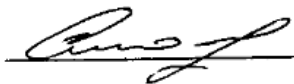


МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии


Работа допущена к защите

 Руководитель
« 08 » 06 20 19 г.

КУРСОВОЙ ПРОЕКТ

по дисциплине «Проектная деятельность»

на тему: «Разработка программной системы имитатора местности: редактор местности»

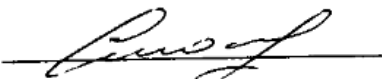
Студент  Марочкин М.А.

Шифр 170584

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71-ПГ

Руководитель  Смоляков М.В.

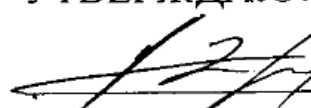
Оценка: « отлично »

Дата 08.06.19

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
«2» марта 20 19 г.

ЗАДАНИЕ
на курсовой проект

по дисциплине «Проектная деятельность»

Студент Марочкин М.А.

Шифр 170584

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71-ПГ

1.Тема курсового проекта:

«Разработка программной системы имитатора местности: редактор местности»

2.Срок сдачи студентом законченной работы: «8» июня 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 курсового проекта

Назначение и область применения ИЗП

Основные требования к программному обеспечению

Входные и выходные данные ИЗП

Проектирование программной системы имитатора местности

Проектирование блока редактора местности

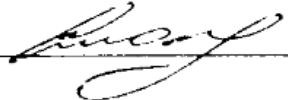
Диаграмма состояний интерфейса

Особенности реализации классов

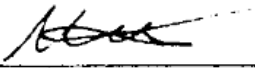
Примеры функционирования программного обеспечения

5. Отчетный материал курсового проекта

Пояснительная записка курсового проекта; презентация

Руководитель  Смоляков М.В.

Задание принял к исполнению: «2» марта 2019

Подпись студента 

Содержание

Введение.....	4
1 Описание программного обеспечения	5
1.1 Назначение и область применения.....	5
1.2 Основные требования к программному обеспечению	6
1.3 Входные и выходные данные.....	9
1.4 Частное задание на разработку	10
2 Проектирование программной системы имитатора местности	12
2.1 Общая структура	12
2.2 Проектирование блока редактора местности	13
2.3 Диаграмма состояний интерфейса	27
3 Реализация блока редактора местности.....	30
3.1 Особенности реализации класса Helper.....	30
3.2 Особенности реализации класса LandscapeLoader	31
3.3 Особенности реализации класса SurfacesManager.....	31
3.4 Особенности реализации класса loadObjectList.....	32
3.5 Примеры функционирования программного обеспечения.....	33
Заключение	38
Список использованных источников	39
Приложение А (обязательное) Исходный текст программы.....	40

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 курсового проекта является реализация модуля редактора местности блока имитатора местности, являющегося составной частью программного обеспечения ИЗП.

Задачами курсового проекта 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) 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 технологического стенда 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

- 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 участок местности из набора готовых ландшафтов, добавлять, удалять и редактировать 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 и 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, %.

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

Курсовой проект, в рамках которого разрабатывается подсистема имитатора местности, является комплексным и содержит ряд отдельных подзадач.

В данном курсовом проекте будет рассмотрено проектирование редактора местности программной системы ИЗП и реализация блока редактора местности, позволяющего пользователю программного обеспечения проводить настройку требуемого для целей моделирования участка ландшафта.

Для выполнения поставленной задачи и реализации ПО 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.

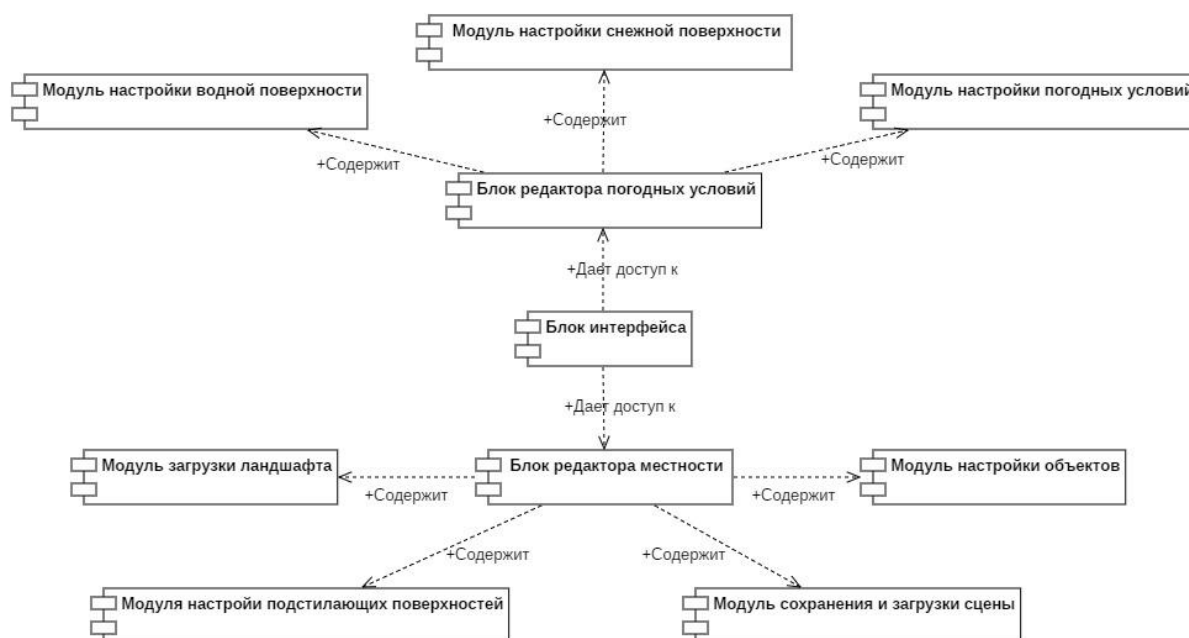


Рисунок 2 – Диаграмма компонентов имитатора местности

2.2 Проектирование блока редактора местности

В редактор местности программной системы имитатора местности с точки зрения функциональных особенностей входящих в его состав скриптов можно выделить две группы классов:

- 1) управляющие классы, реализующие логику работы блока и его функционирование.
- 2) классы интерфейса, реализующие связь между программой и пользователем.

К управляющим классам блока редактора местности можно отнести классы: LandscapeLoader, MapCreator, Surface, SurfacesManager, SaveableObj, Helper, Editor, EditorMovement.

Класс LandscapeLoader предназначен для загрузки выбранной пользователем местности из набора ландшафтов.

Поля класса:

- 1) wallPrefab – объект границы зоны редактирования местности;
- 2) walls – массив объектов, хранящихся в wallPrefab;
- 3) water_prefab – объект водной поверхности;
- 4) terrain – ссылка на объект загруженного ландшафта;
- 5) loadTerrainName – имя файла для загрузки ландшафта.

Методы класса:

- 1) GetLoadLandscapeName() – получение имени файла загруженного ландшафта;
- 2) LoadLandscape() – загрузка ландшафта по имени файла;
- 3) SetBorderWalls() – установка границ зоны редактирования местности.

Класс MapCreator предназначен для создания карты температур и ЕПР по выбранному пользователем ландшафту для дальнейшего моделирования.

Поля класса:

- 1) terrain – ссылка на объект загруженного ландшафта;
- 2) terrainData – данные, хранящие информацию об объекте terrain;
- 3) temperatureTexture – текстура, хранящая карту температур подстилающих поверхностей;
- 4) temperatureMaterial – ссылка на материал, использующий карту температур;
- 5) eprTexture – текстура, хранящая карту ЕПР подстилающих поверхностей;
- 6) eprMaterial – ссылка на материал, использующий карту ЕПР.

Методы класса:

1) CreateMaps() – генерация карты температур и ЕПР для загруженного ландшафта.

Класс Surface предназначен для хранения информации о подстилающей поверхности.

Поля класса:

- 1) type – тип подстилающей поверхности;
- 2) name – наименование подстилающей поверхности;
- 3) epr – значение ЕПР подстилающей поверхности;
- 4) temperature – значение температуры подстилающей поверхности;
- 5) texture – текстура подстилающей поверхности.

Класс SurfacesManager предназначен для настройки пользователем параметров подстилающих поверхностей.

Поля класса:

- 1) areaMenu – ссылка на объект панели меню настройки подстилающих поверхностей для взаимодействия с интерфейсом;
- 2) swamp – информация о подстилающей поверхности типа «Болото»;
- 3) field – информация о подстилающей поверхности типа «Поле»;
- 4) bush – информация о подстилающей поверхности типа «Кустарник»;
- 5) forest – информация о подстилающей поверхности типа «Лес»;
- 6) rock – информация о подстилающей поверхности типа «Скальный грунт»;
- 7) water – информация о подстилающей поверхности типа «Вода»;
- 8) ice – информация о подстилающей поверхности типа «Лед»;
- 9) snow – информация о подстилающей поверхности типа «Снег».

Методы класса:

1) Awake() – стандартный метод, вызываемый перед началом работы класса;

2) SetSurfacesTemperature() – установка значений температур подстилающих поверхностей;

3) GetTemperatures() – получение значений температур подстилающих поверхностей;

4) GetEprs() – получение значений ЕПР подстилающих поверхностей;

5) SetTemperatureForWater() – установка значения температуры для водных поверхностей;

6) SetTemperatureForIce() – установка значения температуры для льда;

7) SetTempsForSurfaces() – установка значений температур подстилающих поверхностей;

8) GetTemperatureElement() – получение данных о значениях температур подстилающих поверхностей для сохранения.

Класс SaveableObj предназначен для хранения информации о редактируемых пользователем объектах и их параметрах.

Поля класса:

1) objectName – имя редактируемого объекта;

2) temperature – значение температуры редактируемого объекта

3) epr – значение ЕПР редактируемого объекта;

4) swimmingObject – значение, определяющие возможность установки объекта на водную поверхность;

5) temperatureTexture – текстура соответствующая температуре объекта;

6) ed – ссылка на объект, с помощью которого производится редактирование местности.

Методы класса:

1) Awake() – стандартный метод, вызываемый перед началом работы класса;

2) Start() – стандартный метод, вызываемый с началом работы класса;

3) ON() – получение имени редактируемого объекта;

4) OnDestroy() – стандартный метод, вызываемый перед уничтожением экземпляра класса;

5) `GetElement()` – получение данных о редактируемом объекте для сохранения;

6) `GetDistance()` – получение расстояние от редактируемого объекта до положения на местности пользователя;

7) `DestroySelf()` – уничтожение редактируемого объекта.

Класс `Helper` предназначен для сохранения и загрузки созданных пользователем прототипов местности.

Поля класса:

1) `terrainManager` – ссылка на объект, хранящий информацию о текущем ландшафте;

2) `weatherController` - ссылка на объект, хранящий информацию о текущих погодных условиях;

3) `editor` – ссылка на объект, с помощью которого производится редактирование местности;

4) `SavePath` – путь до каталога с файлами сохраненных прототипов местности;

5) `objects` – массив объектов класса `SaveableObj`, предназначенный для хранения информации о редактируемых объектах;

6) `waterObjects` – массив объектов водных поверхностей, присутствующих на местности;

7) `ed` – внутренняя ссылка на объект, с помощью которого производится редактирование местности;

8) `terrain_name` – имя объекта загруженного ландшафта;

9) `root` – корневой элемент для сохранения данных в формате XML;

10) `currentSceneName` – текущее имя файла для сохранения прототипа местности;

11) `saveName` – имя файла прототипа местности, используемое при сохранении.

Методы класса:

1) `Start()` – стандартный метод, вызываемый с началом работы класса;

- 2) ClearSceneName() – очистка имени файла для сохранения;
- 3) SetSceneName() – установка имени файла для сохранения;
- 4) TrySave() – сохранение файла прототипа местности с учетом возможных ошибок;
- 5) Save() – сохранение файла прототипа местности;
- 6) Load() – загрузка файла прототипа местности;
- 7) Wait() – ожидание чтения и загрузки файла;
- 8) GenerateScene() – генерация местности по загруженному файлу.

Класс Editor предназначен для редактирования местности пользователем.

Поля класса:

- 1) tData – данные, хранящие информацию об объекте загруженного ландшафта;
- 2) _camera – ссылка на объект – камеру, выводящую изображение редактируемой местности на экран;
- 3) RightCamera – ссылка на дополнительную камеру;
- 4) BottomRightCamera – ссылка на дополнительную камеру;
- 5) cameraComp – стандартный компонент, обеспечивающий работу камер;
- 6) objectListLoader – ссылка на объект интерфейса выбора редактируемых объектов из списка;
- 7) menuClick – ссылка на объект интерфейса, обрабатывающий нажатия мыши по меню;
- 8) settingsMenu – ссылка на объект интерфейса настройки параметров редактируемых объектов;
- 9) waterMenu – ссылка на объект интерфейса настройки параметров водных поверхностей;
- 10) terrainManager – ссылка на объект, хранящий информацию о текущем ландшафте;

11) `editingProjector` – ссылка на объект, подсвечивающий действия пользователя на редактируемой местности;

12) `myName` – имя выбранного из списка редактируемого объекта;

13) `editObject` – ссылка на текущий выбранный для редактирования объект;

14) `target` – значение, определяющие возможность выбора нового объекта для редактирования;

15) `snowSetting` – значение, определяющие возможность установки нового участка снежной поверхности.

Методы класса:

1) `Start()` – стандартный метод, вызываемый с началом работы класса;

2) `Update()` – стандартный метод, постоянно вызываемый в течение всей работы цикла;

3) `SetEditorMode()` – установка текущего режима редактирования;

4) `SetTerrainData()` – установка данных об объекте загруженного ландшафта.

Класс `EditorMovement` предназначен для перемещения пользователя внутри зоны редактируемой местности.

Поля класса:

1) `XPos` – поле для вывода координаты X пользователя в пределах зоны редактируемой местности;

2) `YPos` – поле для вывода координаты Y пользователя в пределах зоны редактируемой местности;

3) `ZPos` – поле для вывода координаты Z пользователя в пределах зоны редактируемой местности;

4) `axes` – стандартный компонент, определяющий возможные оси движения мыши;

5) `speed` – скорость передвижения пользователя;

6) `sensitivityHor` – горизонтальная чувствительность мыши;

7) `sensitivityVert` – вертикальная чувствительность мыши;

8) `minimumVert` – минимальный угол наклона пользовательской камеры по вертикали;

9) `maximumVert` – максимальный угол наклона пользовательской камеры по вертикали;

10) `_rotationX` – угол поворота пользовательской камеры по горизонтали;

11) `moveX` – координата X, отражающая изменение позиции пользователя;

12) `moveY` – координата Y, отражающая изменение позиции пользователя;

13) `moveZ` – координата Z, отражающая изменение позиции пользователя;

14) `ray` – стандартный компонент – луч, рассчитывающий текущую высоту пользователя над объектом ландшафта в зоне редактируемой местности;

15) `hit` – точка попадания луча на объекте ландшафта;

16) `move` – вектор передвижения пользователя;

17) `characterController` – стандартный компонент, позволяющий осуществлять управление.

Методы класса:

1) `Start()` – стандартный метод, вызываемый с началом работы класса;

2) `Update()` – стандартный метод, постоянно вызываемый в течение всей работы цикла.

На основании описания управляющих классов можно составить диаграмму классов, представленную на рисунке 3. Класс `MonoBehaviour` является стандартным и родительским для указанных на диаграмме классов.

К классам интерфейса блока редактора местности можно отнести

Класс `loadTerrainList` предназначен для загрузки выбранной пользователем местности из набора ландшафтов.

1) menuPanel – ссылка на объект панели меню выбора ландшафта;

3) `creationPosition` – позиция отображения превью модели ландшафта;

ландшафта;

5) obj – объект превью модели ландшафта;

6) `terrainManager` – ссылка на объект, хранящий информацию о текущем ландшафте;

7) `activeTerrainName` – имя текущего выбранного для загрузки ландшафта.

Методы класса:

1) `Start()` – стандартный метод, вызываемый с началом работы класса;

2) `ImportObject()` – загрузка объекта превью модели ландшафта;

3) `DestroyObject()` – уничтожение объекта превью модели ландшафта;

4) `SetActiveTerrainName()` – установка имени текущего выбранного ландшафта;

5) `LoadLandscape()` – загрузка выбранного ландшафта.

Класс `areaMenuScript` предназначен для настройки параметров подстилающих поверхностей.

Поля класса:

1) `forestTslider` – стандартный элемент интерфейса – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Лес»;

2) `fieldTslider` – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Поле»;

3) `bushTslider` – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Кустарник»;

4) `swampTslider` – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Болото»;

5) `rockTslider` – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Скальный грунт»;

6) `waterTslider` – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Вода»;

7) `snowTslider` – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Снег»;

8) iceTslider – слайдер, отвечающий за установку значения температуры подстилающей поверхности типа «Лед»;

9) terrainManager – ссылка на объект, хранящий информацию о текущем ландшафте.

Методы класса:

1) SetTemperature() – установка значений температур;

2) LoadTemperature() – получение значений температур.

Класс loadObjectList предназначен для загрузки выбранного пользователем редактируемого объекта из набора объектов.

Поля класса:

1) menuPanel – ссылка на объект панели меню выбора редактируемых объектов;

2) buttonPrefab – объект – элемент списка редактируемых объектов;

3) creationPosition – позиция отображения превью модели редактируемого объекта;

4) controller – объект, управляющий анимацией превью модели редактируемого объекта;

5) obj – объект превью модели редактируемого объекта;

6) previewCamera – объект – камера, отображающая превью модели редактируемого объекта;

7) activeObjectName – имя текущего выбранного для загрузки редактируемого объекта.

Методы класса:

1) Start() – стандартный метод, вызываемый с началом работы класса;

2) ImportObject() – загрузка объекта превью модели редактируемого объекта;

3) DestroyObject() – уничтожение объекта превью модели редактируемого объекта;

4) SetActiveObjectName() – установка имени текущего выбранного редактируемого объекта;

5) `GetActiveObjectName()` – получение имени текущего выбранного редактируемого объекта.

Класс `ObjectSettings` предназначен для хранения информации о параметрах редактируемого объекта.

Поля класса:

- 1) `temperature` – значение температуры редактируемого объекта;
- 2) `position` – вектор позиции редактируемого объекта;
- 3) `rotation` – вектор поворота редактируемого объекта.

Методы класса:

- 1) `SetSettings()` – установка параметров редактируемого объекта.

Класс `settingMenu` предназначен для редактирования пользователем параметров редактируемого объекта.

Поля класса:

- 1) `editObject` – ссылка на текущий редактируемый объект;
- 2) `editTransform` – стандартный компонент, содержащий информацию о положении объекта;
- 3) `nameText` – поле для отображения имени редактируемого объекта;
- 4) `tempSlider` – слайдер для изменения значения температуры редактируемого объекта;
- 5) `lastSettings` – объект класса `ObjectSettings` для хранения настроек редактируемого объекта;
- 6) `xPos` – поле для ввода координаты X позиции объекта;
- 7) `yPos` – поле для ввода координаты Y позиции объекта;
- 8) `zPos` – поле для ввода координаты Z позиции объекта;
- 9) `xRot` – поле для ввода угла поворота объекта по оси X;
- 10) `yRot` – поле для ввода угла поворота объекта по оси Y;
- 11) `zRot` – поле для ввода угла поворота объекта по оси Z;
- 12) `offset` – смещение положения объекта по отношению к ландшафту;
- 13) `nfi` – стандартный компонент, отвечающий за форматирование данных, вводимых пользователем.

Методы класса:

- 1) Start() – стандартный метод, вызываемый с началом работы класса;
- 2) DeleteObject() – уничтожение текущего редактируемого объекта;
- 3) SetEditObject() – установка текущего редактируемого объекта;
- 4) SetLastSettingsToObject() – установка последних сохраненных параметров для текущего редактируемого объекта;
- 5) SaveObjectSettings() – сохранение текущих параметров редактируемого объекта;
- 6) SetXPos() – установка координаты X позиции объекта;
- 7) SetYPos() – установка координаты Y позиции объекта;
- 8) SetZPos() – установка координаты Z позиции объекта;
- 9) SetXRot() – установка угла поворота объекта по оси X;
- 10) SetYRot() – установка угла поворота объекта по оси Y;
- 11) SetZRot() – установка угла поворота объекта по оси Z.

Класс saveScene предназначен для сохранения текущего прототипа редактируемой пользователем местности.

Поля класса:

- 1) menuPanel – ссылка на объект панели меню сохранения местности;
- 2) buttonPrefab – объект – элемент списка сохраненных файлов;
- 3) filepathInput – поле для вывода пути до директории для сохранения файла;
- 4) filenameInput – поле для ввода имени сохраняемого файла;
- 5) saveButton – кнопка, по нажатию на которую происходит сохранение файла.

Методы класса:

- 1) Start() – стандартный метод, вызываемый с началом работы класса;
- 2) RefreshList() – обновление списка сохраненных файлов;
- 3) ClearDirectoryList() – очистка списка сохраненных файлов;
- 4) CheckFilename() – проверка введенного имени файла на допустимость;

5) `OpenFile()` – сохранение местности.

Класс `loadScene` предназначен для загрузки выбранного пользователем прототипа редактируемой местности.

Поля класса:

- 1) `menuPanel` – ссылка на объект панели меню загрузки местности;
- 2) `buttonPrefab` – объект – элемент списка сохраненных файлов;
- 3) `filepathInput` – поле для вывода пути до директории для загрузки файла;
- 4) `filenameInput` – поле для ввода имени загружаемого файла;
- 5) `loadButton` - кнопка, по нажатию на которую происходит загрузка файла.

Методы класса:

- 1) `Start()` – стандартный метод, вызываемый с началом работы класса.
- 2) `RefreshList()` – обновление списка доступных для загрузки файлов;
- 3) `ClearDirectoryList()` – очистка списка доступных для загрузки файлов;
- 4) `OpenFile()` – загрузка местности.

На основании описания классов интерфейса можно составить диаграмму классов, представленную на рисунке 4. Класс `MonoBehaviour` является стандартным и родительским для указанных на диаграмме классов.

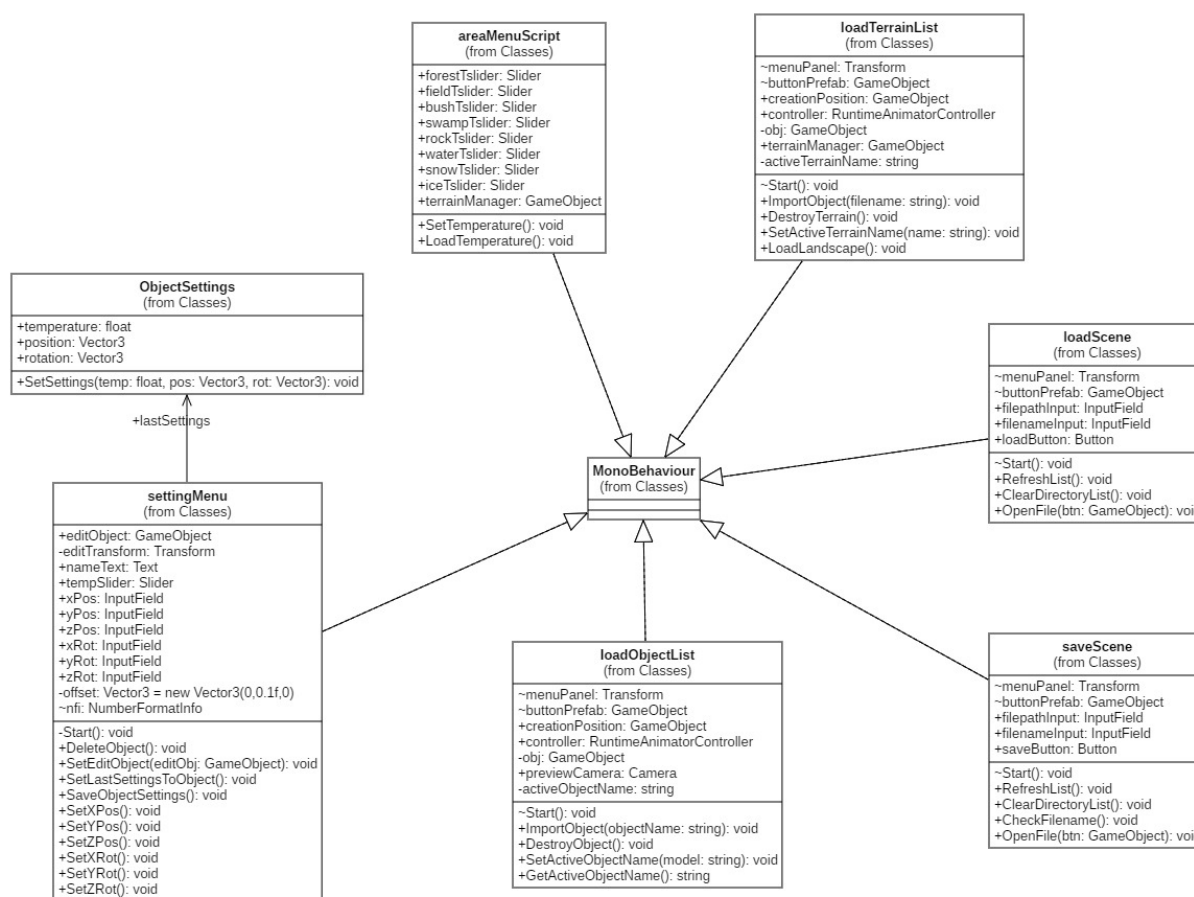


Рисунок 4 – Диаграмма управляющих классов редактора местности

2.3 Диаграмма состояний интерфейса

При запуске редактора местности происходит загрузка и установка интерфейса редактора, а также устанавливается режим редактирования объектов. После этого система перейдет в состояния редактирования местности и пользователь сможет использовать функции программной системы.

При выборе пункта меню «Ландшафт» система переходит в состояние выбора ландшафта для загрузки. При этом будет отображено соответствующие меню выбора и выведен список доступных ландшафтов. Нажатие на кнопку «Выбрать» в данном меню приведет к уничтожению уже имеющегося ландшафта и загрузке выбранного из списка, а нажатие на кнопку «Отмена» позволит закрыть меню без внесения изменений.

При клике мыши на точки местности система перейдет в состояние добавления объекта. По координатам клика мыши будет определена позиция для объекта, после чего произойдет его установка и возврат в состояние редактирования местности.

При выборе пункта меню «Загрузить» будет отображено меню загрузки сцены, в котором будет выведен список ранее сохраненных файлов, доступных для загрузки. Нажатие кнопки «Загрузить» в данном меню приведет к уничтожению существующей местности и загрузке новой из выбранного файла. Нажатие кнопки «Отмена» закроет меню без внесения изменений.

При выборе пункта меню «Сохранить» будет отображено меню сохранения сцены, в котором будет выведен список ранее сохраненных файлов. Пользователь имеет возможность ввести желаемое имя файла в соответствующее поле для ввода. При этом будет произведена проверка корректности введенного имени. Нажатие кнопки «Сохранить» в данном меню приведет к сохранению текущей сцены в файл. Нажатие кнопки «Отмена» закроет меню без внесения изменений.

При клике мыши на любом редактируемом объекте будет отображено меню настройки параметров объекта. При этом в соответствующие поля меню будут выведены текущие параметры объекта. Вносимые пользователем изменения будут применяться к объекту и отображаться. Нажатие кнопки «Сохранить» в данном меню приведет к сохранению текущих параметров объекта. Нажатие кнопки «Отмена» закроет меню и возвратит параметры объекта к последним сохраненным для него значениям.

При нажатии кнопки «Выход» редактор местности завершит свою работу, а система перейдет в главное меню.

Диаграмма состояний системы редактора местности представлена на рисунке 5.

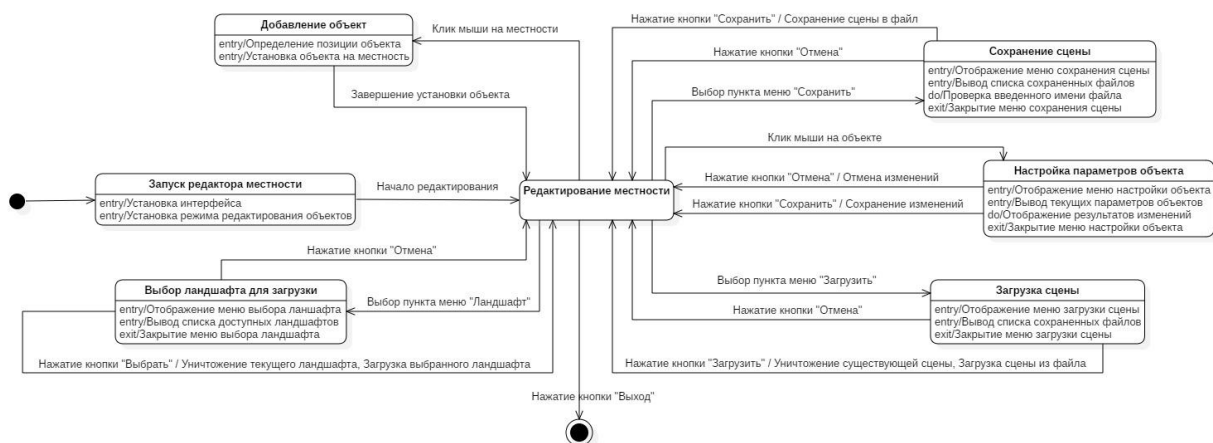


Рисунок 5 – Диаграмма состояний редактора местности

3 Реализация блока редактора местности

3.1 Особенности реализации класса **Helper**

Основное назначение класса **Helper** состоит в сохранении и загрузке файла сцены с редактируемой местностью в формате **xml**. Для хранения информации о присутствующих на местности редактируемых объектах в данном классе присутствует поле «**objects**» типа «**List**», а для хранения информации об объектах водных поверхностей – поле «**waterObjects**» типа «**List**». Также в классе имеется поле «**terrainManager**» типа «**GameObject**» – ссылка на объект, хранящий информацию о текущих параметрах ландшафта.

После того, как пользователь программы примет решение о сохранении местности в файл и в соответствующем меню интерфейса подтвердит действие нажатием кнопки «Сохранить», программой будет вызван метод **TrySave()**, проверяющий была ли эта сцена сохранена ранее под каким либо именем. Если сохранение производилось ранее, то программа перезапишет имеющийся файл, вызвав метод **Save()**. Если же нет, то пользователю будет выведено диалоговое окно, появляющееся также в случае нажатия кнопки «Сохранить как», в котором необходимо ввести свое имя файла для сохранения или выбрать существующий файл для перезаписи. После чего по нажатию кнопки «Сохранить» вызовется метод **Save()**.

Метод **Save()** создает во внутренней памяти структуру данных формата **xml**, в которую в определенной последовательности заносятся необходимые для сохранения сцены параметры. Например, данные об объекте ландшафта, данные о различных параметрах подстилающих поверхностей и др.

Данные о редактируемых и водных объектах на местности класс **Helper** сохраняет, осуществляя прохождение по всем элементам соответствующих списков в цикле и получая от каждого из них необходимые параметры. После получения всех необходимых данных класс сохраняет их в файл формат **xml**, имеющий ранее выбранное пользователем имя.

После принятия пользователем решения о загрузке существующей местности, выбора им существующего файла сцены и нажатия на кнопку «Загрузить» будет вызван метод `Load()`, проверяющий корректность файла и возможность загрузки. Если файл является корректным, вызывается метод `GenerateScene()`, выделяющий из файла данные, записанные в формате `xml`, и осуществляющий на их основе загрузку сцены. При этом восстанавливаются сохраненные параметры местности и на нее добавляются экземпляры редактируемых объектов. Если до загрузки новой сцены пользователь уже производил редактирование, все его изменения будут удалены.

3.2 Особенности реализации класса `LandscapeLoader`

При загрузке сцены, производимой классом `Helper`, необходимо добавить на сцену ранее выбранный пользователем ландшафт. За реализацию этой функции отвечает класс `LandscapeLoader`.

Для загрузки ландшафта вызывается метод `LoadLandscape()`, принимающий в качестве параметра строковую переменную, хранящую имя файла ландшафта. В ходе загрузки уничтожается ранее добавленный пользователем ландшафт, а затем добавляется новый. Ссылка на объект загруженного ландшафта помещается в переменную «`terrain`». Помимо этого, на сцене создается снежная поверхность с нулевой толщиной.

Затем вызывается метод `SetBorderWalls()`, устанавливающий на границах загруженного ландшафта невидимые пользователем «стены», не позволяющие ему выходить за пределы моделируемой области. Объект — экземпляр такой «стены» хранится в поле «`wallPrefab`», сами добавленные на сцену «стены» добавляются в массив «`walls`» класса.

3.3 Особенности реализации класса `SurfacesManager`

Класс `SurfacManager` предназначен для хранения параметров подстилающих поверхностей редактируемой местности. Поля «`forest`», «`field`», «`bush`», «`swamp`», «`rock`», «`water`», «`snow`», «`ice`», являющиеся экземплярами

класса `Surface`, предназначены для хранения информации о соответствующих типах поверхностей.

Метод класса `SetSurfacesTemperature()` вызывается при изменении пользователем значений температур поверхностей. При вызове метода происходит получение измененных значений температур из модуля интерфейса и их запись в поля соответствующих классов.

Метод `GetTemperatures()` возвращает массив значений температур, а метод `GetEprs()` – массив значений ЕПР подстилающих поверхностей.

Метод `SetTemperatureForWater()` устанавливает значение температуры для всех присутствующих на сцене объектов водной поверхности, а `SetTemperatureForIce()` – для всех объектов ледяной поверхности.

Метод `GetTemperatureElement()` вызывается из класса `Helper` и возвращает информацию о параметрах подстилающих поверхностей в виде структуры данных формата `xml` для дальнейшего сохранения.

3.4 Особенности реализации класса `loadObjectList`

При нажатии пользователем на пункт меню «Объект» в приложении будет отображено диалоговое окно со списком объектов и возможностью выбора нужного. За корректную работу данного окна отвечает класс `loadObjectList`.

Метод `Start()`, вызываемый с началом работы класса, создает список редактируемых объектов, загружаемых из ресурсов приложения, и выводит его на экран. Каждый элемент списка представляет собой кнопку, нажатие на которую вызовет метод `ImportObject()`, загружающий невидимую копию объекта на заранее определенную в переменной «`creationPosition`» позицию для отображения превью модели объекта. При этом также вызывается метод `SetActiveObjectName()`, записывающий имя текущего выбранного объекта в переменную «`activeObjectName`».

Если при выборе нового объекта другой, ранее выбранный, уже имелся произойдет уничтожение его загруженной копии через вызов метода DestroyObject().

Аналогичным образом построена работа класса loadTerrainList, обеспечивающего работу меню выбора пользователем ландшафта.

3.5 Примеры функционирования программного обеспечения

При запуске программной системы ИЗП пользователю предоставляется главное меню приложения, изображенное на рисунке 6, с возможностью перехода в основные блоки для дальнейшего использования их функций. Переход в блок имитатора местности осуществляется при помощи клика мыши на кнопке «Редактор местности».

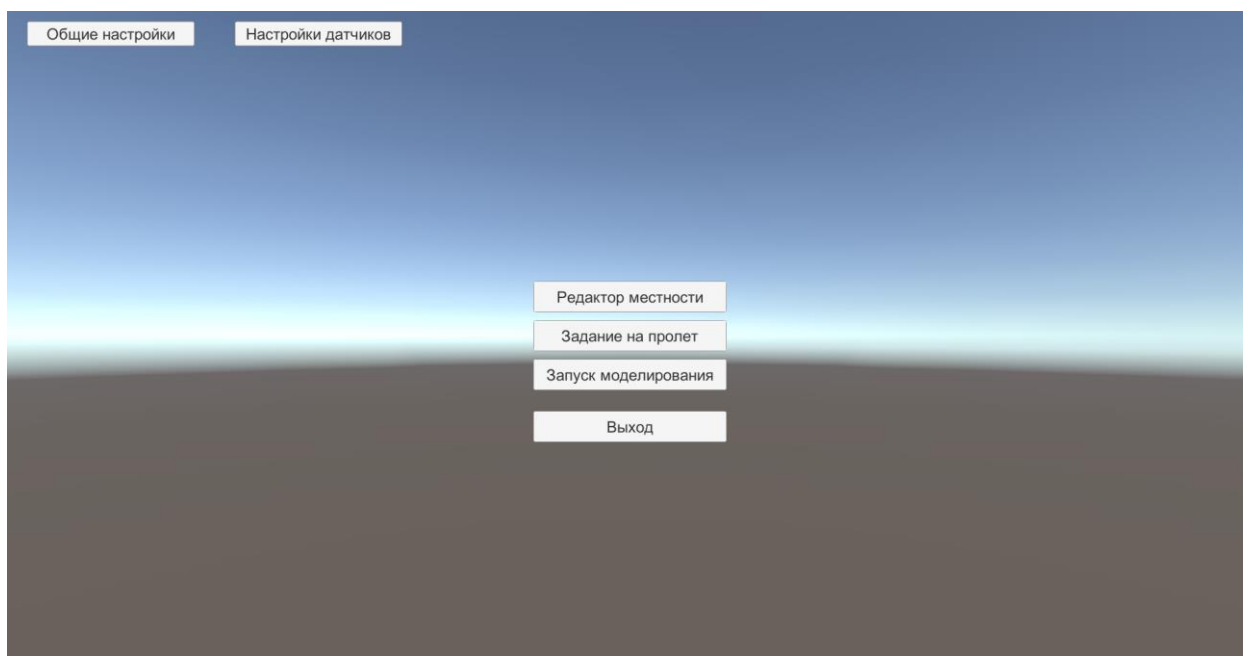


Рисунок 6 – Главное меню программы

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 местности. Главное меню разрабатываемого блока представлено на рисунке 7.

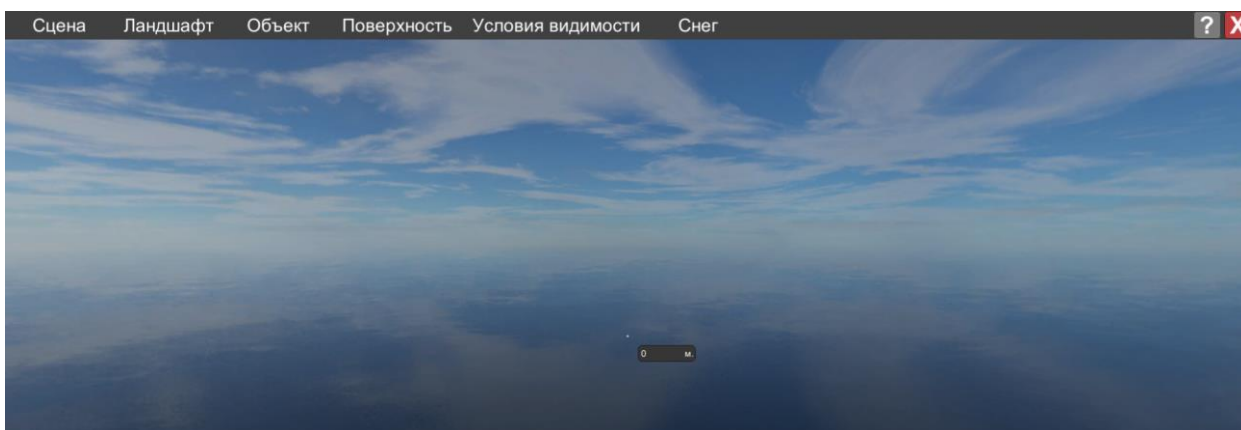


Рисунок 7 – Главное меню имитатора местности

При нажатии мышью на пункте меню «Сцена» пользователю станет доступен список действий для создания новой сцены, сохранения и загрузки существующей из файла. Меню с данным списком представлено на рисунке 8.

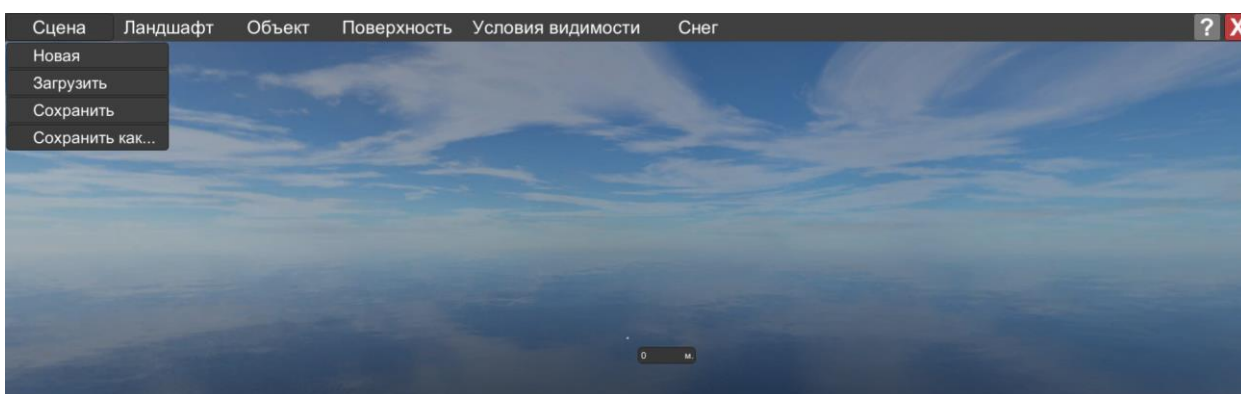


Рисунок 8 – Список действий по работе со сценой

При выборе пункта подменю «Новая» программа отменит все проведенные пользователем изменения и вернет состояний местности к начальному, т.е. имеющемуся при запуске программы.

Выбор пункта подменю «Сохранить» приведет к записи проделанных изменений в файл сохранения местности.

При выборе пункта подменю «Загрузить» пользователю станет доступно диалоговое окно со списком имеющихся сохраненных файлов, выбрав один из которых, возможно загрузить сохраненную в этот файл местность.

Диалоговое окно для загрузки файла сцены представлено на рисунке 9. Окно, предназначенное для сохранения сцены в файл, вызываемое при выборе пункта подменю «Сохранить как...», имеет аналогичный вид.

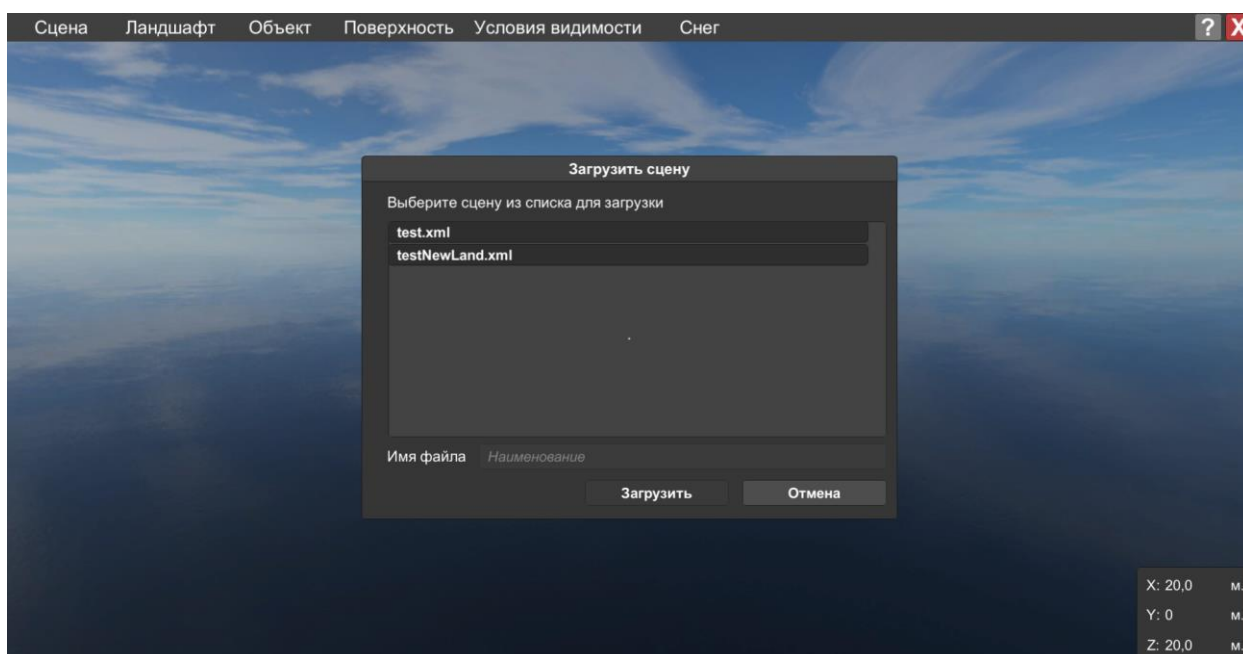


Рисунок 9 – Диалоговое окно для загрузки сцены

При нажатии на пункте главного меню «Ландшафт» пользователю отобразится окно со списком имеющихся ландшафтов, один из которых можно выбрать и загрузить на сцену в качестве текущего ландшафта. Закрывать окно можно при помощи нажатия кнопки «Отмена».

Интерфейс выбора ландшафта представлен на рисунке 10.

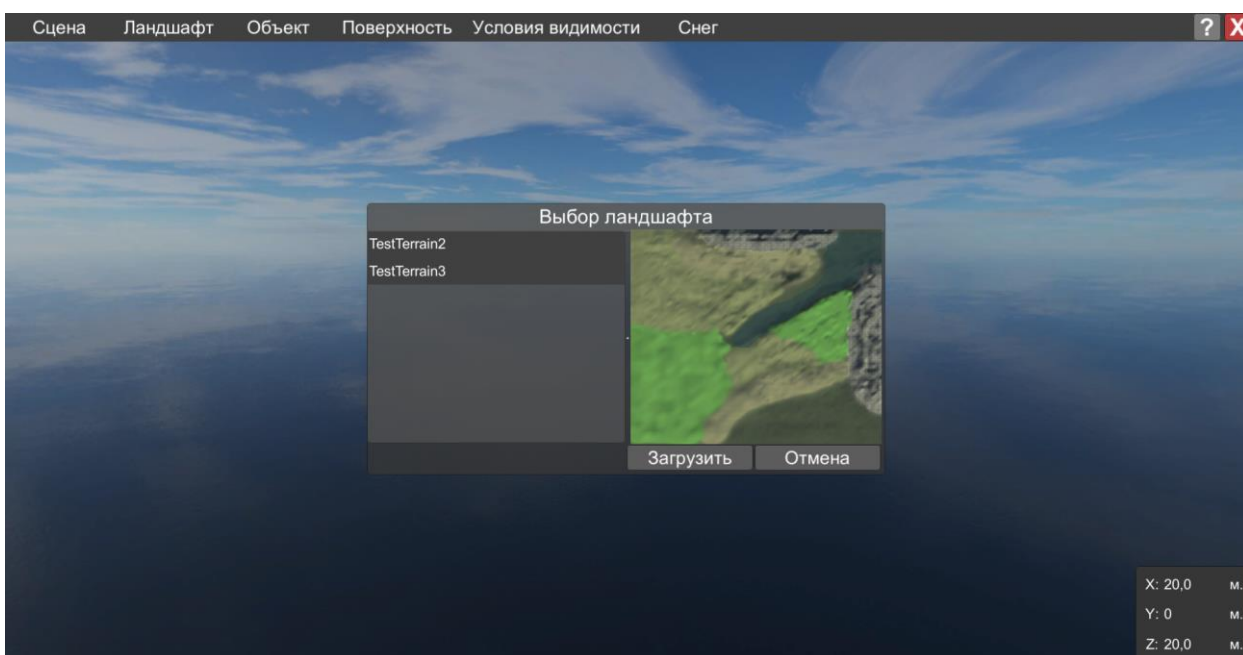


Рисунок 10 – Диалоговое окно для выбора ландшафта

При нажатии на пункте главного меню «Поверхность» будет отображено представленное на рисунке 11 окно для настройки значений

температуры для различных типов подстилающих поверхностей. Изменить значение необходимого параметра можно с помощью изменения положения соответствующего ползунка.

Сохранение изменений производится по нажатию на кнопку «Сохранить».



Рисунок 11 – Диалоговое окно для настройки поверхностей

При нажатии на пункте главного меню «Объект» будет отображено окно со списком доступных редактируемых объектов. При выборе одного из объектов в списке в правой части окна отобразится его превью для наглядного представления.

Для использования выбранного объекта как устанавливаемого на сцену необходимо подтвердить выбор нажатием на кнопку «Использовать». Выход из данного окна осуществляется нажатием на кнопку «Отмена».

Диалоговое окно для выбора редактируемого объекта представлено на рисунке 12.

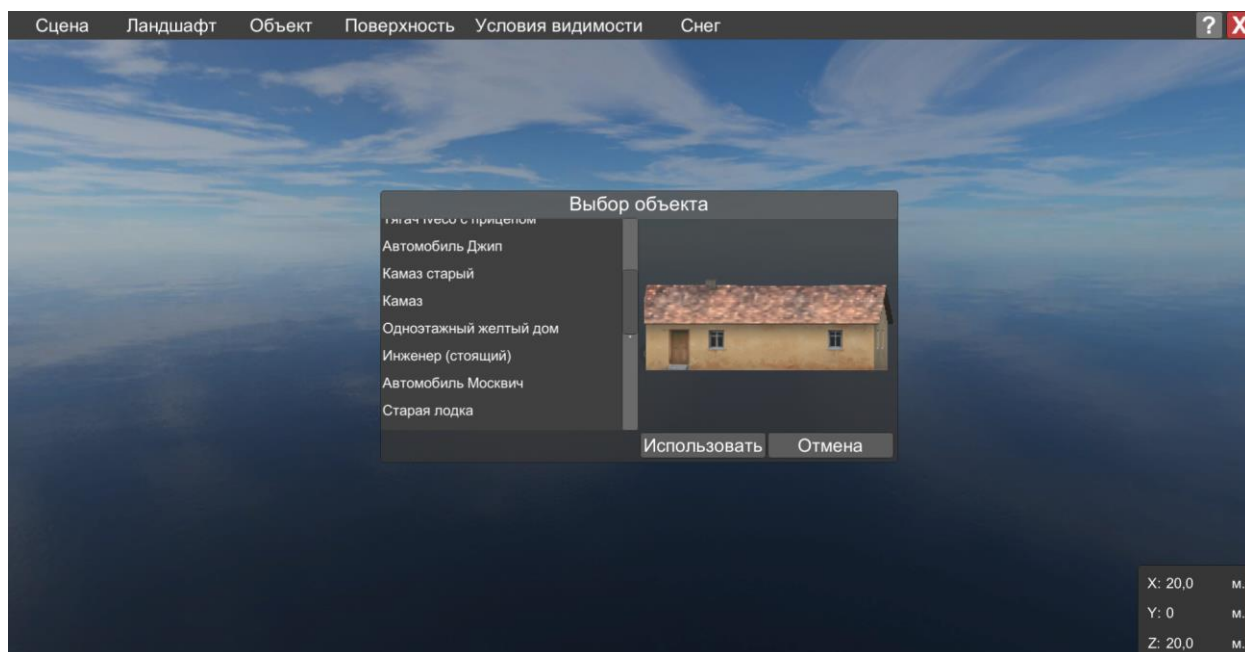


Рисунок 12 – Диалоговое окно для выбора объекта

При клике левой кнопкой мыши на добавленный на местность редактируемый объект пользователю будет выведено меню настройки параметров объекта. В данном окне доступно изменение значения температуры, а также координат положения объекта на местности и его поворота. Сохранение изменений параметров объекта происходит по нажатию на кнопку «Сохранить», его удаление со сцены – по нажатию на кнопку «Удалить».

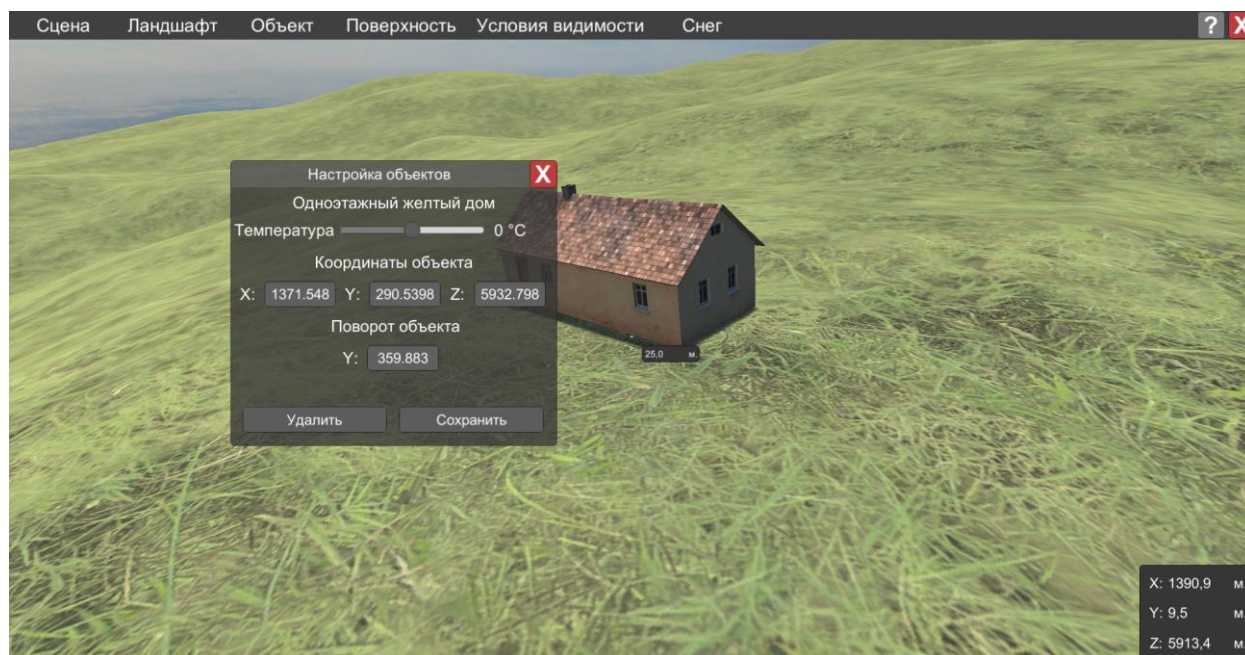


Рисунок 13 – Диалоговое окно для изменения параметров объекта

Заключение

В ходе выполнения курсового проекта 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, а также были описаны особенности реализуемого блока программной системы.

Так как все задачи курсового проекта были выполнены, а требуемое программное обеспечение было реализовано, курсовой проект можно считать выполненным.

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. Торн, А. Искусство создания сценариев в Unity [Текст] / А. Торн. – Москва: ДМК Пресс, 2016. – 360 с.
5. Хокинг, Дж. Unity в действии. Мультиплатформенная разработка на C# [Текст] / Дж. Хокинг. – СПб.: Питер, 2016. – 336 с.

Приложение А

(обязательное)

Исходный текст программы

Файл Editor.cs

```

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

public class Editor : MonoBehaviour {
    private EditorMode mode = EditorMode.editing;

    private TerrainData tData;

    public GameObject _camera;
    GameObject RightCamera;
    GameObject BottomRightCamera;
    Camera cameraComp;

    public GameObject objectListLoader;
    public GameObject menuClick;
    public GameObject settingsMenu;
    public GameObject waterMenu;

    public GameObject terrainManager;

    public GameObject editingProjector;

    string myName;

    public GameObject editObject;
    public bool target = true;
    public bool snowSetting = false;

    public void SetEditorMode(EditorMode m) {
        mode = m;
        if (m == EditorMode.snow_setting)
            editingProjector.SetActive(true);
        else editingProjector.SetActive(false);
    }

    public void SetTerrainData(GameObject t) {
        tData = t.GetComponent<Terrain>().terrainData;
    }

    void Start () {
        _camera = transform.Find("MainCamera").gameObject;
        RightCamera = transform.Find("RightCamera").gameObject;
        BottomRightCamera =
transform.Find("BottomRightCamera").gameObject;
        cameraComp = _camera.GetComponent<Camera>();
    }

    void Update () {
        Ray ray = cameraComp.ScreenPointToRay(Input.mousePosition);
        RaycastHit rayHit;

        if (Physics.Raycast(ray, out rayHit)) {

```

```

        if (mode == EditorMode.snow_setting) {
            editingProjector.transform.position = new
Vector3(rayHit.point.x, editingProjector.transform.position.y,
rayHit.point.z);
        }

        if (Input.GetMouseButtonDown(0) && target) {
            if (EventSystem.current.IsPointerOverGameObject()) {
                return;
            }

            Transform objectHit = rayHit.transform;

            if (mode == EditorMode.editing) {
                if (objectHit.GetComponent<SaveableObj>()) {
                    editObject = objectHit.gameObject;
                }
            }

            menuClick.GetComponent<onMenuButtonClick>().onMenuClick(4);

            settingsMenu.GetComponent<settingMenu>().SetEditObject(editObject);
        } else {
            int mask = 1 << 12;
            mask = ~mask;

            Physics.Raycast(rayHit.point + new Vector3(0,
10f, 0), Vector3.down, out rayHit, Mathf.Infinity, mask);
            objectHit = rayHit.transform;

            GameObject instObject;
            myName =
objectListLoader.GetComponent<loadObjectList>().GetActiveObjectName();

            if (!String.IsNullOrEmpty(myName)) {
                instObject =
Instantiate(Resources.Load<GameObject>("EditableObjects/" + myName),
rayHit.point, Quaternion.identity);
                instObject.name = myName;

                SaveableObj so =
instObject.GetComponent<SaveableObj>();

                if ( (objectHit.GetComponent<Terrain>() &&
so.swimmingObject) || (objectHit.GetComponent<WaterScript>() &&
!so.swimmingObject) || (objectHit.GetComponent<IceScript>() &&
so.swimmingObject) ) {
                    Destroy(instObject);
                    return;
                }

                if (instObject.tag != "YOrient") {
                    instObject.transform.rotation =
Quaternion.FromToRotation(Vector3.up, rayHit.normal);
                }
            }
        }
    }
    else if (mode == EditorMode.snow_setting &&
!objectHit.GetComponent<WaterScript>()) {
        RaycastHit downRayHit;
        Physics.Raycast(rayHit.point - new Vector3(0, 0.01f,
0), Vector3.down, out downRayHit);
    }
}

```

```

        if (objectHit.GetComponent<Terrain>()) {
StartCoroutine(terrainManager.GetComponent<SnowTerrainGenerator>().SetSnow(rayHit.point, false));
        }
        else
        if (objectHit.GetComponent<IceScript>()) {
StartCoroutine(terrainManager.GetComponent<SnowTerrainGenerator>().SetSnow(rayHit.point, true));
        }
        else
        if (downRayHit.transform.GetComponent<IceScript>()) {
StartCoroutine(terrainManager.GetComponent<SnowTerrainGenerator>().SetSnow(downRayHit.point, true));
        }
        }
        snowSetting = true;
    }
}

if (Input.GetMouseButtonDown(1)) {
    Transform objectHit = rayHit.transform;
    if (objectHit.GetComponent<WaterScript>()) {
menuClick.GetComponent<onMenuButtonClick>().onMenuClick(7);

waterMenu.GetComponent<waterMenu>().SetActiveWater(objectHit.gameObject);
    }
    else if (objectHit.GetComponent<IceScript>()) {
menuClick.GetComponent<onMenuButtonClick>().onMenuClick(7);
        GameObject water = objectHit.parent.gameObject;

waterMenu.GetComponent<waterMenu>().SetActiveWater(water);
    }
}

if (Input.GetMouseButton(0) && snowSetting && mode ==
EditorMode.snow_setting) {
    if (EventSystem.current.IsPointerOverGameObject()) {
        return;
    }

    Transform objectHit = rayHit.transform;

    RaycastHit downRayHit;
    Physics.Raycast(rayHit.point - new Vector3(0, 0.01f, 0),
Vector3.down, out downRayHit);

    if (objectHit.GetComponent<Terrain>()) {
StartCoroutine(terrainManager.GetComponent<SnowTerrainGenerator>().SetSnow(rayHit.point, false));
    } else
    if (objectHit.GetComponent<IceScript>()) {
StartCoroutine(terrainManager.GetComponent<SnowTerrainGenerator>().SetSnow(rayHit.point, true));
    } else
    if (downRayHit.transform.GetComponent<IceScript>()) {

```

```

StartCoroutine(terrainManager.GetComponent<SnowTerrainGenerator>().SetSnow(do
wnRayHit.point, true));
    }
    }
}
}

```

Файл EditorMovement.cs

```

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

public class EditorMovement : MonoBehaviour {
    public Text XPos, YPos, ZPos;

    public enum RotationAxes {
        MouseXAndY = 0,
        MouseX = 1,
        MouseY = 2
    }

    public RotationAxes axes = RotationAxes.MouseXAndY;

    public float speed = 5f;

    public float sensitivityHor = 9.0f;
    public float sensitivityVert = 9.0f;
    public float minimumVert = -45.0f;
    public float maximumVert = 45.0f;

    private float _rotationX = 0;

    float moveX;
    float moveZ;
    float moveY;

    Ray ray;
    RaycastHit hit;

    Vector3 move;
    CharacterController characterController;

    void Start() {
        characterController = GetComponent<CharacterController>();
    }

    void Update() {
        RaycastHit downHit;

        XPos.text = string.Format("{0:0.0}", transform.position.x);
        if (Physics.Raycast(transform.position, Vector3.down, out
downHit)) {
            YPos.text = string.Format("{0:0.0}",
Vector3.Distance(transform.position, downHit.point));
        } else {
            YPos.text = "0";
        }

        ZPos.text = string.Format("{0:0.0}", transform.position.z);
    }
}

```

```

        moveX = Input.GetAxis("Horizontal") * speed;
        moveZ = Input.GetAxis("Vertical") * speed;

        move = new Vector3(moveX * Time.deltaTime, moveY *
Time.deltaTime, moveZ * Time.deltaTime);
        move = transform.TransformDirection(move);
        characterController.Move(move);

        if (Input.GetMouseButton(2)) {
            if (axes == RotationAxes.MouseX) {
                transform.Rotate(0, Input.GetAxis("Mouse X") *
sensitivityHor, 0);
            }
            else
            if (axes == RotationAxes.MouseY) {
                _rotationX -= Input.GetAxis("Mouse Y") * sensitivityVert;
                _rotationX = Mathf.Clamp(_rotationX, minimumVert,
maximumVert);

                float rotationY = transform.localEulerAngles.y;

                transform.localEulerAngles = new Vector3(_rotationX,
rotationY, 0);
            }
            else {
                _rotationX -= Input.GetAxis("Mouse Y") * sensitivityVert;
                _rotationX = Mathf.Clamp(_rotationX, minimumVert,
maximumVert);

                float delta = Input.GetAxis("Mouse X") * sensitivityHor;
                float rotationY = transform.localEulerAngles.y + delta;

                transform.localEulerAngles = new Vector3(_rotationX,
rotationY, 0);
            }
        }
    }
}

```

Файл EditorModes.cs

```

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

public enum EditorMode {
    editing,
    snow_setting,
    water_setting
};

```

Файл Surfaces.cs

```

public enum Surfaces {
    swamp,
    field,
    bush,
    forest,
    rock
};

```

Файл LandscapeLoader.cs

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class LandscapeLoader : MonoBehaviour
{
    public GameObject wallPrefab;

    GameObject[] walls = new GameObject[5];

    public GameObject water_prefab;

    public GameObject t;

    private string loadTerrainName = "";

    public string GetLoadLandscapeName() {
        return loadTerrainName;
    }

    public void LoadLandscape(string filename)
    {
        loadTerrainName = filename;

        if (GameObject.Find("CreateTerrain"))
            Destroy(GameObject.Find("CreateTerrain"));
        if (GameObject.Find("SnowTerrain"))
            Destroy(GameObject.Find("SnowTerrain"));

        Vector3 position = new Vector3(0, 0.1f, 0);

        GameObject terrain_object =
            Instantiate(Resources.Load<GameObject>("Landscapes/" + loadTerrainName),
                position, Quaternion.identity);

        terrain_object.name = "CreateTerrain";
        terrain_object.gameObject.tag = "CreateTerrain";

        GetComponent<MapCreator>().CreateMaps();

        SetBorderWalls(terrain_object.GetComponent<Terrain>().terrainData.size.x);

        GetComponent<SnowTerrainGenerator>().CreateSnowTerrain(terrain_object);
    }

    public void SetBorderWalls(float tResolution) {
        foreach (GameObject b in walls) {
            if (b != null) Destroy(b);
        }

        walls[0] = Instantiate(wallPrefab, new Vector3(0, 5000,
            tResolution / 2), Quaternion.identity);
        walls[0].transform.localScale = new Vector3(1, 10000,
            tResolution);

        walls[1] = Instantiate(wallPrefab, new Vector3(tResolution / 2,
            5000, tResolution), Quaternion.identity);
        walls[1].transform.localScale = new Vector3(tResolution, 10000,
            1);

        walls[2] = Instantiate(wallPrefab, new Vector3(tResolution, 5000,
            tResolution / 2), Quaternion.identity);
        walls[2].transform.localScale = new Vector3(1, 10000,
            tResolution);
    }
}

```

```

        walls[3] = Instantiate(wallPrefab, new Vector3(tResolution / 2,
5000, 0), Quaternion.identity);
        walls[3].transform.localScale = new Vector3(tResolution, 10000,
1);

        walls[4] = Instantiate(wallPrefab, new Vector3(tResolution / 2,
10000, tResolution / 2), Quaternion.identity);
        walls[4].transform.localScale = new Vector3(tResolution, 1,
tResolution);
    }
}

```

Файл MapCreator.cs

```

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

public class MapCreator : MonoBehaviour
{
    public GameObject terrain;
    public TerrainData terrainData;

    public Texture2D temperatureTexture;
    public Material temperatureMaterial;

    public Texture2D eprTexture;
    public Material eprMaterial;

    public void CreateMaps() {
        Debug.Log("123");
        terrain = GameObject.FindWithTag("CreateTerrain");
        terrainData = terrain.GetComponent<Terrain>().terrainData;

        float[] temps =
GetComponent<SurfacesManager>().GetTemperatures();
        float[] eprs = GetComponent<SurfacesManager>().GetEprs();

        temperatureTexture.Resize(terrainData.alphamapWidth,
terrainData.alphamapHeight);
        eprTexture.Resize(terrainData.alphamapWidth,
terrainData.alphamapHeight);

        Color tempColor = new Color();
        Color eprColor = new Color();

        float[, ,] alphamapData = terrainData.GetAlphamaps(0, 0,
terrainData.alphamapWidth, terrainData.alphamapHeight);
        int layersCount = terrainData.alphamapLayers;

        float max_value;

        for (int y = 0; y < terrainData.alphamapHeight; y++) {
            for (int x = 0; x < terrainData.alphamapWidth; x++) {
                float[] values = new float[layersCount];

                for (int i = 0; i < layersCount; i++) {
                    values[i] = alphamapData[y, x, i];
                }

                max_value = Mathf.Max(values);
            }
        }
    }
}

```

```

        for (int i = 0; i < values.Length; i++) {
            if (values[i] == max_value) {
                float color = temps[i] / 100;
                tempColor.r = tempColor.g = tempColor.b = color;

                color = eprs[i] / 100;
                eprColor.r = eprColor.g = eprColor.b = color;
                break;
            }
        }

        temperatureTexture.SetPixel(x, y, tempColor);
        eprTexture.SetPixel(x, y, eprColor);
    }
}

temperatureTexture.Apply();
byte[] bytes = temperatureTexture.EncodeToPNG();
string filename = Application.streamingAssetsPath +
"/Maps/TemperatureMap.png";
File.WriteAllBytes(filename, bytes);

eprTexture.Apply();
bytes = eprTexture.EncodeToPNG();
filename = Application.streamingAssetsPath + "/Maps/EprMap.png";
File.WriteAllBytes(filename, bytes);

GetComponent<SurfacesManager>().SetTempsForSurfaces();
}
}

```

Файл Helper.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using UnityEngine.UI;
using System.Xml.Linq;
using System.IO;
using System;
using UnityEngine.SceneManagement;

public class Helper : MonoBehaviour {
    public GameObject terrainManager;
    public GameObject weatherController;
    public GameObject editor;

    private string SavePath;

    public List<SaveableObj> objects = new List<SaveableObj>();
    public List<WaterScript> waterObjects = new List<WaterScript>();

    GameObject ed;
    public string terrain_name;

    XElement root = new XElement ("Save");

    void Start() {
        SavePath = Application.streamingAssetsPath + "/Saves/";

        ed = GameObject.Find("Editor");
    }
}

```



```

    }

    private string currentSceneName = "";
    public InputField saveName;

    public void ClearSceneName()
    {
        currentSceneName = "";
    }

    public void SetSceneName(InputField input)
    {
        currentSceneName = input.text;
        Debug.Log(currentSceneName);
    }

    public void TrySave(GameObject dialog)
    {
        if (currentSceneName == "")
        {
            dialog.SetActive(true);
        }
        else
        {
            saveName.text = currentSceneName;
            Save(saveName);
        }
    }

    public void Save(InputField saveName) {

        XElement root = new XElement ("Save");

        XElement TimeElement = new XElement("Time",
DateTime.Now.ToString("HH:mm:ss"));
        root.Add(TimeElement);

        XElement TerrainElement = new XElement("Landscape");
        TerrainElement.Add(new XAttribute("Path",
terrainManager.GetComponent<LandscapeLoader>().GetLoadLandscapeName()));
        root.Add(TerrainElement);

        XElement snowTElement = new XElement("SnowMap");
        string[] tmp = saveName.text.Split('.');

        terrainManager.GetComponent<SnowTerrainGenerator>().GenerateSnowHeightMapWithSave(tmp[0]);
        snowTElement.Add(new XAttribute("Map", tmp[0]));
        root.Add(snowTElement);

        XElement edit = new XElement("Editor", ed.transform.position);
        root.Add(edit);

        XElement sTempElement =
terrainManager.GetComponent<SurfacesManager>().GetTemperatureElement();
        root.Add(sTempElement);

        XElement waterElement = new XElement("Water");
        foreach (WaterScript waterObj in waterObjects) {
            XElement saveObj = waterObj.GetWaterElementWithParameters();
            waterElement.Add(saveObj);
        }
        root.Add(waterElement);
    }

```

```

        XElement SaveObjects = new XElement ("EditableObjects");
        root.Add(SaveObjects);
        foreach (SaveableObj obj in objects) {
            XElement saveObj = obj.GetElement ();
            SaveObjects.Add (saveObj);
        }

root.Add(weatherController.GetComponent<WeatherSaver>().GetWeatherAndLightEle
ment());

        XDocument saveDoc = new XDocument (root);

        if(!saveName.text.EndsWith(".xml"))
        {
            saveName.text = saveName.text + ".xml";
        }
        File.WriteAllText (SavePath + saveName.text, saveDoc.ToString ());
        currentSceneName = saveName.text;
        Debug.Log(currentSceneName);
    }

    public void Load (InputField loadName) {
        XElement root = null;

        if (!File.Exists (SavePath)) {
            Debug.Log ("Save not found!");
            if (File.Exists (Application.streamingAssetsPath + "/Saves/"
+ loadName.text))
                root = XDocument.Parse (File.ReadAllText
(Application.streamingAssetsPath + "/Saves/" + loadName.text)).Element
("Save");
        }
        else {
            root = XDocument.Parse (File.ReadAllText (SavePath)).Element
("Save");
            Debug.Log ("Load fine!");
        }

        if (root == null) {
            Debug.Log ("Level load failed!");
            return;
        }

        if (GameObject.Find("CreateTerrain"))
            Destroy(GameObject.Find("CreateTerrain"));
        if (GameObject.Find("SnowTerrain"))
            Destroy(GameObject.Find("SnowTerrain"));

        WaterScript[] water_objects = FindObjectsOfType<WaterScript>();
        for (int i = 0; i < water_objects.Length; i++) {
            Destroy(water_objects[i].gameObject);
        }

        StartCoroutine(Wait(root));
    }

    IEnumerator Wait(XElement r) {
        print(Time.time);
        Debug.Log("Pregen");
        yield return new WaitForSeconds(0.5f);
        Debug.Log("gen");
    }

```

```

        GenerateScene(r);
    }

    private void GenerateScene(XElement root)
    {
        GameObject instObject;

        foreach(SaveableObj obj in objects)
        {
            obj.DestroySelf();
        }

        XElement sTEl = root.Element("SurfaceTemperature");

        terrainManager.GetComponent<SurfacesManager>().SetSurfacesTemperature(float.Parse(sTEl.Element("Forest").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Field").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Bush").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Swamp").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Rock").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Water").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Snow").Attribute("Temperature").Value),

float.Parse(sTEl.Element("Ice").Attribute("Temperature").Value));

        XElement TerrainElement = root.Element("Landscape");

        terrainManager.GetComponent<LandscapeLoader>().LoadLandscape(TerrainElement.Attribute("Path").Value);

        XElement snowTEl = root.Element("SnowMap");

        terrainManager.GetComponent<SnowTerrainGenerator>().CreateTerrainFromHeightmapSave(snowTEl.Attribute("Map").Value);

        XElement TimeElement = root.Element("Time");

        XElement waterElement = root.Element("Water");
        foreach (XElement waterEl in waterElement.Elements("Water")) {
            Vector3 position = Vector3.zero;
            position.x = float.Parse(waterEl.Attribute("XPos").Value);
            position.y = float.Parse(waterEl.Attribute("YPos").Value);
            position.z = float.Parse(waterEl.Attribute("ZPos").Value);

            Collider[] hitColliders = Physics.OverlapSphere(position,

0.001f);

            WaterScript ws = hitColliders[0].GetComponent<WaterScript>();

ws.SetIceHeight(float.Parse(waterEl.Attribute("IceHeight").Value));

ws.SetWavesMark(int.Parse(waterEl.Attribute("WavesMark").Value));
            ws.SetDepth(float.Parse(waterEl.Attribute("Depth").Value));
        }
    }

```

```

        XElement SaveObjects = root.Element("EditableObjects");
        foreach (XElement instance in SaveObjects.Elements("Object")) {
            Vector3 position = Vector3.zero;
            Vector3 rotation = Vector3.zero;

            XElement modelNameElement = instance.Element("ModelName");
            XElement temperature_element =
instance.Element("Temperature");

            XElement pos_element = instance.Element("Position");
            position.x = float.Parse(pos_element.Attribute
("XPos").Value);
            position.y = float.Parse(pos_element.Attribute
("YPos").Value);
            position.z = float.Parse(pos_element.Attribute
("ZPos").Value);

            XElement rot_element = instance.Element("Rotation");
            rotation.x =
float.Parse(rot_element.Attribute("XRot").Value);
            rotation.y =
float.Parse(rot_element.Attribute("YRot").Value);
            rotation.z =
float.Parse(rot_element.Attribute("ZRot").Value);

            Debug.Log("Rotation:" + rotation.x + " ; " + rotation.y + " ;
" + rotation.z);

            GameObject new_object =
Instantiate(Resources.Load<GameObject>("EditableObjects/" +
modelNameElement.Value), position, Quaternion.identity) as GameObject;

            new_object.transform.rotation = Quaternion.Euler(rotation);
            new_object.name = modelNameElement.Value;

            new_object.GetComponent<SaveableObj>().temperature =
float.Parse(temperature_element.Value);
        }

        if (weatherController != null) {
            XElement weatherElement =
root.Element("VisibilityConditions");
            XElement lightEl = weatherElement.Element("Lighting");

            weatherController.GetComponent<WeatherSaver>().SetLightProperties((LightSource
eType) Enum.Parse(typeof(LightSourceType),
lightEl.Attribute("LightSourceType").Value),

float.Parse(lightEl.Attribute("LightIntensity").Value),

float.Parse(lightEl.Attribute("XLightSourcePos").Value),

float.Parse(lightEl.Attribute("ZLightSourcePos").Value));

            Debug.Log("Light X: " +
lightEl.Attribute("XLightSourcePos").Value);
            XElement wCondEl =
weatherElement.Element("WeatherConditions");

            weatherController.GetComponent<WeatherSaver>().SetWeatherConditions((WeatherT
ype) Enum.Parse(typeof(WeatherType), wCondEl.Attribute("RainfallType").Value),

float.Parse(wCondEl.Attribute("RainfallIntensity").Value));

```

```

        weatherController.GetComponent<WeatherSaver>().SetSettings();
    }

    onSliderChangeValue[] sliders =
FindObjectsOfTypeAll<onSliderChangeValue>() as onSliderChangeValue[];
    Debug.Log("Sliders: " + sliders.Length);
    for (int i = 0; i < sliders.Length; i++) {
        sliders[i].ChangeSliderValue();
    }
}
}

```

Файл SaveableObj.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Xml.Linq;

public class SaveableObj : MonoBehaviour {

    [SerializeField]
    public string objectName;
    [SerializeField]
    public float temperature = 273.15f;
    public float epr = 75f;
    private Helper helper;

    public bool swimmingObject;

    public Texture temperatureTexture;

    GameObject ed;

    public string ON()
    {
        return objectName;
    }

    private void Awake()
    {
        helper = FindObjectOf<Helper> ();
    }

    private void Start ()
    {
        helper.objects.Add (this);
        ed = GameObject.Find("Editor");
    }

    private void OnDestroy()
    {
        helper.objects.Remove (this);
    }

    public XElement GetElement()
    {
        XElement name_element = new XElement("Name", objectName);
        XElement modelName_element = new XElement("ModelName", name);
        XElement temperature_element = new XElement("Temperature",
temperature);
        XElement dist_element = new XElement("Distance", GetDistance());
    }
}

```

```

        XAttribute xPos = new XAttribute("XPos", transform.position.x);
        XAttribute yPos = new XAttribute("YPos", transform.position.y);
        XAttribute zPos = new XAttribute("ZPos", transform.position.z);
        XElement pos_element = new XElement("Position", xPos, yPos,
zPos);

        XAttribute xRot = new XAttribute("XRot",
transform.eulerAngles.x);
        XAttribute yRot = new XAttribute("YRot",
transform.eulerAngles.y);
        XAttribute zRot = new XAttribute("ZRot",
transform.eulerAngles.z);
        XElement rot_element = new XElement("Rotation", xRot, yRot,
zRot);

        XElement element = new XElement("Object", name_element,
modelName_element, temperature_element, dist_element, pos_element,
rot_element);

        return element;
    }

    float GetDistance() {
        return Vector3.Distance(this.gameObject.transform.position,
ed.transform.position);
    }

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

```

Файл SurfacesManager.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Xml.Linq;

[System.Serializable]
public class Surface {
    public Surfaces type;
    public string name;
    public float epr;
    public float temperature;
    public Texture2D texture;
}

public class SurfacesManager : MonoBehaviour
{
    public GameObject areaMenu;

    public Surface forest = new Surface();
    public Surface field = new Surface();
    public Surface bush = new Surface();
    public Surface swamp = new Surface();
    public Surface rock = new Surface();
    public Surface water = new Surface();
    public Surface snow = new Surface();
}

```

```

public Surface ice = new Surface();

private void Awake() {
    forest.epr = -17;
    field.epr = -15;
    bush.epr = -23;
    swamp.epr = -15;
    rock.epr = -20;
    water.epr = -35;
    snow.epr = -11;
    ice.epr = -13;
}

public void SetSurfacesTemperature(float forestT, float fieldT, float
bushT, float swampT, float rockT, float waterT, float snowT, float iceT) {
    forest.temperature = forestT;
    field.temperature = fieldT;
    bush.temperature = bushT;
    swamp.temperature = swampT;
    rock.temperature = rockT;
    water.temperature = waterT;
    snow.temperature = snowT;
    ice.temperature = iceT;

    if (areaMenu != null) {
        areaMenu.GetComponent<areaMenuScript>().LoadTemperature();
    }
}

public float[] GetTemperatures() {
    float[] temps = { swamp.temperature, field.temperature,
bush.temperature, forest.temperature, rock.temperature };
    return temps;
}

public float[] GetEprs() {
    float[] eprs = { swamp.epr, field.epr, bush.epr, forest.epr,
rock.epr };
    return eprs;
}

public void SetTemperatureForWater() {
    var waters = FindObjectsOfType<WaterScript>();
    for (int i = 0; i < waters.Length; i++) {
        waters[i].SetTemperature(water.temperature);
    }
}

public void SetTemperatureForIce() {
    var ices = FindObjectsOfType<IceScript>();
    for (int i = 0; i < ices.Length; i++) {
        ices[i].SetTemperature(ice.temperature);
    }
}

public void SetTempsForSurfaces() {
    SetTemperatureForWater();
    SetTemperatureForIce();
}

public XElement GetTemperatureElement() {
    XElement tempRoot = new XElement("SurfaceTemperature");

```

```

XElement fo = new XElement("Forest");
fo.Add(new XAttribute("Temperature", forest.temperature));

XElement fi = new XElement("Field");
fi.Add(new XAttribute("Temperature", field.temperature));

XElement bu = new XElement("Bush");
bu.Add(new XAttribute("Temperature", bush.temperature));

XElement sw = new XElement("Swamp");
sw.Add(new XAttribute("Temperature", swamp.temperature));

XElement ro = new XElement("Rock");
ro.Add(new XAttribute("Temperature", rock.temperature));

XElement wa = new XElement("Water");
wa.Add(new XAttribute("Temperature", water.temperature));

XElement sn = new XElement("Snow");
sn.Add(new XAttribute("Temperature", snow.temperature));

XElement ic = new XElement("Ice");
ic.Add(new XAttribute("Temperature", ice.temperature));

tempRoot.Add(fo, fi, bu, sw, ro, wa, sn, ic);

return tempRoot;
}
}

```

Файл areaMenuScript.cs

```

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

public class areaMenuScript : MonoBehaviour {
    public Slider forestTslider;
    public Slider fieldTslider;
    public Slider bushTslider;
    public Slider swampTslider;
    public Slider rockTslider;
    public Slider waterTslider;
    public Slider snowTslider;
    public Slider iceTslider;

    public GameObject terrainManager;

    public void SetTemperature() {

terrainManager.GetComponent<SurfacesManager>().SetSurfacesTemperature(forestT
slider.value,

fieldTslider.value,

bushTslider.value,

swampTslider.value,

rockTslider.value,

waterTslider.value,

```



```

snowTslider.value,
iceTslider.value);
    }

    public void LoadTemperature() {
        SurfacesManager sm =
terrainManager.GetComponent<SurfacesManager>();

        forestTslider.value = sm.forest.temperature;
        fieldTslider.value = sm.field.temperature;
        bushTslider.value = sm.bush.temperature;
        swampTslider.value = sm.swamp.temperature;
        rockTslider.value = sm.rock.temperature;
        waterTslider.value = sm.water.temperature;
        snowTslider.value = sm.snow.temperature;
        iceTslider.value = sm.ice.temperature;
    }
}

```

Файл loadTerrainList.cs

```

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

public class loadTerrainList : MonoBehaviour {

    [SerializeField] Transform menuPanel;
    [SerializeField] GameObject buttonPrefab;

    public GameObject creationPosition;
    public RuntimeAnimatorController controller;
    private GameObject obj;

    public GameObject terrainManager;

    [SerializeField]
    private string activeTerrainName;

    void Start() {
        string subfolder_path = "Landscapes";
        Object[] landscapes = Resources.LoadAll(subfolder_path,
typeof(GameObject));

        for (int i = 0; i < landscapes.Length; i++) {
            GameObject button = (GameObject)Instantiate(buttonPrefab);

            string modelName = landscapes[i].name;
            button.GetComponentInChildren<Text>().text = modelName;

            button.GetComponent<Button>().onClick.AddListener(delegate {
ImportObject(modelName); });
            button.GetComponent<Button>().onClick.AddListener(delegate {
SetActiveTerrainName(modelName); });
            button.transform.SetParent(menuPanel);
            button.transform.localScale = new Vector3(1f, 1f, 1f);
        }
    }

    public void ImportObject(string filename)

```

```

    {
        DestroyTerrain();

        obj = Instantiate(Resources.Load<GameObject>("Landscapes/" +
filename), creationPosition.transform.position, Quaternion.identity);

        obj.layer = LayerMask.NameToLayer("PreviewTerrain");
        foreach (Transform trans in
obj.transform.GetComponentsInChildren<Transform>(true))
        {
            trans.gameObject.layer =
LayerMask.NameToLayer("PreviewTerrain");
        }
    }

    public void DestroyTerrain()
    {
        if(obj)
        {
            Destroy(obj);
        }
    }

    public void SetActiveTerrainName(string name) {
        activeTerrainName = name;
    }

    public void LoadLandscape() {

terrainManager.GetComponent<LandscapeLoader>().LoadLandscape(activeTerrainNam
e);
    }
}

```

Файл loadObjectList.cs

```

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

public class loadObjectList : MonoBehaviour
{
    [SerializeField] Transform menuPanel;
    [SerializeField] GameObject buttonPrefab;

    public GameObject creationPosition;
    public RuntimeAnimatorController controller;
    private GameObject obj;

    public Camera previewCamera;

    private string activeObjectName;

    void Start()
    {
        string subfolder_path = "EditableObjects";
        Object[] useObjects = Resources.LoadAll(subfolder_path,
typeof(GameObject));

        for (int i = 0; i < useObjects.Length; i++)
        {

```

```

        GameObject button = (GameObject) Instantiate(buttonPrefab);

        GameObject tempObject =
        (GameObject) Instantiate(useObjects[i]);

        string modelName = useObjects[i].name;
        button.GetComponentInChildren<Text>().text =
tempObject.GetComponent<SaveableObj>().objectName;

        Destroy(tempObject);

        button.GetComponent<Button>().onClick.AddListener(delegate {
ImportObject(modelName); });
        button.GetComponent<Button>().onClick.AddListener(delegate {
SetActiveObjectName(modelName); });
        button.transform.SetParent(menuPanel);
        button.transform.localScale = new Vector3(1f, 1f, 1f);
    }
}

public void ImportObject(string objectName)
{
    DestroyObject();

    GameObject terrain =
    (GameObject) Resources.Load("EditableObjects/" + objectName);

    float colSizeX = 0, colSizeY = 0;

    if (terrain.GetComponent<SaveableObj>()) {
        if (terrain.GetComponent<BoxCollider>().size.z >
terrain.GetComponent<BoxCollider>().size.x) {
            colSizeX = terrain.GetComponent<BoxCollider>().size.z *
terrain.transform.localScale.z; // Корректировка размера по Z
        }
        else {
            colSizeX = terrain.GetComponent<BoxCollider>().size.x *
terrain.transform.localScale.x; // Корректировка размера по X
        }
        colSizeY = terrain.GetComponent<BoxCollider>().size.y;
        if (colSizeX > colSizeY)
        {
            previewCamera.orthographicSize = colSizeX / 2;
            obj = Instantiate(terrain,
creationPosition.transform.position + new Vector3(0, (-colSizeY) / 2,
colSizeX), new Quaternion(0, 0, 0, 0));
        }
        else
        {
            previewCamera.orthographicSize = colSizeY / 2;
            obj = Instantiate(terrain,
creationPosition.transform.position + new Vector3(0, (-colSizeY) / 2,
colSizeY), new Quaternion(0, 0, 0, 0));
        }
    } else {
        obj = Instantiate(terrain,
creationPosition.transform.position, new Quaternion(0, 0, 0, 0));
    }

    obj.layer = LayerMask.NameToLayer("PreviewObject");
    foreach (Transform trans in
obj.transform.GetComponentsInChildren<Transform>(true))
    {

```

```

        trans.gameObject.layer =
LayerMask.NameToLayer("PreviewObject");
    }

    Animator anim = obj.AddComponent<Animator>();
    anim.runtimeAnimatorController = controller;
}

public void DestroyObject()
{
    if (obj)
    {
        Destroy(obj);
    }
}

public void SetActiveObjectName(string model) {
    activeObjectName = model;
    Debug.Log(activeObjectName);
}

public string GetActiveObjectName() {
    return activeObjectName;
}
}

```

Файл loadScene.cs

```

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

public class loadScene : MonoBehaviour
{
    [SerializeField] Transform menuPanel;
    [SerializeField] GameObject buttonPrefab;

    public InputField filepathInput,
        filenameInput;

    public Button loadButton;

    void Start()
    {
        RefreshList();
    }

    public void RefreshList()
    {
        ClearDirectoryList();
        string[] files =
System.IO.Directory.GetFiles(Application.streamingAssetsPath + "/Saves/",
"*.xml");
        for (int i = 0; i < files.Length; i++)
        {
            GameObject button = (GameObject)Instantiate(buttonPrefab);

            string[] tmp = files[i].Split('/');
            files[i] = tmp[tmp.Length - 1];

            button.GetComponentInChildren<Text>().text = files[i];
        }
    }
}

```

```

        button.GetComponent<Button>().onClick.AddListener(delegate {
OpenFile(button); });
        button.transform.SetParent(menuPanel);
        button.transform.localScale = new Vector3(1f, 1f, 1f);
    }
}

public void ClearDirectoryList()
{
    foreach (RectTransform btn in menuPanel)
    {
        Destroy(btn.gameObject);
    }

    filenameInput.text = "";
    loadButton.interactable = false;
}

public void OpenFile(GameObject btn)
{
    filenameInput.text = btn.GetComponentInChildren<Text>().text;
    loadButton.interactable = true;
}
}

```

Файл saveScene.cs

```

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

public class saveScene : MonoBehaviour
{
    [SerializeField] Transform menuPanel;
    [SerializeField] GameObject buttonPrefab;

    public InputField filepathInput,
        filenameInput;

    public Button saveButton;

    void Start()
    {
        string[] files =
System.IO.Directory.GetFiles(Application.streamingAssetsPath + "/Saves/",
"*.xml");
        for (int i = 0; i < files.Length; i++)
        {
            GameObject button = (GameObject)Instantiate(buttonPrefab);

            string[] tmp = files[i].Split('/');
            files[i] = tmp[tmp.Length - 1];

            button.GetComponentInChildren<Text>().text = files[i];

            button.GetComponent<Button>().onClick.AddListener(delegate {
OpenFile(button); });
            button.transform.SetParent(menuPanel);
            button.transform.localScale = new Vector3(1f, 1f, 1f);
        }
    }
}

```

```

    public void RefreshList()
    {
        ClearDirectoryList();
        string[] files =
System.IO.Directory.GetFiles(Application.streamingAssetsPath + "/Saves/",
"*.xml");
        for (int i = 0; i < files.Length; i++)
        {
            GameObject button = (GameObject)Instantiate(buttonPrefab);

            string[] tmp = files[i].Split('/');
            files[i] = tmp[tmp.Length - 1];

            button.GetComponentInChildren<Text>().text = files[i];

            button.GetComponent<Button>().onClick.AddListener(delegate {
OpenFile(button); });
            button.transform.SetParent(menuPanel);
            button.transform.localScale = new Vector3(1f, 1f, 1f);
        }
    }

    public void ClearDirectoryList()
    {
        foreach (RectTransform btn in menuPanel)
        {
            Destroy(btn.gameObject);
        }

        filenameInput.text = "";
        saveButton.interactable = false;
    }

    public void CheckFilename()
    {
        if (filenameInput.text == "")
        {
            saveButton.interactable = false;
        }
        else
        {
            saveButton.interactable = true;
        }
    }

    public void OpenFile(GameObject btn)
    {
        filenameInput.text = btn.GetComponentInChildren<Text>().text;
        saveButton.interactable = true;
    }
}

```

Файл settingMenu.cs

```

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

[SerializeField]
public class ObjectSettings {
    public float temperature;
    public Vector3 position;
}

```

```

        public Vector3 rotation;

        public void SetSettings( float temp, Vector3 pos, Vector3 rot) {
            temperature = temp;
            position = pos;
            rotation = rot;
        }
    }

    public class settingMenu : MonoBehaviour
    {
        public GameObject editObject;
        private Transform editTransform;

        public ObjectSettings lastSettings = new ObjectSettings();

        public Text nameText;
        public Slider tempSlider;
        public InputField xPos, yPos, zPos, xRot, yRot, zRot;

        private Vector3 offset = new Vector3(0, 0.1f, 0);

        public void DeleteObject() {
            Destroy(editObject);
        }

        public void SetEditObject(GameObject editObj) {
            if (editObject != null) SetLastSettingsToObject();

            editObject = editObj;
            editTransform = editObject.transform;

            lastSettings.SetSettings(editObj.GetComponent<SaveableObj>().temperature -
            273.15f, editTransform.position, editTransform.transform.eulerAngles);

            nameText.text = editObj.GetComponent<SaveableObj>().objectName;

            tempSlider.value = lastSettings.temperature;

            xPos.text = lastSettings.position.x.ToString();
            yPos.text = lastSettings.position.y.ToString();
            zPos.text = lastSettings.position.z.ToString();

            xRot.text = lastSettings.rotation.x.ToString();
            yRot.text = lastSettings.rotation.y.ToString();
            zRot.text = lastSettings.rotation.z.ToString();

            tempSlider.GetComponent<OnSliderChangeValue>().ChangeSliderValue();
        }

        public void SetLastSettingsToObject() {
            editObject.GetComponent<SaveableObj>().temperature =
            lastSettings.temperature + 273.15f;

            editTransform.position = new Vector3(lastSettings.position.x,
            lastSettings.position.y, lastSettings.position.z);
            editTransform.rotation =
            Quaternion.Euler(lastSettings.rotation.x, lastSettings.rotation.y,
            lastSettings.rotation.z);
        }
    }

```

```

        public void SaveObjectSettings() {
            editObject.GetComponent<SaveableObj>().temperature =
tempSlider.value + 273.15f;

            editTransform.position = new Vector3(float.Parse(xPos.text),
float.Parse(yPos.text), float.Parse(zPos.text));

lastSettings.SetSettings(editObject.GetComponent<SaveableObj>().temperature -
273.15f, editTransform.position, editTransform.transform.eulerAngles);
        }

        public void SetXPos() {
            editTransform.position = new Vector3(float.Parse(xPos.text),
editTransform.position.y, editTransform.position.z);
        }

        public void SetYPos() {
            editTransform.position = new Vector3(editTransform.position.x,
float.Parse(yPos.text), editTransform.position.z);
        }

        public void SetZPos() {
            editTransform.position = new Vector3(editTransform.position.x,
editTransform.position.y, float.Parse(zPos.text));
        }

        public void SetXRot() {

            editTransform.rotation = Quaternion.Euler(float.Parse(xRot.text),
editTransform.rotation.eulerAngles.y, editTransform.rotation.eulerAngles.z);
        }

        public void SetYRot() {
            RaycastHit rayHit;
            Physics.Raycast(editTransform.position + offset, Vector3.down,
out rayHit);

            Debug.Log(rayHit.transform.gameObject.name);

            if (editTransform.tag != "YOrient") {float degree =
float.Parse(yRot.text) - editTransform.rotation.eulerAngles.y;
                Debug.Log("Rotate to: " + degree);
                editTransform.RotateAround(rayHit.point, rayHit.normal,
degree);
            } else {
                editTransform.rotation =
Quaternion.Euler(editTransform.rotation.eulerAngles.x,
float.Parse(yRot.text), editTransform.rotation.eulerAngles.z);
            }
        }

        public void SetZRot() {
            editTransform.rotation =
Quaternion.Euler(editTransform.rotation.eulerAngles.x,
editTransform.rotation.eulerAngles.y, float.Parse(zRot.text));
        }
    }

```