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 местности имитатора закабинного пространства»

Студент _____ Марочкин М.А.

Шифр 170584

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

Орел 2019

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 Марочкин М.А. Шифр 170584

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 местности 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

Описание программного обеспечения

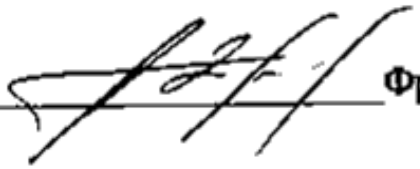
Проектирование программной системы имитатора местности

Проектирование алгоритмов блока редактора местности

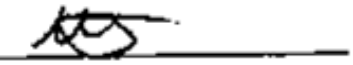
Реализация алгоритмов блока редактора местности

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 9
1.4 Частное задание на разработку 11
2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ ИМИТАТОРА МЕСТНОСТИ 12
2.1 Общая структура 12
2.2 Проектирование блока редактора местности 13
2.3 Диаграмма состояний интерфейса 26
3 ПРОЕКТИРОВАНИЕ АЛГОРИТМОВ БЛОКА РЕДАКТОРА МЕСТНОСТИ 29
3.1 Модуль загрузки ландшафта 29
3.2 Модуль настройки подстилающих поверхностей 31
3.3 Модуль настройки объектов 32
4 РЕАЛИЗАЦИЯ АЛГОРИТМОВ БЛОКА РЕДАКТОРА МЕСТНОСТИ 34
4.1 Реализации алгоритмов класса LandscapeLoader 34
4.2 Реализации алгоритмов класса MapCreator 35
4.3 Реализации алгоритмов класса SurfacesManager 37
4.4 Реализации алгоритмов класса settingMenu 38
ЗАКЛЮЧЕНИЕ 40
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41
ПРИЛОЖЕНИЕ А (ОБЯЗАТЕЛЬНОЕ) ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ 42

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 местности 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 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 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.

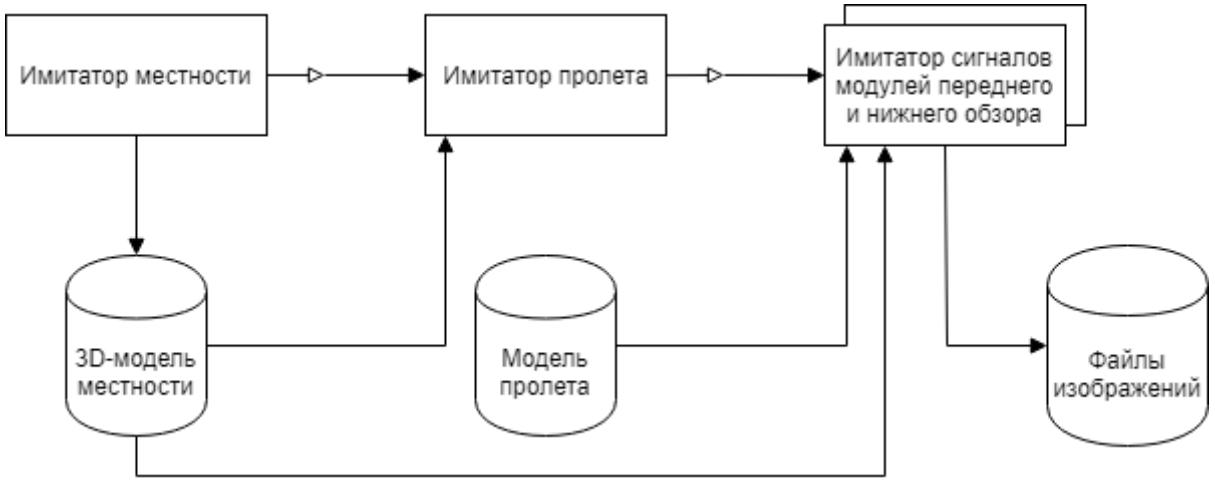


Рисунок 1 – Схема взаимодействия программ программной системы ИЗП

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 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 изображений, 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

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

В данной курсовой работе будут рассмотрены проектирование и реализация алгоритмов блока редактора местности программной системы ИЗП, 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 настройки 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.

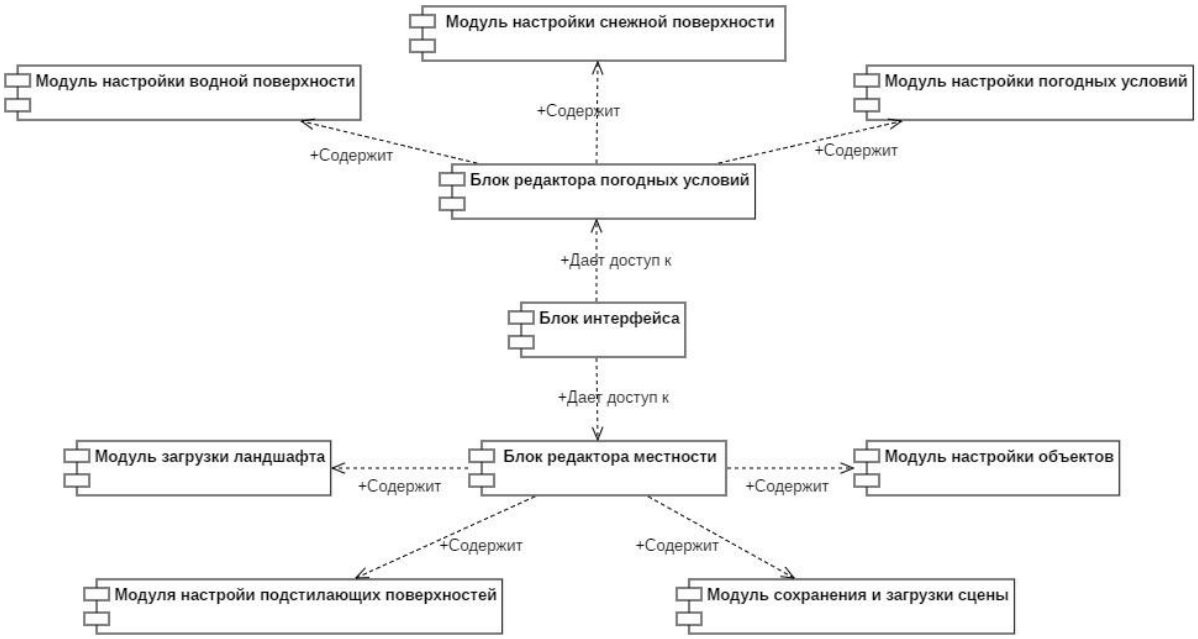


Рисунок 2 – Диаграмма компонентов имитатора местности

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.

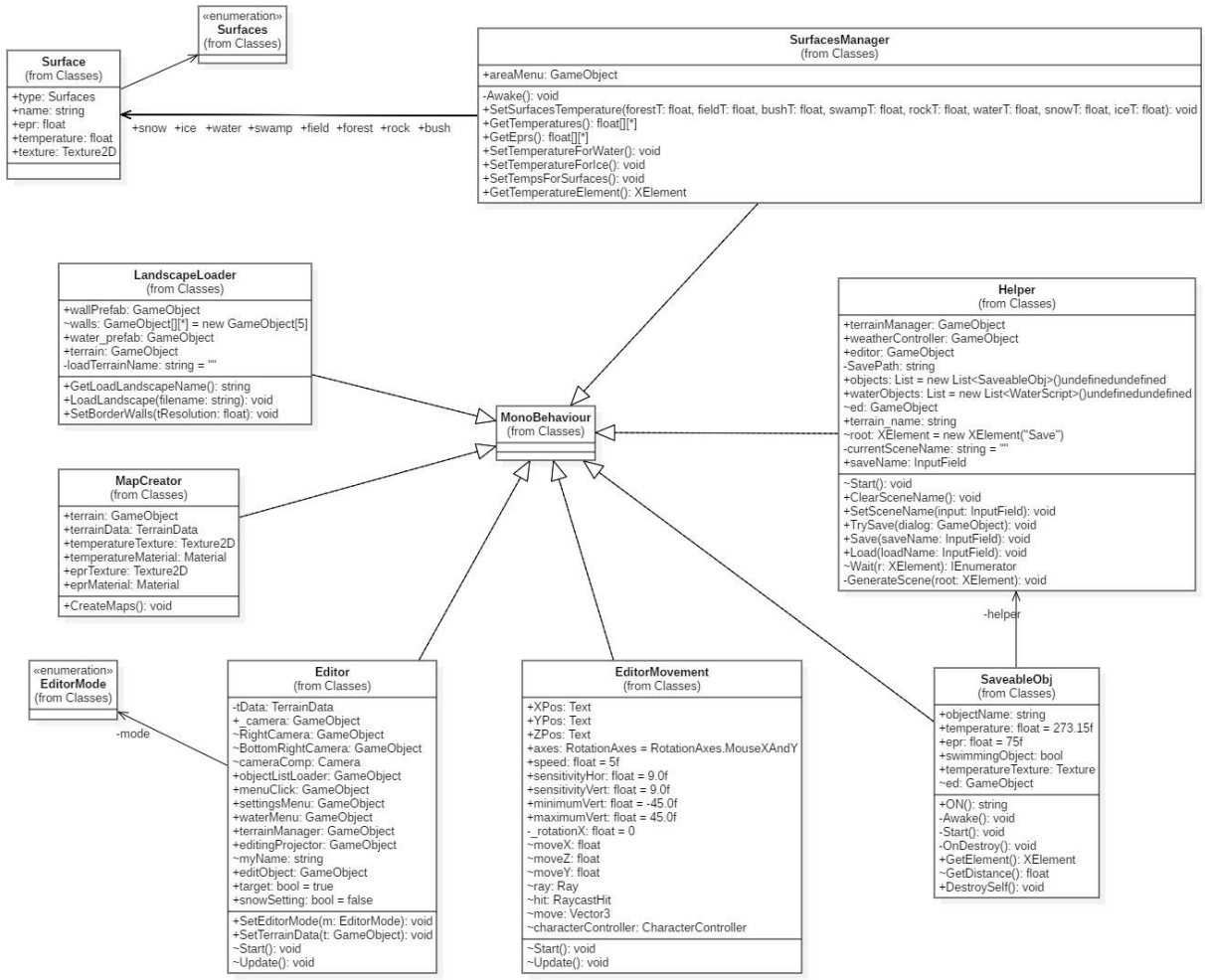


Рисунок 3 – Диаграмма управляющих классов редактора местности

К классам интерфейса блока редактора местности можно отнести классы: loadTerrainList, areaMenuScript, loadObjectList, ObjectSettings, settingMenu, saveScene, loadScene.

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

Поля класса:

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

2) buttonPrefab – объект – элемент списка ландшафтов;

3) creationPosition – позиция отображения превью модели ландшафта;

4) controller – объект, управляющий анимацией превью модели 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.

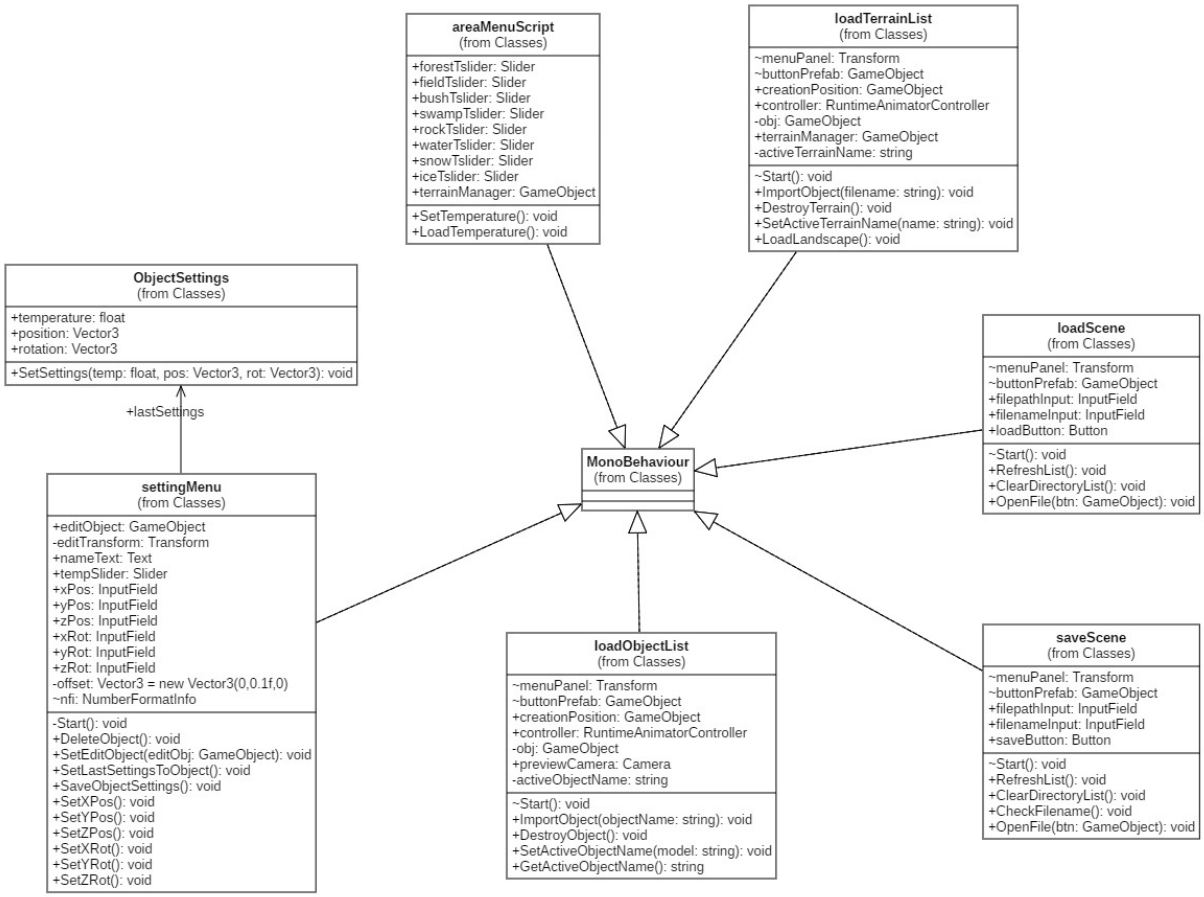


Рисунок 4 – Диаграмма управляющих классов редактора местности

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.

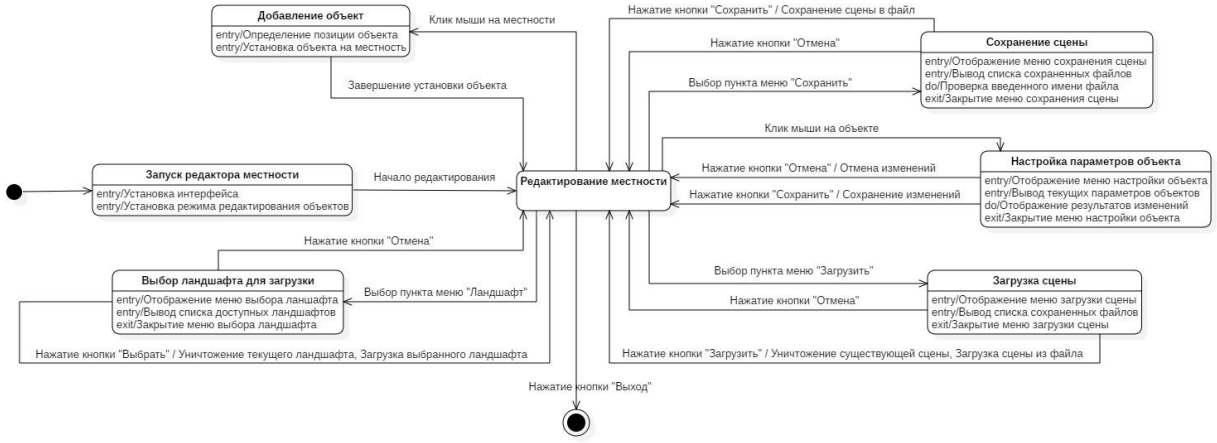


Рисунок 5 – Диаграмма состояний редактора местности

3 ПРОЕКТИРОВАНИЕ АЛГОРИТМОВ БЛОКА РЕДАКТОРА МЕСТНОСТИ

Рассмотрим подробно процесс проектирования следующих модулей редактора местности:

1) модуль загрузки ландшафта;

2) модуль настройки подстилающих поверхностей;

3) модуль настройки объектов.

3.1 Модуль загрузки ландшафта

В данном модуле наибольший интерес при проектировании представляют метод «LoadLandscape» класса «LandscapeLoader» и метод «CreateMaps» класса «MapCreator».

Метод «LoadLandscape» реализует загрузку требуемого ландшафта из набора заготовок на сцену. Алгоритм предусматривает уничтожение уже существующих на сцене ландшафта и его снежного покрова, установку нового ландшафта, выделение его границ и создание новой карты подстилающих поверхностей. Общая схема алгоритма представлена на рисунке 6.

Метод «CreateMaps» создает карты текстур для загруженного ландшафта. Данные карты текстур хранят информацию о значении температуры и эффективной площади рассеивания (ЭПР) в каждой точке ландшафта в зависимости от находящейся в этой точке подстилающей поверхности. Суть алгоритма сводится к получению установленных значений параметров поверхностей, генерации массива данных о типе поверхности в каждой точке ландшафта и созданию текстур, представляющих собой карту значений температур и ЭПР. Общая схема данного алгоритма представлена на рисунке 7.

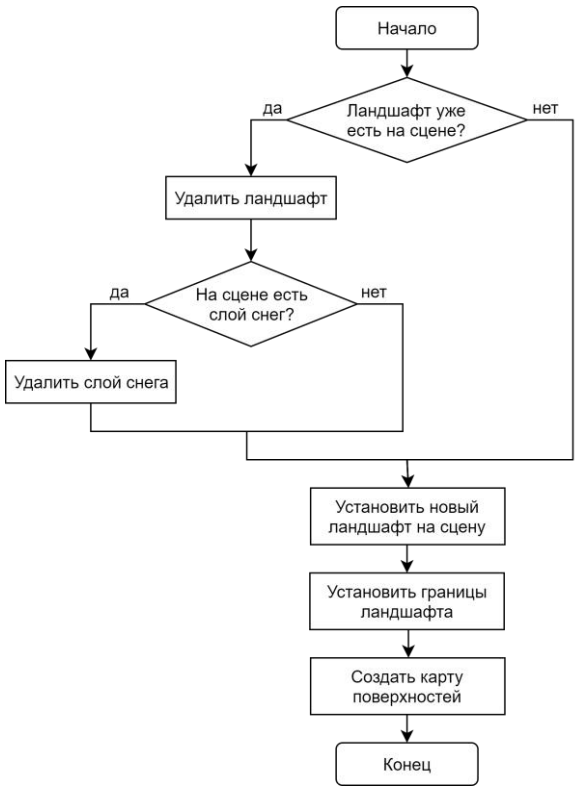


Рисунок 6 – Схема алгоритма загрузки ландшафта

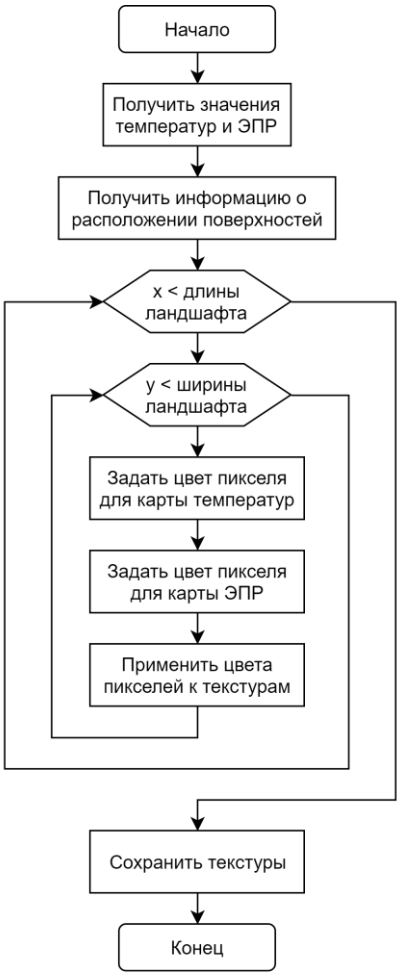


Рисунок 7 – Схема алгоритма создания текстурных карт

3.2 Модуль настройки подстилающих поверхностей

В данном модуле рассмотрим проектирование методов «SetSurfacesTemperature» и «SetTemperatureForWater» класса «SurfacesManager».

Метод «SetSurfacesTemperature» реализует установку значений температур для подстилающих поверхностей. Алгоритм предусматривает присвоение параметров для поверхностей всех типов. Его общая схема представлена на рисунке 8.

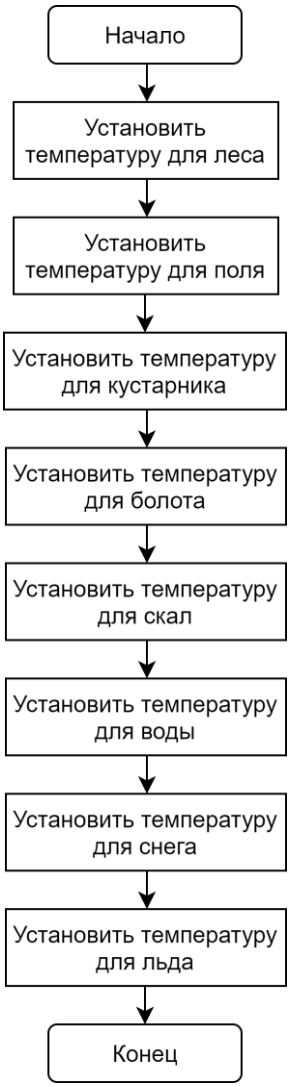


Рисунок 8 – Схема алгоритма установки температур поверхностей

Метод «SetTemperatureForWater» реализует установку значения температуры воды для всех объектов воды на сцене. Алгоритм предусматривает получения списка всех присутствующих ан сцене объектов воды и присвоение им значения температуры водной поверхности. Его общая схема представлена на рисунке 9.

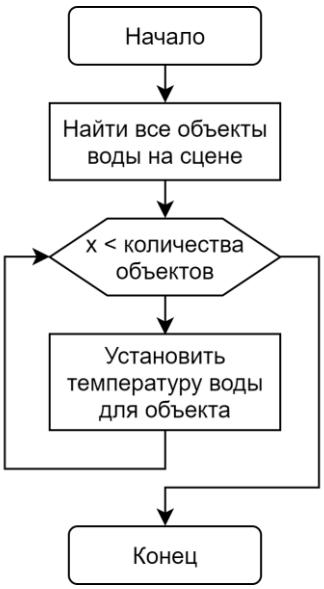


Рисунок 9 – Схема алгоритма установки температуры объектам воды

3.3 Модуль настройки объектов

В данном модуле рассмотрим проектирование методов «SaveObjectSettings», «SetYRot» и «SetPos» класса «settingMenu».

Данные метод отвечают за установку заданных пользователем параметров для редактируемого объекта на сцене. Алгоритм предусматривает присвоение объекту таких параметров как температура, поворот по вертикальной оси Y, координаты в пространстве. Его общая схема представлена на рисунке 10.

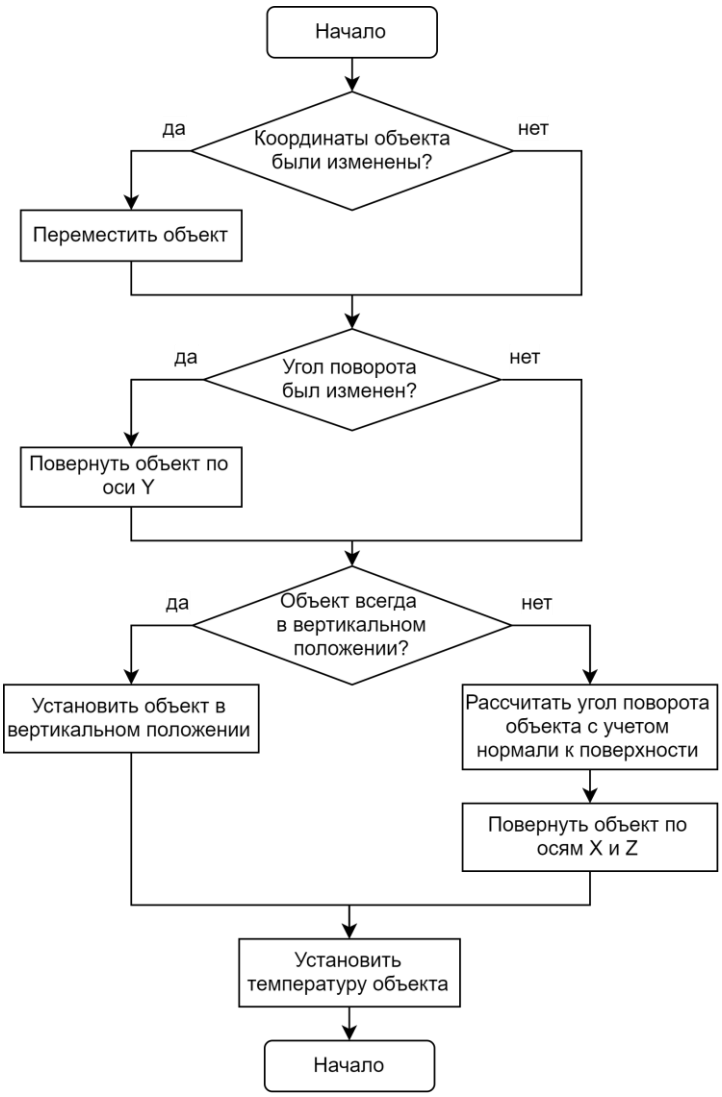


Рисунок 10 – Схема алгоритма установки параметров объекта

4 РЕАЛИЗАЦИЯ АЛГОРИТМОВ БЛОКА РЕДАКТОРА МЕСТНОСТИ

4.1 Реализации алгоритмов класса LandscapeLoader

За реализацию добавления ландшафта на сцену 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.

```
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);
}
```

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.

Полный текст класса приведен в приложении А.

4.2 Реализации алгоритмов класса MapCreator

В вышеописанном методе происходит вызов метода «CreateMaps» класса «MarCreator». Данный класс отвечает за создание текстурной карты поверхностей, используемой в процессе моделирование изображений.

В начале работы метода производится поиск на сцене текущего ландшафта, информация о котором заносится в переменную «terrainData». Затем, с помощью вызова методов класса «SurfacesManage», происходит получение значений характеристик поверхностей, хранящихся в виде массивов.

```
terrain = GameObject.FindWithTag("CreateTerrain");
terrainData = terrain.GetComponent<Terrain>().terrainData;

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

Далее выполняется изменения текущего разрешения текстурных карт для их соответствия карте поверхностей ландшафта, а также создаются переменные для хранения цвета пикселя каждой карты.

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

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

В переменную «alphamapData», представляющую собой трехмерный массив вещественных чисел, заносится информацию о типе поверхности в каждой точке ландшафта. После чего в двух вложенных циклах производится поиск

максимального значения, соответствующего доминирующему типу поверхности в данной точке. Это необходимо для точного определения характеристик на границе нескольких типов поверхностей.

Затем по индексу найденного значения типа поверхности, соответствующему индексу значения температуры и ЭПР данной поверхности в соответствующих массивах, определяется цвет текстурных карт в текущем пикселе и заносится в переменные «tempColor» и «eprColor». Далее найденные цвета присваиваются соответствующим текстурам.

```
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] / 200;
            tempColor.r = tempColor.g = tempColor.b = color;

            color = (eprs[i] + 50) / 200;
            eprColor.r = eprColor.g = eprColor.b = color;
            break;
         }
      }

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

Перед завершением работы метода происходит сохранение изменений в текстурах и сохранение самих текстур в виде файлов – изображений с разрешением «.png».

```
            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();
        }
```

Полный текст класса приведен в приложении А.

4.3 Реализации алгоритмов 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. В качестве параметров метода используются вещественные значения температур поверхностей.

```
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();
        }
    }
```

Метод SetTemperatureForWater() устанавливает значение температуры для всех присутствующих на сцене объектов водной поверхности, а SetTemperatureForIce() – для всех объектов ледяной поверхности. В переменную «waters» с помощью метода «FindObjectsOfType» заносится список объектов класса «WaterScript». После чего в цикле изменяется значение соответствующего поля класса.

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

Полный текст класса приведен в приложении А.

4.4 Реализации алгоритмов класса settingMenu

Данный класс «settingMenu» отвечает за изменение пользователем параметров установленного на сцене редактируемого объекта. Изменению подлежат такие характеристики объекта, как его координаты, угол поворота по оси Y и температура.

При выборе объекта, подлежащего редактированию, ссылка на него заносится в переменную «editObject». Метод «SaveObjectSettings» записывает в поле «temperature» класса «SaveableObj» объекта его новую температуру, значение которой получается из соответствующего элемента интерфейса. Для корректной работы датчиков в процессе моделирования температура объекта хранится в Кельвинах.

После изменения температуры происходит изменение позиции объекта и сохранение полученных настроек в качестве текущих.

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

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

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

Метод «SetYRot» устанавливает поворот объекта с учетом возможности его поворота по осям X и Z. Необходимость объекта всегда находиться в вертикальном положении, т.е. соответствие направления координатной оси Y объекта направления мировой вертикальной оси, определяется его «тэгом». Если «тэг» объекта имеет значение «YOrient», то происходит его размещение в точке ландшафта в вертикальном положении. Если же это не так, происходит расчет вектора нормали к точке ландшафта, а объект поворачивается так, чтобы его координатная ось Y совпала с данным вектором. Точка ландшафта, в которую необходимо установить объект определяется через метод «Raycast», находящим координаты точки, расположенной под центром объекта.

```
public void SetYRot() {
    RaycastHit rayHit;
    Physics.Raycast(editTransform.position + offset, Vector3.down, out
rayHit);
    if (editTransform.tag != "YOrient") {float degree = float.Parse(yRot.text,
CultureInfo.InvariantCulture) - editTransform.rotation.eulerAngles.y;
        editTransform.RotateAround(rayHit.point, rayHit.normal, degree);
    } else {
        editTransform.rotation =
Quaternion.Euler(editTransform.rotation.eulerAngles.x, float.Parse(yRot.text,
CultureInfo.InvariantCulture), editTransform.rotation.eulerAngles.z);
    }
}
```

Полный текст класса приведен в приложении А.

ЗАКЛЮЧЕНИЕ

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

Так как все задачи курсовой работы были выполнены, а требуемое программное обеспечение было реализовано, 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. 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 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 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 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));
        }
    }
```