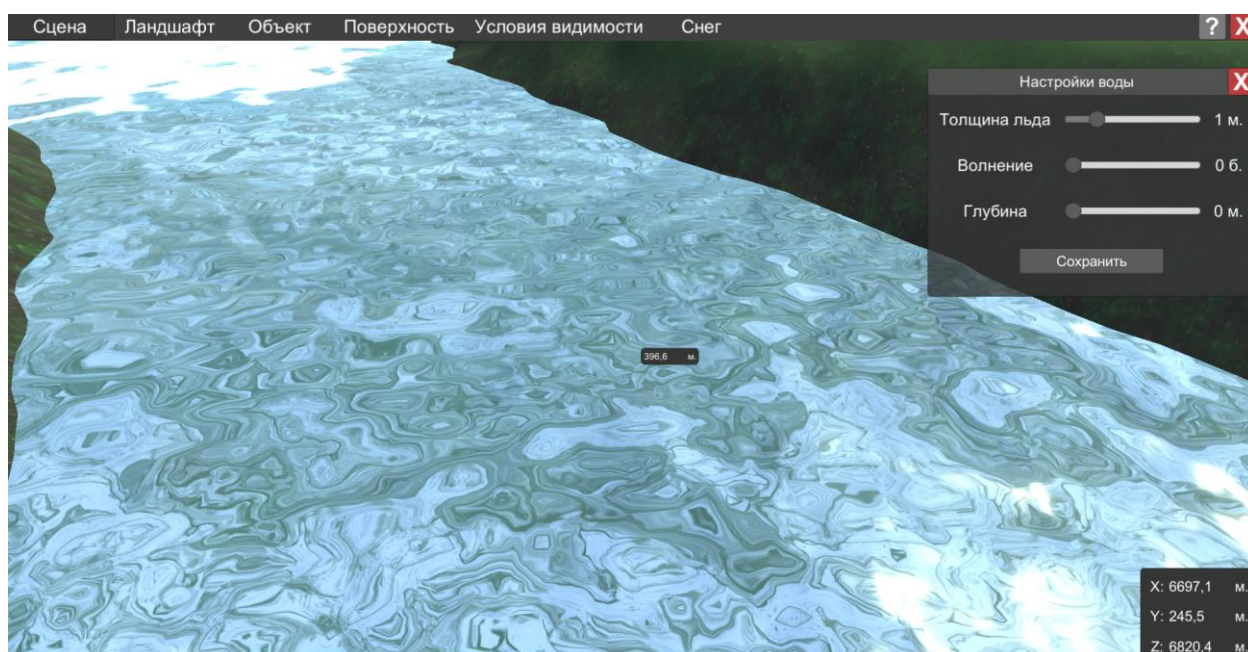


Рисунок 23 — Изменение значения толщины льда

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были определены назначение и область применения, общие требования к программному обеспечению и его входные и выходные данные, была спроектирована и описана с помощью диаграмм языка UML структура реализуемой программной системы, были спроектированы алгоритмы реализуемого блока, а также определены и описаны особенности реализуемого блока программной системы.

Так как все задачи курсовой работы были выполнены, а требуемое программное обеспечение и требуемые алгоритмы были реализованы, курсовую работу можно считать выполненной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Техническое задание на «Программный имитатор закабинного пространства» от 30.08.2018

2. Программный имитатор закабинного пространства для применения в составе технологического стенда комплексной настройки и проверки комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики: Пояснительная записка УРКТ. 04.08.01-01 81 01 ЛУ

3. Программный имитатор закабинного пространства для применения в составе технологического стенда комплексной настройки и проверки комплекса для обеспечения поисково-спасательных операций, проводимых с помощью летательных аппаратов в условиях Арктики: Описание программы УРКТ. 04.08.01-01 13 01 ЛУ

4. Шилдт, Г. С# 4.0: полное руководство [Текст] / Г. Шилдт. — ООО «И.Д. Вильямс», 2011. — 297 с.

5. Дикинсон, К. Оптимизация игр в Unity 5. Советы и методы оптимизации приложений [Текст] / К. Дикинсон. — ДМК-Пресс, 2017. — 306 с.

ПРИЛОЖЕНИЕ А

(обязательное)

ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

«waterMenu.cs»

```
using System.Collections;using System.Collections.Generic;using UnityEngine;using UnityEngine.UI;
public class waterMenu : MonoBehaviour{
    public Slider iceHeightSlider, wavesMarkSlider, depthSlider;
    public GameObject acriveWater;  private WaterScript activeScript;
    public float lastDepth, lastIceHeight;  public int lastWavesMark;
    public void SetActiveWater(GameObject waterObject) {
        if (acriveWater != null) SetLastSettingsToObject();
        acriveWater = waterObject;
        activeScript = acriveWater.GetComponent<WaterScript>();
        lastIceHeight = activeScript.GetIceHeight();
        lastWavesMark = activeScript.GetWavesMark();
        lastDepth = activeScript.GetDepth();
        iceHeightSlider.value = lastIceHeight;
        iceHeightSlider.GetComponent<onSliderChange Value>().ChangeSliderValue();
        wavesMarkSlider.value = lastWavesMark;
        wavesMarkSlider.GetComponent<onSliderChange Value>().ChangeSliderValue();
        depthSlider.value = lastDepth;
        depthSlider.GetComponent<onSliderChange Value>().ChangeSliderValue();
    }
    public void SetIceHeightToObject() {    activeScript.SetIceHeight(iceHeightSlider.value);  }
    public void SetLastSettingsToObject() {
        activeScript.SetIceHeight(lastIceHeight);    activeScript.SetWavesMark(lastWavesMark);
        activeScript.SetDepth(lastDepth);
    }
    public void SaveSettingsToObject() {
        lastIceHeight = iceHeightSlider.value;    lastWavesMark = (int)wavesMarkSlider.value;
        lastDepth = depthSlider.value;
        activeScript.SetIceHeight(lastIceHeight);    activeScript.SetWavesMark(lastWavesMark);
        activeScript.SetDepth(lastDepth);
    }
    public void SliderValueChange() {
        onSliderChange Value[] onSliders = GetComponentsInChildren<onSliderChange Value>();
        for (int i = 0; i < onSliders.Length; i++) {    onSliders[i].ChangeSliderValue();    }
    }
}
```

«WaterScript.cs»

```
using System.Collections;using System.Collections.Generic;using UnityEngine;
using System.Xml.Linq;
public class WaterScript : MonoBehaviour{
    [SerializeField] public float wavesAmplitude, depth, iceHeight;
    [SerializeField] private int wavesMark;
    private Helper helper;
    public GameObject icePrefab;
    public GameObject ice;
    public float temperature;
    public float epr = -35;
    private void Awake() {    helper = FindObjectOfType<Helper>();  }
    private void Start() {    helper.waterObjects.Add(this);  }
    private void OnDestroy() {    Debug.Log("WaterDestroy");    helper.waterObjects.Remove(this);  }
    public void SetTemperature(float temp) {    temperature = temp;  }
    public XElement GetWaterElement() {
        XElement waterElement = new XElement("Water");
        XAttribute posX = new XAttribute("PosX", transform.position.x);
```



```

        XAttribute posY = new XAttribute("PosY", transform.position.y);
        XAttribute posZ = new XAttribute("PosZ", transform.position.z);
        XAttribute scaleX = new XAttribute("ScaleX", transform.localScale.x);
        XAttribute scaleY = new XAttribute("ScaleY", transform.localScale.y);
        XAttribute scaleZ = new XAttribute("ScaleZ", transform.localScale.z);
        waterElement.Add(posX, posY, posZ, scaleX, scaleY, scaleZ);
        return waterElement;
    }

    public XElement GetWaterElementWithParameters() {
        XElement waterElement = new XElement("Water");
        XAttribute posX = new XAttribute("XPos", transform.position.x);
        XAttribute posY = new XAttribute("YPos", transform.position.y);
        XAttribute posZ = new XAttribute("ZPos", transform.position.z);
        Debug.Log("W_POS: " + transform.position.x + " " + transform.position.y + " " + transform.position.z);
        XAttribute ih = new XAttribute("IceHeight", iceHeight);
        XAttribute m = new XAttribute("WavesMark", wavesMark);
        XAttribute d = new XAttribute("Depth", depth);
        waterElement.Add(posX, posY, posZ, ih, m, d);
        return waterElement;
    }

    public void DestroySelf() { Destroy(this.gameObject); }

    public void SetProperties(int mark, float depthValue, float iHeight) {
        wavesMark = mark;
        switch (wavesMark) {
            case 0: wavesAmplitude = 0; break;
            case 1: wavesAmplitude = Random.Range(0f, 0.1f); break;
            case 2: wavesAmplitude = Random.Range(0.1f, 0.5f); break;
        }
        depth = depthValue; iceHeight = iHeight;
    }

    public void SetIceHeight(float value) {
        Debug.Log("SET ice: " + value);
        if (value > 0) {
            if (ice == null) {
                ice = Instantiate(icePrefab, transform.position + new Vector3(0, 1f, 0), Quaternion.identity, transform) as
                GameObject;
            }
        }
        else {
            if (ice != null) { Destroy(ice); }
        }
        iceHeight = value;
    }

    public void SetWavesMark(int value) {
        wavesMark = value;
        switch (wavesMark) {
            case 0: wavesAmplitude = 0; break;
            case 1: wavesAmplitude = Random.Range(0f, 0.1f); break;
            case 2: wavesAmplitude = Random.Range(0.1f, 0.5f); break;
        }
    }

    public void SetDepth(float value) { depth = value; }
    public float GetIceHeight() { return iceHeight; }
    public int GetWavesMark() { return wavesMark; }
    public float GetDepth() { return depth; }
}

```

«IceScript.cs»

```

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

public class IceScript : MonoBehaviour{
    public float temperature; public float epr = -13;
    public void SetTemperature(float temp) { temperature = temp; }
}

```

«SnowMenuScript.cs»

```
using System.Collections;using System.Collections.Generic;using UnityEngine;
public class SnowMenuScript : MonoBehaviour{
    public GameObject editor;
    private void OnEnable() {    editor.GetComponent<Editor>().SetEditorMode(EditorMode.snow_setting);    }
    private void OnDisable() {    editor.GetComponent<Editor>().SetEditorMode(EditorMode.editing);    }
}
```

«SnowTerrainGenerator.cs»

```
using System.Collections;using System.Collections.Generic;using UnityEngine;using UnityEngine.UI;
using System.IO;
public class SnowTerrainGenerator : MonoBehaviour {
    public InputField height;    public InputField size;
    public float snowHeight = 1f;    public int snowSize = 5;
    private float heightAlpha;    public float snowHeightAlpha;
    public float mapAlphaX;    public float mapAlphaY;
    public GameObject snow_terrain;    private Terrain snowTerComp;
    private Terrain terComp;
    public Texture2D snow_texture;    public Material snowMaterial;
    private SplatPrototype[] snow_splat = new SplatPrototype[1];
    private float[,] heights;    public GameObject editingProjector;    public bool randomHeights;
    public void SetHeight() {    snowHeight = float.Parse(height.text) / 100;    }

    public void SetSize() {
        snowSize = int.Parse(size.text); editingProjector.GetComponent<Projector>().orthographicSize = snowSize;
    }
    public void CreateSnowTerrain(GameObject t) {
        TerrainData tData = new TerrainData();    terComp = t.GetComponent<Terrain>();
        tData.heightmapResolution = 513;
        snow_terrain = Terrain.CreateTerrainGameObject(tData);
        snow_terrain.transform.position -= new Vector3(t.transform.position.x, 0.001f, t.transform.position.z);
        snow_splat[0] = new SplatPrototype();    snow_splat[0].texture = snow_texture;
        snowTerComp = snow_terrain.GetComponent<Terrain>();
        snowTerComp.terrainData.size = t.GetComponent<Terrain>().terrainData.size;
        snowTerComp.terrainData.splatPrototypes = snow_splat;
        snowTerComp.materialType = Terrain.MaterialType.Custom;
        snowTerComp.materialTemplate = snowMaterial;
        snow_terrain.name = "SnowTerrain";
        snow_terrain.layer = 12;
        snow_terrain.tag = "SnowTag";
        heightAlpha = 1 / terComp.terrainData.size.y;
        snowHeightAlpha = 1 / snowTerComp.terrainData.size.y;

        mapAlphaX = snowTerComp.terrainData.heightmapResolution / snowTerComp.terrainData.size.x;
        mapAlphaY = snowTerComp.terrainData.heightmapResolution / snowTerComp.terrainData.size.z;
        heights = terComp.terrainData.GetHeights(0, 0, terComp.terrainData.heightmapWidth,
        terComp.terrainData.heightmapHeight);
        float[,] snow_heights = snowTerComp.terrainData.GetHeights(0, 0,
        snowTerComp.terrainData.heightmapWidth, snowTerComp.terrainData.heightmapHeight);
        for (int x = 0; x < terComp.terrainData.heightmapWidth; x++) {
            for (int y = 0; y < terComp.terrainData.heightmapHeight; y++) {    snow_heights[y, x] = 0;    }
        }
        snowTerComp.terrainData.SetHeights(0, 0, snow_heights);
    }
    public IEnumerator SetSnow(Vector3 position, bool isIce) {
        Vector3Int st_position = new Vector3Int((int)(position.x * mapAlphaX), (int)position.y, (int)(position.z *
        mapAlphaY));
        int x_pos = st_position.x;    int y_pos = st_position.z;
        int x_null = x_pos - snowSize / 2;    int y_null = y_pos - snowSize / 2;
        int x_fin = x_pos + snowSize / 2;    int y_fin = y_pos + snowSize / 2;
        if (x_null < 0) x_null = 0;    if (y_null < 0) y_null = 0;
```



```

if (x_fin > snowTerComp.terrainData.heightmapWidth) x_fin = snowTerComp.terrainData.heightmapWidth;
if (y_fin > snowTerComp.terrainData.heightmapHeight) y_fin = snowTerComp.terrainData.heightmapHeight;
Vector3 heightPosition;
int layerMask = (1 << 12) | (1 << 14);
layerMask = ~layerMask;
float[,] snow_heights = snowTerComp.terrainData.GetHeights(0, 0,
snowTerComp.terrainData.heightmapWidth, snowTerComp.terrainData.heightmapHeight);
for (int x = x_null; x <= x_fin; x++) {
    for (int y = y_null; y <= y_fin; y++) {
        if (RoundSize(x_pos, y_pos, x, y) <= Mathf.Pow(snowSize, 2)) {
            heightPosition = new Vector3(x / mapAlphaX, 9000, y / mapAlphaY);
            RaycastHit downRayHit;
            Physics.Raycast(heightPosition, Vector3.down, out downRayHit, Mathf.Infinity, layerMask);
            Transform pointTransform = downRayHit.transform;
            WaterScript ws = pointTransform.GetComponent<WaterScript>();
            if (snowHeight > 0) {
                if (ws != null) {
                    if (ws.GetIceHeight() > 0) {
                        snow_heights[y, x] = (snowHeight + downRayHit.point.y) / snowTerComp.terrainData.size.y;
                    }
                }
                if (pointTransform.GetComponent<IceScript>()) {
                    snow_heights[y, x] = (snowHeight + downRayHit.point.y) / snowTerComp.terrainData.size.y;
                }
                if (pointTransform.GetComponent<Terrain>()) {
                    snow_heights[y, x] = heights[y, x] + (snowHeight + 0.1f) / snowTerComp.terrainData.size.y;
                }
            }
            if (snowHeight == 0) {
                if (ws != null) {
                    if (ws.GetIceHeight() <= 0) { snow_heights[y, x] = heights[y, x] - 0.1f; }
                    snow_heights[y, x] = heights[y, x] - 0.1f;
                }
            }
        }
    }
}
snowTerComp.terrainData.SetHeights(0, 0, snow_heights);
yield return null;
}

public int RoundSize(int x_null, int y_null, int x, int y) {
    float res = Mathf.Pow(((float)x - (float)x_null), 2) + Mathf.Pow(((float)y - (float)y_null), 2); return (int)res; }
public void GenerateSnowHeightMapWithSave(string mapName) {
    int width = snowTerComp.terrainData.heightmapWidth;
    int height = snowTerComp.terrainData.heightmapHeight;
    Texture2D mapTexture = new Texture2D(width, height);
    float[,] snow_heights = snowTerComp.terrainData.GetHeights(0, 0,
snowTerComp.terrainData.heightmapWidth, snowTerComp.terrainData.heightmapHeight);
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            float delta = snow_heights[y, x] - heights[y, x];
            if (delta <= 0) { delta = 0; mapTexture.SetPixel(x, y, new Color(0f, 0f, 0f)); }
            else { mapTexture.SetPixel(x, y, new Color(delta * 100, delta * 100, delta * 100)); }
        }
    }
    mapTexture.Apply();
    byte[] bytes = mapTexture.EncodeToPNG();
    string filename = Application.streamingAssetsPath + "/SnowMaps/" + mapName + ".png";
    File.WriteAllBytes(filename, bytes);
}

public void CreateTerrainFromHeightmapSave(string mapName) {
    GameObject st = GameObject.Find("SnowTerrain");
    if (st == null) { return; }
    Terrain stComp = st.GetComponent<Terrain>();

```

```

int resolution = stComp.terrainData.heightmapResolution;
byte[] b; b = File.ReadAllBytes(Application.streamingAssetsPath + "/SnowMaps/" + mapName + ".png");
Texture2D tex = new Texture2D(resolution, resolution);
tex.LoadImage(b);
float[,] snow_heights = new float[resolution, resolution];
for (int x = 0; x < terComp.terrainData.heightmapWidth; x++) {
    for (int y = 0; y < terComp.terrainData.heightmapHeight; y++) {
        if (tex.GetPixel(x, y).grayscale == 0) { snow_heights[y, x] = 0; }
        else { snow_heights[y, x] = heights[y, x] + tex.GetPixel(x, y).grayscale / 100; }
    }
}
stComp.terrainData.SetHeights(0, 0, snow_heights);
}
public void FillScerneWithSnow() {
    int lastSize = snowSize; snowSize = 500;
    StartCoroutine(SetSnow(new Vector3(5000, 100, 5000), false)); snowSize = lastSize;
}
}

```

«SnowVortexScript.cs»

```

using System.Collections;using System.Collections.Generic;using UnityEngine;
public class SnowVortexScript : MonoBehaviour{
    public GameObject snowVortex; private GameObject curSnowVortex;
    bool isSnowVortexExist;
    void Update() { EnableSnowVortex(); }
    public void EnableSnowVortex() {
        RaycastHit hit;
        if (Physics.Raycast(transform.position, Vector3.down, out hit)) {
            if (hit.transform.gameObject.tag == "SnowTag" && hit.distance <= 30) {
                if (isSnowVortexExist) { Destroy(curSnowVortex); }
                curSnowVortex = Instantiate(snowVortex, hit.point, Quaternion.Euler(new Vector3(90, 0, 0)));
                isSnowVortexExist = true;
            }
            else { isSnowVortexExist = false; Destroy(curSnowVortex); }
        }
    }
}

```

«weatherMenu.cs»

```

using System.Collections;using System.Collections.Generic;using UnityEngine;using UnityEngine.UI;
public class weatherMenu : MonoBehaviour{
    public GameObject weatherController;
    WeatherSaver weatherSaver;
    public ToggleGroup weatherGroup;
    public Toggle nope, snow, rain, fog;
    public Slider downfallSlider;
    public Toggle sun, moon;
    public Slider lightSlider;
    public InputField xField, yField;
    private WeatherType lastWeatherType;
    private LightSourceType lastLightType;
    private void Start() {
        weatherSaver = weatherController.GetComponent<WeatherSaver>();
        lastWeatherType = WeatherType.none; lastLightType = LightSourceType.sun;
    }

    public void SaveProperties() {
        WeatherType active = WeatherType.none;
        if (nope.isOn) active = WeatherType.none; if (snow.isOn) active = WeatherType.snow;
        if (rain.isOn) active = WeatherType.rain; if (fog.isOn) active = WeatherType.fog;
        float intensity = downfallSlider.value;
        weatherSaver.SetWeatherConditions(active, intensity);
    }
}

```

```

        LightSourceType lightType = LightSourceType.sun;
        if (sun.isOn) lightType = LightSourceType.sun;    if (moon.isOn) lightType = LightSourceType.moon;
        intensity = lightSlider.value;    float x = int.Parse(xField.text);    float z = int.Parse(yField.text);
        weatherSaver.SetLightProperties(lightType, intensity, x, z);
    }

    public void LoadProperties() {
        weatherSaver = weatherController.GetComponent<WeatherSaver>();
        switch (weatherSaver.weatherType) {
            case WeatherType.none: {snow.isOn = false; rain.isOn = false; fog.isOn = false; nope.isOn = true; break;
            }
            case WeatherType.snow: {
                nope.isOn = false;    rain.isOn = false;    fog.isOn = false;    snow.isOn = true;    break;
            }
            case WeatherType.rain: {nope.isOn = false; snow.isOn = false;isOn = false; rain.isOn = true; break;
            }
            case WeatherType.fog: { nope.isOn = false; snow.isOn = false; rain.isOn = false; fog.isOn = true; break;
            }
        }
        downfallSlider.value = weatherSaver.downfallIntensity;
        lightSlider.value = weatherSaver.lightIntensity;
        switch (weatherSaver.sourceType) {
            case LightSourceType.sun: { moon.isOn = false;isOn = true; break;
            }
            case LightSourceType.moon: {sun.isOn = false; moon.isOn = true; break;
            }
        }
        xField.text = weatherSaver.lightPositionX.ToString();
        yField.text = weatherSaver.lightPositionZ.ToString();
    }
}

```

«WeatherSaver.cs»

```

using System.Collections;using System.Collections.Generic;using UnityEngine;using System.Xml.Linq;
public enum WeatherType { none = -1, snow = 0, rain = 1, fog = 2}
public enum LightSourceType { sun, moon}
public class WeatherSaver : MonoBehaviour{
    public GameObject weatherController; public GameObject weatherMenuPanel;
    public WeatherType weatherType = WeatherType.none; public float downfallIntensity;
    public LightSourceType sourceType = LightSourceType.sun;
    public float lightIntensity; public float lightPositionX; public float lightPositionZ;
    public void SetWeatherConditions(WeatherType type, float intensity) {
        weatherType = type;    downfallIntensity = intensity;
    }
    public void SetLightProperties(LightSourceType type, float intensity, float x, float z) {
        sourceType = type;    lightIntensity = intensity;    lightPositionX = x;    lightPositionZ = z;
    }
    public void SetSettings() {
        GetComponent<WeatherEditor>().SetActive((int)weatherType);
        GetComponent<WeatherEditor>().SetIntensity(lightIntensity);
        GetComponent<LightIntensity>().SetLightSource((int)sourceType);
        GetComponent<LightIntensity>().SetPositionOfLightSource(lightPositionX, lightPositionZ);
        GetComponent<LightIntensity>().SetIntensity(lightIntensity);
        if (weatherMenuPanel != null) {    weatherMenuPanel.GetComponent<weatherMenu>().LoadProperties(); }
    }
    public XElement GetWeatherAndLightElement() {
        XElement weatherRoot = new XElement("VisibilityConditions");
        XElement lightElement = new XElement("Lighting");
        lightElement.Add(new XAttribute("XLightSourcePos", lightPositionX));
        lightElement.Add(new XAttribute("ZLightSourcePos", lightPositionZ));
    }
}

```

```

lightElement.Add(new XAttribute("LightSourceType", sourceType));
lightElement.Add(new XAttribute("LightIntensity", lightIntensity));
XElement weatherElement = new XElement("WeatherConditions");
weatherElement.Add(new XAttribute("RainfallType", weatherType));
weatherElement.Add(new XAttribute("RainfallIntensity", downfallIntensity));
weatherRoot.Add(lightElement, weatherElement);
return weatherRoot;
}
}

```

«LightIntensity.cs»

```

using System.Collections;using System.Collections.Generic;using UnityEngine;using UnityEngine.UI;
public class LightIntensity : MonoBehaviour {
    public GameObject editLight;    public Slider intensitySlider;
    public InputField posXField;    public InputField posZField;
    public Material skyMaterial;    private Color skyColor;
    public GameObject moon;    public Material materialMoon;    public Material materialSun;
    public float posX, posY, posZ, radius;
    public void SetIntensity() {
        float currentIntensity = 0;
        if (intensitySlider != null) {            currentIntensity = intensitySlider.value;        }
        editLight.GetComponent<Light>().intensity = currentIntensity * 0.01f;
        RenderSettings.ambientIntensity = currentIntensity * 0.01f;
        skyColor = new Color(currentIntensity * 2.55f * (1 / 255), currentIntensity * 2.55f * (1 / 255), currentIntensity
* 2.55f * (1 / 255));
        skyMaterial.SetColor("SkyTintColor", skyColor);
    }
    public void SetIntensity(float intense) {
        var currentIntensity = intense;
        editLight.GetComponent<Light>().intensity = currentIntensity * 0.01f;
        RenderSettings.ambientIntensity = currentIntensity * 0.01f;
        skyColor = new Color(currentIntensity * 2.55f * (1 / 255), currentIntensity * 2.55f * (1 / 255), currentIntensity
* 2.55f * (1 / 255));
        skyMaterial.SetColor("SkyTintColor", skyColor);
        if (intensitySlider != null) {            intensitySlider.value = intense;        }
    }
    public void SetPositionOfLightSource() {
        posX = float.Parse(posXField.text);
        posZ = float.Parse(posZField.text);
        radius = 20000 * Mathf.Sqrt(2);
        posY = Mathf.Sqrt(radius * radius - Mathf.Pow((posX - 5000), 2) - Mathf.Pow((posZ - 5000), 2));
        editLight.transform.position = new Vector3(posX, posY, posZ);
        editLight.transform.LookAt(new Vector3(5000, 0, 5000));
    }
    public void SetPositionOfLightSource(float x, float z) {
        posX = x;    posZ = z;
        radius = 20000 * Mathf.Sqrt(2);
        posY = Mathf.Sqrt(radius * radius - Mathf.Pow((posX - 5000), 2) - Mathf.Pow((posZ - 5000), 2));
        editLight.transform.position = new Vector3(posX, posY, posZ);
        editLight.transform.LookAt(new Vector3(5000, 0, 5000));
    }
    public void SetLightSource(int type) {
        if (type == 1) {
            moon.SetActive(!moon.activeSelf);    RenderSettings.skybox = materialMoon;
            SetIntensity(50);    moon.transform.LookAt(new Vector3(5000, 0, 5000));
        }
        else if (type == 0) {
            SetIntensity(100);
            RenderSettings.skybox = materialSun;
        }
    }
}
}

```

«WeatherEditor.cs»

```
using System.Collections;using System.Collections.Generic;using UnityEngine;using UnityEngine.UI;
public class WeatherEditor : MonoBehaviour {
    public GameObject snow;  public GameObject rain;
    public GameObject fog;
    public Slider intensitySlider;  public ParticleSystem currentPS;
    public float userIntensity;  public float psIntensity;
    GameObject[] weatherObjects;
    void Start () {    weatherObjects = new GameObject[3] { snow, rain, fog };  }
    public void SetActive(int weatherId) {
        for (int i = 0; i < weatherObjects.Length; i++) {
            if (weatherId == i) {
                if (weatherObjects[i] != null) weatherObjects[i].SetActive(true);
                currentPS = weatherObjects[i].GetComponent<ParticleSystem>();
            } else {
                if (weatherObjects[i] != null) weatherObjects[i].SetActive(false);
            }
        }
    }
    public void SetIntensity() {
        if (currentPS == null) return;    var emission = currentPS.GetComponent<ParticleSystem>().emission;
        emission.rateOverTime = Mathf.RoundToInt(intensitySlider.value * 10);
    }
    public void SetIntensity(float intense) {
        if (currentPS == null) return;    var emission = currentPS.GetComponent<ParticleSystem>().emission;
        emission.rateOverTime = Mathf.RoundToInt(intense * 10);
    }
}
```

«WeatherMoveToEditor.cs»

```
using System.Collections;using System.Collections.Generic;using UnityEngine;
public class WeatherMoveToEditor : MonoBehaviour {
    public GameObject editor;
    void Update () { transform.position = editor.transform.position; }
}
```