МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
Руководитель отдела «КСиТ»
ООО НПП «Учтех-Профи»
Домбровский Кирилл Александрович

_____
«___» _____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___» _____ 2019 г.

Интерактивная обучающая 3D программа «Спутниковая система навигации»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____И.Л. Надточий
«___» _____2019 г.

Автор работы,
студент группы КЭ-222
_____Н.С. Уткин
«___» _____2019 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___» _____2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___» _____2019 г.

ЗАДАНИЕ

на выпускную квалификационную работу магистра

студенту группы КЭ-222
Уткину Никите Сергеевичу
обучающемуся по направлению
09.04.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Интерактивная обучающая 3D программа “Спутниковая система навигации”» утверждена приказом по университету от 25 апреля 2019 г. № 899

2. **Срок сдачи студентом законченной работы:** 1 июня 2019 г.

3. **Исходные данные к работе:**

– техническое задание на разработку программного комплекса «Спутниковые системы навигации»;
– НПП «Учтех-профи». Теоретические пособие. Учебно-лабораторный стенд «Системы спутниковой навигации», 2014. – 98 с.

4. **Перечень подлежащих разработке вопросов:**

- анализ существующих учебных стендов и программ, рассматривающих тематику систем спутниковых навигаций;
- проектирование и разработка модели программного комплекса;
- реализация программного комплекса;
- тестирование и отладка разрабатываемого проекта;
- применение защитных средств для лицензирования ПО.

5. **Дата выдачи задания:** 1 декабря 2018 г.

Руководитель работы _____/*И.Л. Надточий*/

Студент _____/*Н.С. Уткин*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2019	
Разработка модели, проектирование	01.04.2019	
Реализация системы	01.05.2019	
Тестирование, отладка	15.05.2019	
Компоновка текста работы и сдача на нормоконтроль	24.05.2019	
Подготовка презентации и доклада	30.05.2019	

Руководитель работы _____/*И.Л. Надточий*/

Студент _____/*Н.С. Уткин*/

Аннотация

Н.С. Уткин. Интерактивная обучающая 3D программа “Спутниковая система навигации”. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 132 с., 74 ил., библиогр. список – 17 наим.

Целью данной работы является разработка интерактивной программы «Спутниковые системы навигации» в среде разработки Unity.

В рамках работы проведен анализ существующих учебных стендов и программ, связанных с изучением тематики систем спутниковых навигаций (ССН). Программных обучающих демонстрационных аналогов подобной темы не найдено в рамках проанализированного российского и зарубежного рынка. Однако найден диорамный макет с информационными плакатами.

Проведено исследование актуальности данной темы в виде опроса студентов, обучающихся на технических специальностях, связанных с радиоэлектроникой; спроектирована и разработана архитектура программного обеспечения; реализовано само программное обеспечение; проведено тестирование и отладка разрабатываемого программного обеспечения; внедрена защита в разработанное готовое программное приложение для последующего внедрения.

В результате работы было создано интерактивное программное приложение с шестью внутренними модулями, соответствующими шести ключевым темам систем спутниковой навигации.

ОГЛАВЛЕНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1. ССН – система спутниковой навигации.
2. ПК – персональный компьютер.
3. ПО – программное обеспечение.
4. AI – Artificial Intelligence (русс. «Искусственный интеллект»).
5. UI – User Interface (русс. «Интерфейс пользователя»).

ВВЕДЕНИЕ

В рамках данной темы *был исследован практический вопрос* доступности и актуальности изучения теоретической темы, связанной с системами спутниковой навигации. На данный момент в высших учебных заведениях страны на специальностях, связанных с изучением и применением радиоэлектроники, на недостаточном уровне раскрываются или вовсе не раскрываются тема систем спутниковой навигации.

В свою очередь специалисты по окончанию обучения могут пользоваться технологиями ССН и неправильно их применять в разрабатываемых программно-аппаратных комплексах, не понимая на достаточном уровне сути работы технологий ССН. Это может привести к различным ошибкам, вызывающих некорректную работу этих комплексов, что в свою очередь сопровождается дополнительными издержками в виде потраченного времени, рабочих и денежных ресурсов, а также заложенных «спящих» ошибок, которые проявятся в ходе эксплуатации комплексов.

Учебники и теоретические пособия дают полную информацию о работе систем спутниковой навигации, но не являются наглядными с точки зрения работы и применения данных систем.

В настоящее время компьютерные сети и телекоммуникации являются неотъемлемой частью при внедрении и эксплуатации как производственной, так и социальной инфраструктуры страны. В связи с этим, *научной проблемой является* обеспечение соответствующего уровня подготовки специалистов инженерных направлений.

Ведущую роль в процессе обучения технических специалистов играют такие виды учебных занятий, которые позволяют теоретическое знание трансформировать в конкретные практические навыки. Для этой цели безусловную важность приобретает материально-техническое оснащение

учебных аудиторий, а именно соответствующее учебно-лабораторное оборудование.

В последнее время одним из самых перспективных направлений в учебном оборудовании являются стенды по частичной, либо полной виртуализации оборудования.

В данной работе рассматривается одна из подобных интерактивных демонстрационных 3D программ.

Таким образом *целью данной работы* является реализация программного комплекса, наглядно демонстрирующего принцип работы и применения систем спутниковой навигации и дающий общие представления о тематике, посвященной системам спутниковой навигации.

Для достижения поставленной цели работа была разложена на *решение следующих задач*:

1. Проанализировать существующие учебные стенды и программы, рассматривающие тематику систем спутниковых навигаций.
2. Провести исследование учебных планов высших учебных заведений с выявлением того, изучается ли тема систем спутниковой навигаций на рассматриваемых специальностях. Также провести исследование об актуальности и необходимости подобного программного комплекса среди студентов, специальности которых связанны с изучением и применением радиоэлектронных средств.
3. Спроектировать и разработать модель программного комплекса.
4. Разработать архитектуру программного комплекса и представить конечную его реализацию с использованием выбранной среды разработки.
5. Провести тестирование и отладку программы.
6. «Прошить» программное обеспечение защитным средством для лицензирования ПО.

Данные задачи будут выполнятся в том порядке, в котором они представлены выше.

Анализ существующих аналогов будет произведен посредством изучения в сети Интернет рынка учебного оборудования, государственных аукционов и основных подрядчиков, занимающихся созданием учебных стендов и учебного программного обеспечения.

Исследование технических специальностей, связанных с изучением и применением радиоэлектронных устройств, будет проведено в рамках ФГАОУ ВО «ЮУрГУ (НИУ)». Будут найдены данные специальности, изучены их учебные планы и составлен сводный перечень радиотехнических специальностей с итоговой информацией о поиске целевой темы.

Исследование актуальности и необходимости данного программного комплекса среди студентов будет проведено в виде электронного опроса. Будут проанализированы результаты и сделаны соответствующие выводы об актуальности.

Проектирование и разработка модели программного комплекса будет произведено посредством изучения предмета теории систем спутниковой навигации, составления логической структуры программы, анализа и выбора средств для разработки программного комплекса.

Разработка программного комплекса будет проведена с помощью выбранных на предыдущем шаге средств разработки.

Тестирование и отладка программа будет проведена в выбранной среде разработке при различных сценариях использования приложения. Тестирование собранного приложения также будет произведено на различных персональных компьютерах под управлением операционной системы Windows.

Лицензирование программного обеспечения будет произведено каким-либо из методов защиты программного обеспечения после проведения соответствующего анализа существующих средств защиты.

Работа разбита на следующие главы:

1. Анализ предметной области и исследование проблемы.
2. Определение требований.
3. Проектирование программного комплекса.
4. Реализация программного комплекса.
5. Тестирование программного комплекса.
6. Заключение.

Основными источниками информации являются:

– НПП «Учтех-профи». Теоретические пособие. Учебно-лабораторный стенд «Системы спутниковой навигации». – 2014 – 98 с.
– Руководство Unity – https://docs.unity3d.com/Manual/index.html
– В.В. Конин. Национальный авиационный университет. Системы спутниковой радионавигации / В.В. Конин, В.П. Харченко. – Киев: Холтех, 2010. – 520 с.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОБЗОР АНАЛОГОВ

На сегодняшний день на российском рынке существует лишь один аналог, демонстрирующий работу спутниковых навигационных систем. Этим аналогом является интерактивный учебно-тренировочный комплекс по навигационной системе ГЛОНАСС для подготовки специалистов ВС РФ ПО «Зарница» [1].

Данная компания занимается разработкой и производством учебного оборудования для специализированных учебных учреждений Министерства обороны РФ и Министерства образование и науки РФ.

В основе данного комплекса находится интерактивный диорамный макет, который представляет из себя презентационный комплекс,

включающий в себя оригинальный рельеф местности со стратегическими объектами (рисунок 1).

Рисунок 1 – Интерактивный диорамный макет

Интерактивный диорамный макет предназначен для теоретического изучения принципов определения текущих координат объекта и ознакомления с областью применения, возможностями и перспективами навигационной системы ГЛОНАСС.

Преимущества макета:

- физическое взаимодействие с макетом;
- голосовое сопровождение «Виртуальный учитель»;
- оригинальный рельеф местности.

Недостатки макета:

- программное обеспечение перед физическим макетом является более универсальным, а именно:
    - позволяет использовать ПО вне рамок одно лишь стенда, а на большинстве современных рабочих компьютерных местах;

- нет физических ограничений, которые позволяли бы модифицировать, расширять и оперативно исправлять программу;
- возможность удобного выпуска обновления и быстрого внедрения в последующие и в уже приобретенные заказчиками стенды;

– актуальность использования в методах обучения именно программного обеспечения;
– более габаритный стенд;
– рассматривает только систему ГЛОНАСС.

На основании проведенного обзора аналога можно провести сравнительный анализ аналога и разрабатываемого проекта (таблица 1).

Таблица 1 – Сравнительный анализ аналога

Продукт / Особенности	Интерактивный диорамный макет	Разрабатываемый программный комплекс
Физическое взаимодействие	✓	×
Аудио сопровождение	✓	✓
Интерактивность	✓	✓
Демонстрация всех ключевых моментов тематики	×	✓
Универсальность использования	×	✓
Улучшения и обновления	×	✓
Рассмотрены две основные ССН	×	✓

На основе данной таблицы можно сделать вывод, что диорамный проект уступает разрабатываемому программному обеспечению по большинству важным моментам.

1.2. ПРОВЕДЕНИЕ ИССЛЕДОВАНИЯ

В рамках данной работы было проведено исследование, состоящее из двух частей:

- исследование технических специальностей;
- исследование актуальности и необходимости данного программного комплекса.

1.2.1. Исследование технических специальностей

Исследование проведено среди технических специальностей ФГАОУ ВО «ЮУрГУ (НИУ)», связанных с изучением и применением радиоэлектронных устройств.

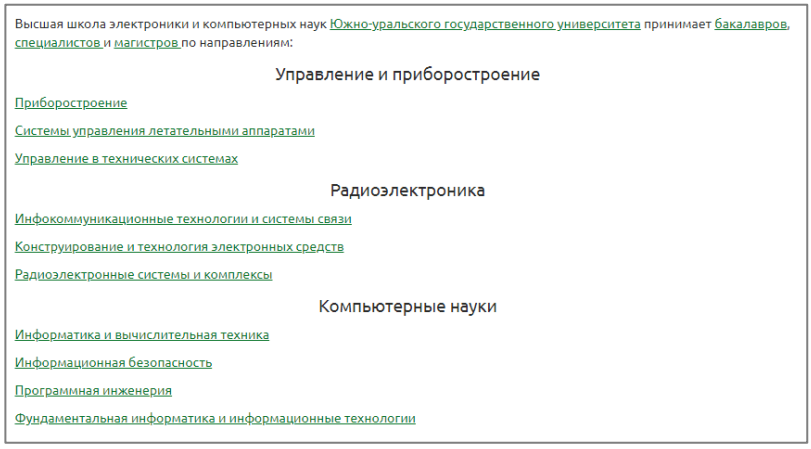
Данное исследование является дополнением ко второму исследованию. Благодаря данному исследованию можно составить перечень специальностей, связанных с изучением радиоэлектроники. Исследуются данные специальности в связи с тем, что студентам на практике приходится иметь дело с модулями приемников спутниковой навигации. Они не уделяют должного внимания при выборе компонентных модулей спутниковой навигации в своих устройствах, что может создать дополнительные трудности в виде ошибок несовместимости или ошибок, вызывающие некорректную работу устройств.

Учебники и теоретические пособия дают полную информацию о работе систем спутниковой навигации, но не являются наглядными с точки зрения работы и применения данных систем.

Все специальности рассматривались на официальном сайте высшей школы электроники и компьютерных наук ФГАОУ ВО «ЮУрГУ (НИУ)» [2], потому что именно в этой школе существуют направления, связанные с радиоэлектроникой.

Были рассмотрены 10 учебных направлений в трех категориях (рисунок 2):

– «Управление и приборостроение»;
    - «Приборостроение»;
    - «Системы управления летательными аппаратами»;
    - «Управление в технических системах»;
– «Радиоэлектроника»;
    - «Инфокоммуникационные технологии и системы связи»;
    - «Конструирование и технология электронных средств»;
    - «Радиоэлектронные системы и комплексы»;
– «Компьютерные науки»;
    - «Информатика и вычислительная техника»;
    - «Информационная безопасность»;
    - «Программная инженерия»;
    - «Фундаментальная информатика и информационные технологии».


Высшая школа электроники и компьютерных наук Южно-уральского государственного университета принимает бакалавров, специалистов и магистров по направлениям:

Управление и приборостроение

Приборостроение

Системы управления летательными аппаратами

Управление в технических системах

Радиоэлектроника

Инфокоммуникационные технологии и системы связи

Конструирование и технология электронных средств

Радиоэлектронные системы и комплексы

Компьютерные науки

Информатика и вычислительная техника

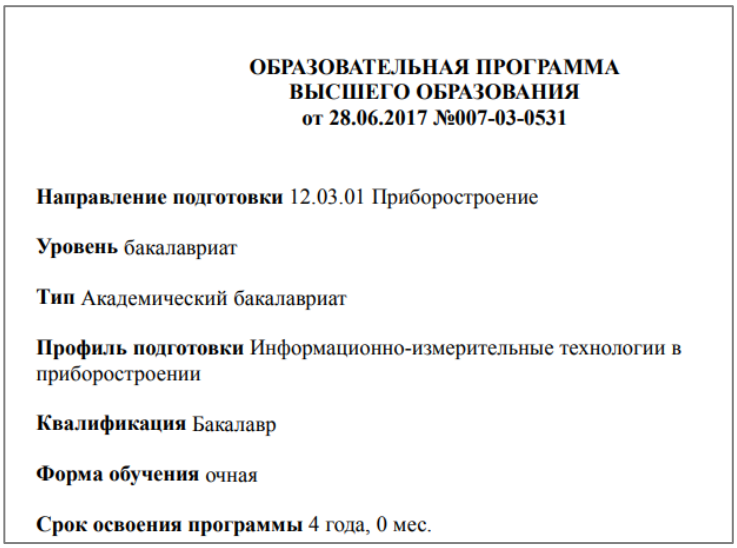
Информационная безопасность

Программная инженерия

Фундаментальная информатика и информационные технологии

Рисунок 2 – Направления подготовки ВШ ЭКН

В каждом из направлений были изучены их образовательные программы и дисциплины (рисунок 3).



ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА
ВЫСШЕГО ОБРАЗОВАНИЯ
от 28.06.2017 №007-03-0531

Направление подготовки 12.03.01 Приборостроение

Уровень бакалавриат

Тип Академический бакалавриат

Профиль подготовки Информационно-измерительные технологии в приборостроении

Квалификация Бакалавр

Форма обучения очная

Срок освоения программы 4 года, 0 мес.

Рисунок 3 – Направление подготовки Приборостроение

В таблице 2 представлены результаты поиска темы, связанной с системами спутниковой навигации.

Таблица 2 – Исследование направлений подготовки ВШ ЭКН

Направление	**Предмет, связанный с темой ССН**
Приборостроение	Нет предмета
Системы управления летательными аппаратами	Пилотажно-навигационные комплексы
Управление в технических системах	Нет предмета
Инфокоммуникационные технологии и системы связи	Нет предмета
Конструирование и технология электронных средств	Нет предмета
Радиоэлектронные системы и комплексы	Основы теории радионавигационных систем и комплексов
Информатика и вычислительная техника	Нет предмета
Информационная безопасность	Нет предмета
Программная инженерия	Нет предмета
Фундаментальная информатика и информационные технологии	Нет предмета

Среди 10 направлений только в двух были найдены дисциплины, которые включают в себя изучение систем спутниковой навигации. По направлению «Системы управления летательными аппаратами» в дисциплине «Пилотажно-навигационные комплексы» уделена небольшая часть лекции изучению данной темы (рисунок 4). По направлению «Радиоэлектронные системы и комплексы» в дисциплине «Основы теории радионавигационных систем и комплексов» вниманию ССН уделена практически все разделы дисциплины (рисунок 5).

7	5	Радиотехнические методы и средства определения навигационных параметров. Свойства и распространение радиоволн. Методы и системы определения координат: дальномерные системы, разностно-дальномерные системы. Угломерные системы (методы: амплитудные, фазовые, амплитудно-фазовые), Угломерно-дальномерные системы. Доплеровские измерители скорости и угла сноса. Радиовысотомеры. Спутниковые навигационные системы.	2

Рисунок 4 – Лекция дисциплины «Пилотажно-навигационные комплексы»

5. Содержание дисциплины

№ раздела	Наименование разделов дисциплины	Объем аудиторных занятий по видам в часах			
		Всего	Л	ПЗ	ЛР
1	Принципы радионавигации	6	6	0	0
2	Методы определения местоположения	8	4	0	4
3	Спутниковые РНС	16	6	0	10
4	Региональные РНУ и РНС	14	4	0	10
5	Комплексные РНС	8	4	0	4
6	Автономные РНУ и РНС	8	4	0	4
7	Навигационные комплексы	4	4	0	0

Рисунок 5 – Содержание дисциплины «Основы теории радионавигационных систем и комплексов»

Исходя из данного исследования можно сделать вывод, что на большинстве радиотехнических направлений не изучается или изучается в недостаточном объеме тема ССН. Лишь одно направление освещает данную тему в достаточном объеме (в размере отдельной дисциплины), потому что направление в целом ориентировано на обучение и изучение радионавигационных систем и устройств.

К тому же теоретический материал в виде учебника или пособия по данной теме хотя и дает в полном объеме изучить ССН, но не является наглядным по сравнению с демонстрационной программой. Программа не

реализуется с целью заменить теоретический материал учебных пособий, методичек и учебников, но является важным дополнением к основному теоретическому материалу, что даст обучающемуся наглядное представление и понимание принципов работы ССН и факторов, влияющих на её работу.

1.2.2. Исследование актуальности и необходимости данного программного комплекса среди студентов

Исследование проведено в виде электронного опроса среди студентов технических специальностей ФГАОУ ВО «ЮУрГУ (НИУ)» ВШ ЭКН, связанных с изучением и применением радиоэлектронных устройств.

Был создан электронный опрос с помощью инструмента «Google Формы» (рисунок 12). Были опрошены студенты и выпускники с направлений, рассмотренных в предыдущем исследовании, или связанных с радиоэлектроникой. В опросе на момент написания данного исследования приняло участие 63 человека.

Опрос состоял из 6 вопросов. В форме ответов были автоматически построены 6 круговых диаграмм с выбранными вариантами ответов (рисунки 6-11).

Рисунок 6 – Первый вопрос

Рисунок 7 – Второй вопрос

Рисунок 8 – Третий вопрос

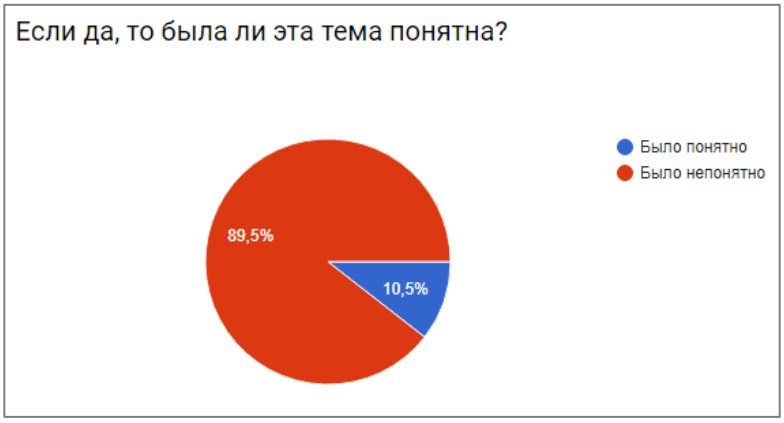


Рисунок 9 – Четвертый вопрос

Рисунок 10 – Пятый вопрос

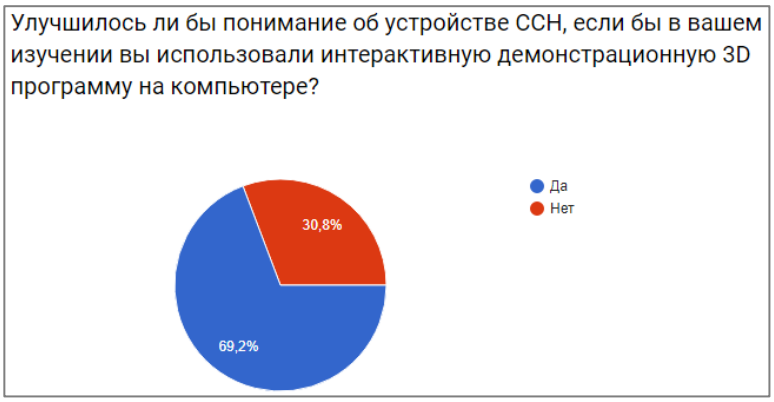


Рисунок 11 – Шестой вопрос

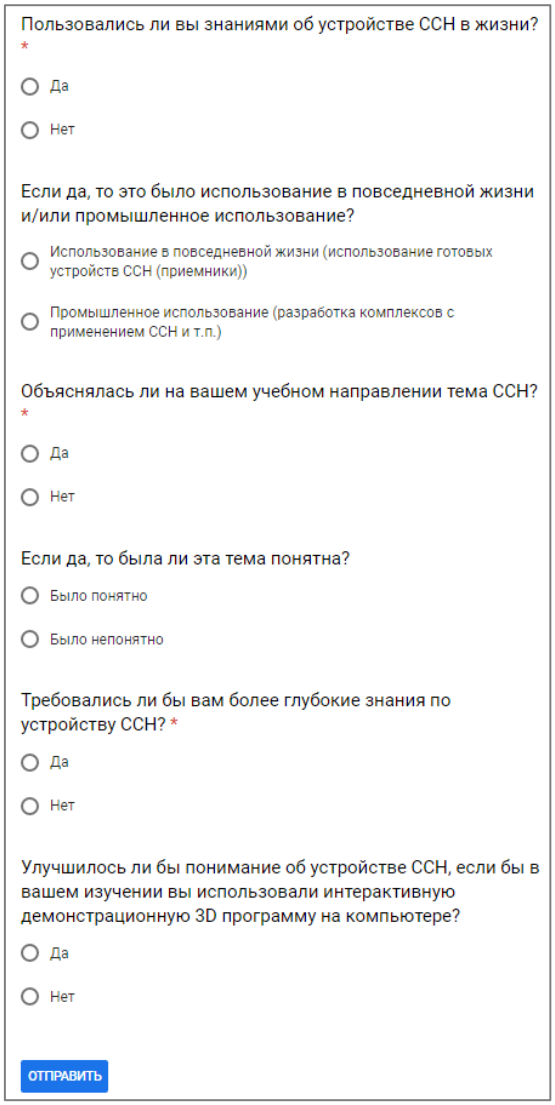


Рисунок 12 – Форма опроса

Из проведенного опроса можно понять, что:

- 85,7% респондентов применяли знания теории ССН в жизни;
- 71,4% из них использовали их только в повседневной жизни;
- 37,5% респондентам затрагивалась или объяснялась в учебных заведениях на дисциплинах тема ССН;
- 10,5% из них была понятна данная тема;
- 31,3% респондентам требовались более глубокие знания по устройству ССН;
- У 69,2% улучшилось бы понимание данной темы, если бы при её изучении они использовали интерактивную демонстрационную 3D программу.

На основании проведенного опроса можно сделать вывод об актуальности создания интерактивной демонстрационной 3D программы, которая позволит в дополнении к основному теоретическому материалу наглядно продемонстрировать принцип работы и устройство систем спутниковой навигации.

1.3. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

1.3.1. Выбор среды разработки

Для реализации описанного программного комплекса необходимо выбрать необходимо выбрать основной инструмент для реализации – программную среду разработки 3D приложений (графический движок).

Наиболее популярными графическими платформами для разработки 3D приложений являются движки Unity, Unreal Engine, CryEngine [3].

1.3.1.1. Unity 3D

Unity 3D [4] является на данный момент одним из наиболее широко используемых движков. В настоящее время 34% из 1000 бесплатных мобильных игр сделаны с помощью Unity. Это платформа для художников, дизайнеров и разработчиков, позволяющая создавать и совместно работать

над потрясающим кинематографическим контентом и последовательностями геймплея, используя инструменты 2D и 3D дизайна, режим мгновенного воспроизведения для быстрого редактирования и мощную систему анимации.

В то время как Unreal может быть более популярным среди ПК и консольных игр, Unity определенно более популярен среди мобильных игр и стала популярным игровым движком для многих разработчиков мобильных игр. Он также находится на переднем крае растущего рынка виртуальной реальности и использовался для создания примерно 90% игр Samsung Gear VR и 53% игр Oculus Rift при запуске.

Также за счет популярности движка у него есть большая база знаний и обучения, большое сообщество и разнообразие Unity пакетов (наборы готовых решений), разрабатываемых другими пользователями. Программирование в Unity происходит на языке C#.

- Разработчик: Unity Technologies.
- Поддерживаемые платформы: Windows, Mac, Linux, iOS, Android, Playstation, Xbox, Windows Phone, Tizen и другие.
- Ценовая политика: бесплатно для личного пользования, платные версии начинаются от 35$ в месяц.

1.3.1.2. Unreal Engine

Движок Unreal [5] предлагает полностью бесплатную версию с мощным движком рендеринга и редактором среды. Он содержит различные интерфейсы прикладного программирования (API) и инструменты, позволяющие создавать виртуальную трехмерную среду, которая очень похожа на реальный мир. Unreal поставляется с частичной документацией, в то время как движки Unity и Source предоставляют полную документацию с подробными примерами. Unreal Engine использует C ++, в то время как Unity использует C# или JavaScript.

- Разработчик: Epic Games.

– Поддерживаемые платформы: Windows, Mac, Linux, iOS, Android, Playstation, Xbox и другие.
– Ценовая политика: бесплатно для использования (5% платежа от валовой выручки после получения первых 3000 долларов за продукт в квартал).

1.3.1.3. CryEngine

CryEngine [15] - инструмент для разработки игр, разработанный компанией Crytek. Он облегчает создание сценариев событий, анимацию и создание 3D-объектов в бесплатном CryEngine SDK. CryEngine предназначен для поддержки платформ и консолей ПК, включая Xbox 360 и PlayStation.

– Разработчик: Crytek.
– Поддерживаемые платформы: Windows, Linux, iOS, Android, Playstation, Xbox и Wii.
– Ценовая политика: Бесплатно. Членство начинается с 50$ в месяц.

1.3.1.3. Сравнение игровых движков

Среди трех рассмотренных движков необходимо выбрать наиболее подходящий для данной работы. Разработка решения для игрового движка, подходящего для всех поставленных целей, является сложной задачей для разработчика. Необходимо определить ключевые принципы проектирования и выбрать из них наиболее важные для поставленной задачи. Пять принципов, которые можно выделить и обозначить аббревиатурой MULER — это модульность (modularity), удобство использования (usability), ресурсы библиотеки (library resources), эффективность (efficiency), эффекты визуализации и качество изображения (rendering effects and visual quality) [7].

1. Модульность

   Игровой движок должен быть реализован через несколько уникальных модулей. Каждый компонент менеджера независим, как отдельный

функциональный блок. Разработчики должны потратить время на реализацию модульной игровой архитектуры, чтобы уменьшить сложность системы и обеспечить надежность. Процесс тестирования и обслуживания становится проще для программиста при необходимости.

2. Удобство использования

   Как определено в [8], удобство использования включает в себя простоту обучения, эффективность использования, запоминаемость, частоту и серьезность ошибок, и субъективную удовлетворенность. Легкость обучения и эффективность использования являются основными аспектами юзабилити.

3. Библиотеки ресурсов

   Системы 3D игровых движков реализованы путем объединения множества библиотек ресурсов. Разработчик выбирает игровой движок на основе имеющихся библиотек ресурсов программирования. Общая библиотека для простого трехмерного игрового движка состоит из 3D-графики, физики, обнаружения столкновений, ввода / вывода, аудио, AI, 3D-графики и сетевой библиотеки. Современные игровые движки могут включать в себя более мощные библиотеки, включая законы физики, обнаружение столкновений и специальные эффекты. Кроме того, ресурсы игрового движка содержат примеры проектов, руководства пользователя и учебные пособия. Эти доступные ресурсы чрезвычайно полезны для разработчика игр.

4. Эффективность

   Эффективность игрового движка означает успешное использование всех входных данных и доступных ресурсов для получения заданного игрового результата. Он включает в себя распределение памяти, использование процессора, процесс рендеринга и другие функции.

5. Эффекты визуализации и качество изображения

Процесс рендеринга производит трехмерную анимированную графику с использованием специальных методов, таких как растеризация, рендеринг на основе изображений, трассировка лучей или любой другой метод. Эти методы ценны для 3D игровых движков.

Составлены результирующие таблицы 3, 4 на основе источника [7] с общей информацией о движках и их оценками по вышеописанным критериям.

Таблица 3 – Сводная информация рассматриваемых движков

Игровой движок	**Язык**	**Поддерживаемые платформы**	**2D**	**3D**	**Лицензия / цена**
Unity	C#	Десктопные: Windows, Mac Os, Linux Мобильные: Android, iOS	Да	Да	Бесплатно для личного пользования
Unreal Engine	C++, VisualScripting	Десктопные: Windows, Mac Os, Linux Мобильные: Android, iOS	Нет	Да	Бесплатно, открытый исходный код
CryEngine	C / C++	Десктопные: Windows, Linux Мобильные: iOS, Android	Нет	Да	Бесплатно

Таблица 4 – Оценка игровых движков на основе модели MULER

Игровой движок	**Модульность**	**Удобство использования**	**Эффективность**	**Библиотеки ресурсов**	**Эффекты визуализации и качество изображения**
Unity	1	5	Приемлемо	Много ресурсов	Приемлемо
Unreal Engine	4	4	Отлично	Много ресурсов	Отлично
CryEngine	3	5	Отлично	Много ресурсов	Отлично

Критерии оценки для каждого принципа определены следующим образом:

1. Модульность: от «0» до «3» для обозначения уровня модульности. «0» означает, что модульность не учитывается в архитектуре игрового движка.
2. Удобство использования: от «1» до «5» для обозначения уровня удобства использования. «5» означает полное удовлетворение пользователя. Оценка присваивается на основе отзывов пользователей и ответов сообщества форума.
3. Ресурсы библиотеки: ограниченные ресурсы, приемлемые ресурсы, очень богатые ресурсы.
4. Эффективность: плохо, хорошо, отлично.
5. Эффекты рендеринга и качество изображения: плохое, хорошее, отличное.

В разрабатываемом проекте важными критериями из представленных является удобство использования и библиотеки ресурсов, так как эти критерии позволяют упростить реализацию программы и позволяют быстро произвести вхождение в устройство работы выбранной среды разработки.

На основании этих критериев был *выбран Unity,* как наиболее подходящий движок для разработки данного программного комплекса, потому что он:

- обладает удобством и простотой в разработке;
- имеет удобный визуальный редактор;
- использует популярный и простой в использовании языке C#;
- имеет большую базу знаний и обучения;
- имеет большое сообщество;
- имеет большое разнообразие библиотек ресурсов.

1.4. ВЫВОД

Данный программный комплекс является основной частью разрабатываемого заказчиком демонстрационного учебного стенда, который предназначен для изучения принципов работы систем спутниковой навигации ГЛОНАСС / GPS и определения местоположения с их помощью, а также их практического применения.

Таким образом, учебный стенд позволяет специалистам, которые при создании различных программно-аппаратных комплексов, не углубляясь в большой объем теоретической информации (если это не является необходимым), наглядно изучить принцип и устройство работы систем спутниковой навигации, что позволит в свою очередь правильно применить данную технологию в своей работе.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

В данном разделе приведены требования, представленные заказчиком проекта в виде технического задания.

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Демонстрационный комплекс «Системы спутниковой навигации ГЛОНАСС-GPS» должен позволять изучить принципы работы систем спутниковой навигации ГЛОНАСС / GPS и принципы определения местоположения с их помощью, а также практическое применение ССН.

В программной части демонстрационного комплекса «Системы спутниковой навигации ГЛОНАСС-GPS» должно быть представлено не менее трех интерактивных модулей:

- История возникновения и развития ССН
- Принципы определения местоположения объекта
- Факторы, влияющие на точность определения координат

Также должно быть представлено не менее трех модулей с аудиовизуальными клипами:

- Структура и отличия систем спутниковой навигации ГЛОНАСС и GPS
- Показатели точности определения координат
- Практическое применение ССН ГЛОНАСС и GPS

Модуль «История возникновения и развития ССН» должен включать в себя описание возникновения ССН, этапы их развития вплоть до текущего времени и представлять собой интерактивную модель с возможностью выбора исследуемого периода.

Модуль «Принципы определения местоположения объекта» должен представлять собой интерактивную модель с возможностью выбора объекта для определения местоположения с описанием основных этапов определения местоположения.

Модуль «Факторы, влияющие на точность определения координат» должен представлять собой интерактивную модель с возможностью выбора различных условий приема сигнала от спутников с описанием их влияния на определение местоположение.

Модуль «Структура и отличия ССН ГЛОНАСС и GPS» должен включать в себя описание структуры навигационных систем, различий между навигационными системами, визуальное представление используемых группировок спутников, а также преимущества и недостатки каждой из систем в виде аудиовизуального клипа.

Модуль «Показатели точности определения координат» должен включать в себя описание характеристик точности определения местоположения, а также примеры их вычисления в виде аудиовизуального клипа.

Модуль «практическое применение ССН ГЛОНАСС и GPS» должен включать в себя описание сценариев применения ССН в реальной жизни в виде аудиовизуального клипа.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

После реализации и тестирования программного обеспечения должен быть предпринят комплекс мер для обеспечения защиты данного ПО от несанкционированного приобретения, использования, распространения, модифицирования, изучения и воссоздания аналогов. Комплексом мер должно быть применение электронного ключа фирмы Guardant.

Минимальные системные требования, при которых программное обеспечение должно стабильно работать (без внезапных отказов, сбоев, остановок и ошибок) и выдавать среднюю кадровую частоту в размере 60 FPS:

- операционная система: Windows 7 и выше;
- центральный процессор Intel Core I3;
- оперативная память 4 Гб;
- видеокарта Radeon R7 240 2 Гб;
- занимаемая память на диске 2 Гб.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Сцены в Unity содержат объекты проекта. Они могут использоваться для создания главного меню, отдельных уровней и для других целей. Можно считать каждый файл сцены отдельным игровым уровнем. В каждой сцене можно разместить объекты окружения, заграждения, декорации, по кусочкам создавая дизайн и саму игру [9].

На рисунке 13 представлена новая созданная пустая сцена, которая по умолчанию включает в себя такие 3D объекты, как камера («Main Camera») и направленный свет («Directional Light»).

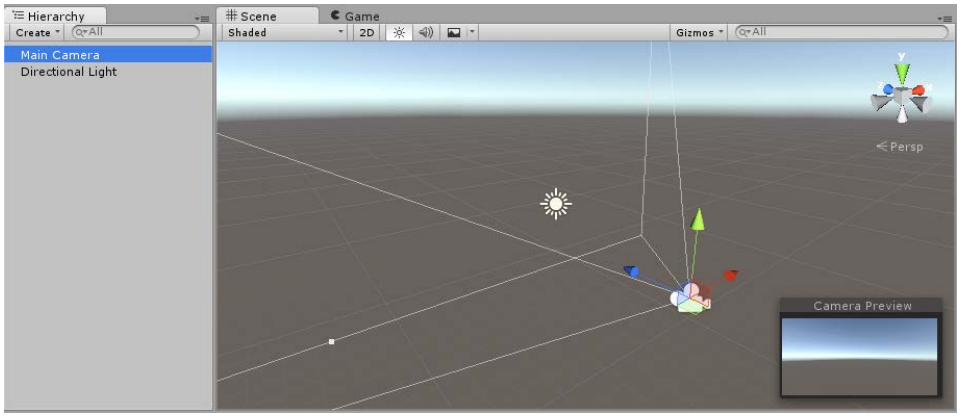


Рисунок 13 – Новая пустая сцена

Весь проект Unity приложения можно представить в виде структурного неориентированного графа сцен (рисунок 14). При запуске программы воспроизводится стартовая сцена «Главное меню». Из неё можно попасть в 6 ключевых сцен, соответствующих 6 модулям (темам) технического задания, а именно:

- «История ССН»
- «Структура и отличия ССН»
- «Принципы определения местоположения»
- «Факторы, влияющие на точность определения координат»
- «Показатели точности»
- «Практическое применение ССН»

В свою очередь, сцена «Факторы, влияющие на точность определения координат» разделена на 5 сцен, соответствующих категориям этих факторов, а именно:

- «Дожди»
- «Снегопады»
- «Атмосфера»
- «Спутники»
- «Город»

Рисунок 14 – Структура сцен проекта, представленная в виде графа

Граф является неориентированным, потому что с точки зрения пользователя по сценам можно перемещаться в обе стороны.

3.2. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ

В данном пункте описывается каждая сцена по отдельности в порядке иерархии сцен в проекте. В частности, описывается: из каких объектов и компонентов она состоит; какие скрипты в ней используются; с помощью чего происходит взаимодействие с пользователем; использование математических моделей (если они присутствуют), воспроизведение работы объектов.

Для графического наполнения сцены использовались 3D модели, предоставленные заказчиком, и 3D модели, распространяемые в свободном доступе. Импортирование моделей в проект происходило в формате «.fbx», который обеспечивает совместимость различных программ трехмерной графики.

Основные понятия, встречающиеся в сценах Unity и описываемые в данном пункте.

1. Игровые объекты (англ. «Game Object», также «объекты» или «объекты сцены») – это фундаментальные объекты в Unity, которые представляют собой, например, персонажей, динамические

интерактивные объекты и декорации [10]. Сами по себе они не представляют какой-то функционал, но действуют как контейнеры для компонентов, которые и реализуют функциональность этих объектов (рисунок 15).

2. Компоненты – это основные элементы поведения в приложении. Они являются функциональными частями каждого игрового объекта (рисунок 15) [11].
3. Скрипты – краткое описание действий, выполняемые системой, которое позволяет создавать свои собственные компоненты, запускать игровые события, изменять свойства компонентов с течением времени и реагировать на ввод пользователя любым способом [12]. Unity поддерживает язык программирования C#.
4. Mesh (русс. «Сетка», также «меш») – основной графический примитив Unity. Сетки составляют большую часть трехмерных миров. Unity поддерживает треугольные или четырехугольные полигональные сетки [12].
5. Shader (русс. «Шейдер») – небольшой скрипт, который содержит математические вычисления и алгоритмы для расчета цвета каждого визуализированного пикселя, основываясь на входном освещении и конфигурации материала [12].
6. Система частиц (англ. «Particles system») – компонент, который имитирует текучие объекты, такие как жидкости, облака и пламя, создавая и анимируя большое количество маленьких 2D-изображений в сцене [12].
7. Collider (русс. «Коллайдер») – невидимая фигура, которая используется для обработки физических столкновений для объекта. Коллайдер не обязательно должен быть точно такой же формы, что и сетка объекта – грубое приближение зачастую более эффективно и неразличимо в игровом процессе [12].

8. Terrain (русс. «Местность») – ландшафт в сцене. Terrain GameObject добавляет большую плоскость на сцену. Можно использовать окно инспектора Terrain для создания детального ландшафта [12].
9. Skybox (русс. «Скайбокс») – специальный тип материала, используемый для представления неба. Обычно шестигранный [12].

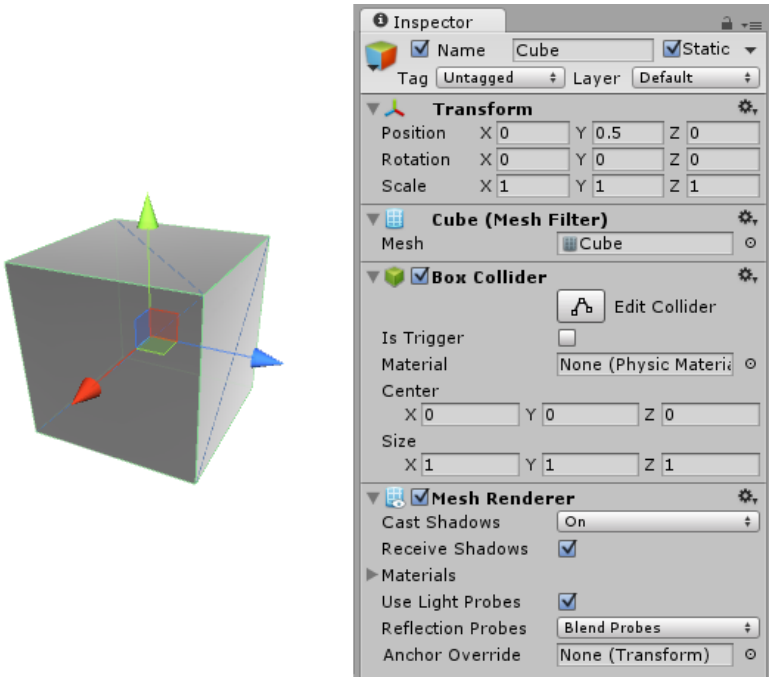


Рисунок 15 – Простой игровой объект «Куб» с различными компонентами

компонентами

Если на разных сценах встречаются одинаковые объекты, компоненты или скрипты, имеющие одинаковую логику работы, то они будут единожды описаны в приведенных ниже пунктах, а при дальнейшем их упоминании по тексту в данном разделе будут ссылаться на свое первичное описание.

Сцены будут рассматриваться в порядке уровней вложенности в проекте.

3.2.1. Сцена «Главное меню» – 1 уровень

Данная сцена является входной точкой программы. Сцена сразу же воспроизводится при запуске программы. Она представляет из себя навигационное меню, позволяющее перейти в соответствующие сцены 2-го уровня, являющиеся тематическими разделами теории ССН.

1. *Объекты сцены.*

В данной сцене используются следующие ключевые объекты (рисунок 16):

– спутник GPS;
– спутник ГЛОНАСС;
– орбита GPS;
– орбита ГЛОНАСС;
– камера;
– планета Земля;
– холст (англ. «Canvas») с объектами UI;
– scripts;
– audioSetter;

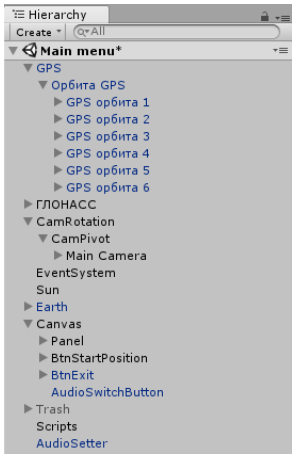


Рисунок 16 – Список объектов на сцене «Главное меню»

Объекты спутников («GPS» и «ГЛОНАСС») сгруппированы по своим орбитам. Это необходимо для единовременной синхронизированной работы спутников между собой в каждой орбите в одной локальной системе координат, что позволяет перемещать спутники в одной группе относительно общей центральной точки родительского объекта.

Камера («Main Camera») является дочерним объектом «CamPivot». Такая группировка необходима для правильной работы скрипта, отвечающего за перемещение камеры вокруг точки опоры («Pivot»).

«EventSystem» является служебным объектом, которые создается автоматически при добавлении объекта «Canvas» на сцену. В данном случае необходим для взаимодействия с устройствами ввода.

«Sun» является направленным источником света в данной сцене и представляет работу Солнца.

«Earth» в основе представляет из себя сферу. На него наложен компонент «Mesh Renderer», отвечающий за наложение материала поверхности Земли, и сам mesh в компоненте «Mesh Filter», предающий геометрию объекту. Важным компонентом является скрипт «Planet», отвечающий за правильную визуализацию текстур, за цвета и за движение атмосферы Земли. За вращение Земли отвечает простой скрипт «Rotate_the_object».

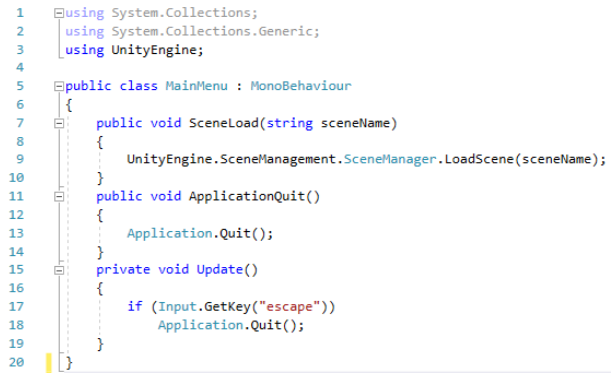
«Canvas» является главным объектом для реализации UI (см. пункт №3).

«Scripts» является обычным контейнером для сбора скриптов, которые не имеют логической или функциональной привязки к какому-либо объекту на сцене. Реализованный скрипт будет работать, если его вынести на сцену в виде компонента объекта.

«AudioSetter» является скриптом, позволяющим работать со звуком на сцене.

2. Скрипты

Скрипт *MainMenu* в данной сцене реализует возможность загружать другую сцену из текущей проигрываемой сцены (функция *SceneLoad()* и возможность выйти из приложения, таким образом позволяя переключаться по сценам второго уровня вложенности в приложении, то есть по тематическим модулям (п. 3.1), и позволяя выйти из приложения (рисунок 17).



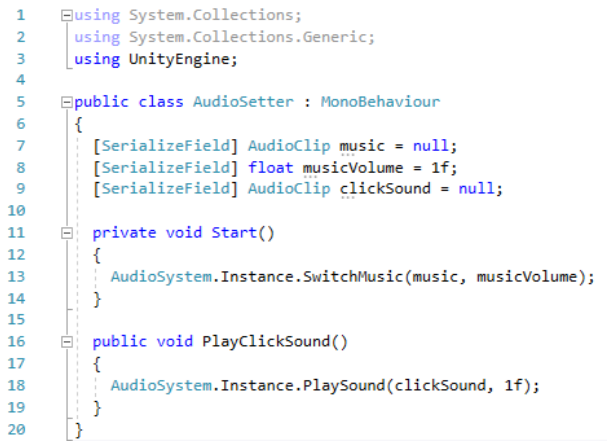
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MainMenu : MonoBehaviour
{
    public void SceneLoad(string sceneName)
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene(sceneName);
    }
    public void ApplicationQuit()
    {
        Application.Quit();
    }
    private void Update()
    {
        if (Input.GetKey("escape"))
            Application.Quit();
    }
}
```

Рисунок 17 – Скрипт "MainMenu"

Скрипт *AudioSetter* позволяет работать со звуком, а именно задавать звуковое сопровождение в сцене. Функция *Start()* является служебной функцией в Unity и вызывается при старте работы сцены, в которой используется данный скрипт. В данном случае при старте сцены переключается музыка и выставляется уровень громкости в соответствии с заданными значениями этих полей. Функция *PlayClickSound()* воспроизводит звук клика при нажатии на UI кнопки (рисунок 18).

Члены класса *music*, *musicVolume* и *clickSound* автоматически появляются в компоненте данного скрипта в виде полей, в которые можно удобно присваивать различные переменные, объекты или компоненты.



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

public class AudioSetter : MonoBehaviour
{
    [SerializeField] AudioClip music = null;
    [SerializeField] float musicVolume = 1f;
    [SerializeField] AudioClip clickSound = null;

    private void Start()
    {
        AudioSystem.Instance.SwitchMusic(music, musicVolume);
    }

    public void PlayClickSound()
    {
        AudioSystem.Instance.PlaySound(clickSound, 1f);
    }
}
```

Рисунок 18 – Скрипт "AudioSetter"

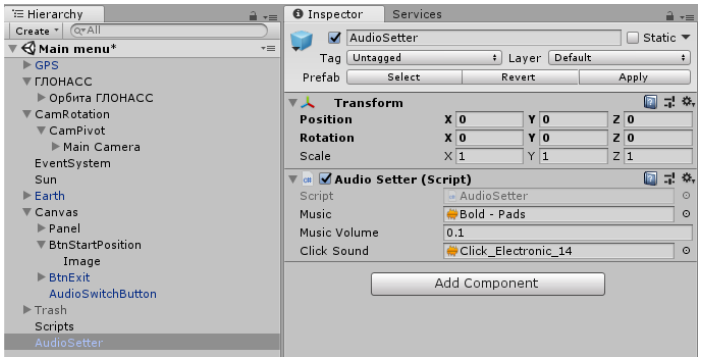


Рисунок 19 – Скрипт "AudioSetter" как компонент объекта

Скрипт *Rotate_the_object* вращает объект с течением времени с заданными параметрами скорости вращения (рисунок 20).

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

public class Rotate_the_object : MonoBehaviour {

    public float Rotation_X=0.0f;
    public float Rotation_Y=20.0f;
    public float Rotation_Z=0.0f;


    // Update is called once per frame
    void Update () {
     transform.Rotate(new Vector3(Rotation_X,Rotation_Y,Rotation_Z) * Time.deltaTime);
    }
}
```

Рисунок 20 – Скрипт "Rotate_the_object"

Скрипт *MainMenuCamControl* обрабатывает параметры нажатия и направления мыши и дает возможность вращать камеру вокруг точки опоры (рисунок 21).

```
void Update ()
{
  Vector2 speed = Vector2.zero;
  if (Input.GetMouseButton(1)) speed = new Vector2(Input.GetAxis(mouseDX) / Time.unscaledDeltaTime, Input.GetAxis(mouseDY) / Time.unscaledDeltaTime);

  smoothedSpeed = Vector2.Lerp(smoothedSpeed, speed, speed.sqrMagnitude > smoothedSpeed.sqrMagnitude ? speedSmoothingCoefGrowth : speedSmoothingCoefFade);
  camRotation.y += smoothedSpeed.x * rotationSpeed * Time.unscaledDeltaTime;
  camRotation.x -= smoothedSpeed.y * rotationSpeed * Time.unscaledDeltaTime;
  if (camRotation.y > 180f)
    camRotation.y -= 360f;
  if (camRotation.y < -180f)
    camRotation.y += 360f;
  camRotation.x = Mathf.Clamp(camRotation.x, -90f, 90f);
  camPivot.localRotation = Quaternion.Euler(camRotation);

  if (Input.GetAxis(Scroll) != 0f)
  {
    camDist = Mathf.Clamp(camDist - Input.GetAxis(Scroll) * radiusChangeSpeed * Time.unscaledDeltaTime, minRadius, maxRadius);
    cam.localPosition = new Vector3(0, 0, -camDist);
  }
}
```

Рисунок 21 – скрипт "MainMenuCamControl"

3. *Взаимодействие с пользователем*

Во всех сценах взаимодействие пользователя происходит через UI элементы. Для переключения на другие сцены используются кнопки, расположенные в верхней части Canvas (рисунок 22). У каждой кнопки есть

событие *OnClick()*, которое обрабатывает нажатие на кнопку. В данном случае происходит вызов функции *SceneLoad()* со входным параметром *string sceneName* из скрипта *MainMenu*, где *sceneName* соответствует названию загружаемой сцены. Также происходит вызов функции *PlayClickSound()* из скрипта *MainMenu* (рисунок 23). Аналогичным образом обрабатываются нажатия на другие кнопки в данной панели с привязанными соответствующими названиями сцен.

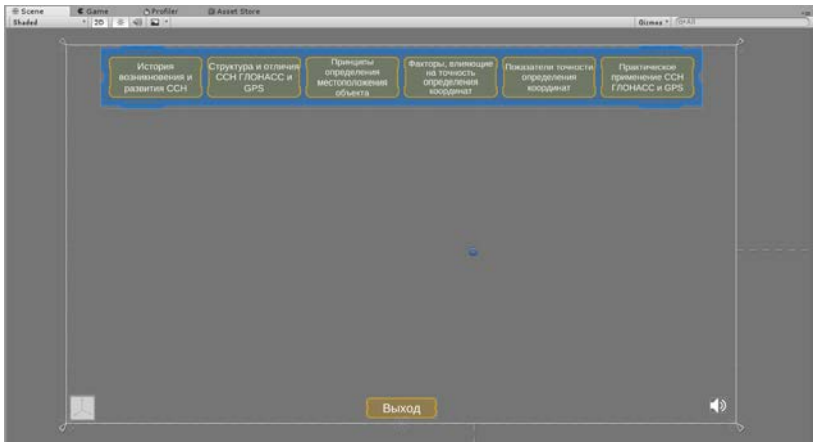


Рисунок 22 – UI элементы на Canvas сцены «Главное меню»

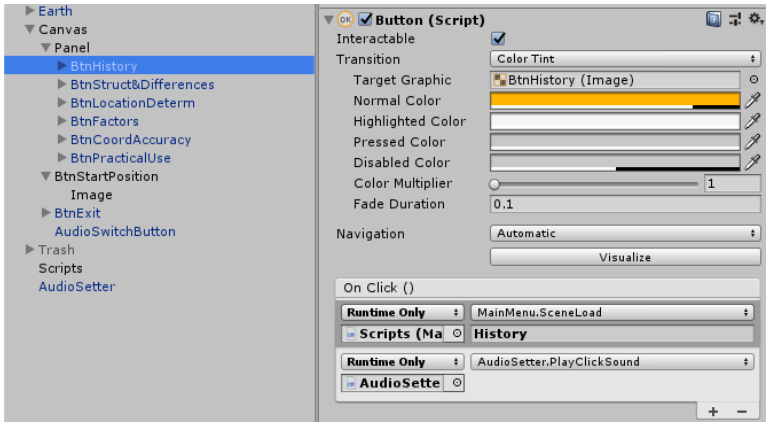


Рисунок 23 – Обработчик события OnClick() на кнопке

4. *Работа объектов*

На данной сцене реализовано вращение сгруппированных спутников по орбитам с общими центральными точками, вокруг которых и происходит вращение. Также вращается Земля вокруг своей оси. Вращение реализуется с помощью скрипта *Rotate_the_object* с заданными параметрами скорости вращения.

Камера вращается вокруг Земли с помощью мыши. Данная функция реализована в скрипте *MainMenuCamControl.*

С помощью кнопок можно загрузить другие сцены, отключить / включить музыку, сбросить положение камеры и выйти из приложения.

3.2.2. Сцена «История ССН» – 2 уровень

На данную сцену можно попасть из сцены «Главное меню». Она представляет из себя интерактивный модуль, посвященный истории и развитию ССН. В данном модуле представлена преимущественно стилизованная текстовая информация с возможностью выбора периодов развития ССН с помощью переключателей.

1. *Объекты сцены.*

В данной сцене большинство 3D объектов повторяется из предыдущей сцены. Представлены полностью другие UI объекты. Полный перечень ключевых объектов (рисунок 24):

- камера;
- спутник GPS;
- спутник ГЛОНАСС;
- орбита GPS;
- орбита ГЛОНАСС;
- планета Земля;
- холст (англ. «Canvas») с объектами UI;
- scripts;
- audioSetter;

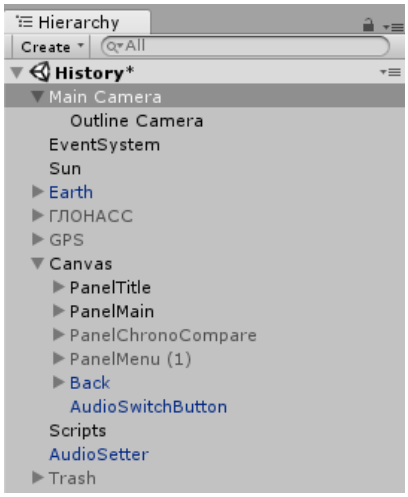


Рисунок 24 – Список объектов на сцене "История ССН"

Все вышеперечисленные объекты имеют такую же функциональную логику работы, как и на предыдущей сцене. Однако теперь камера настроена в ортографической проекции; используется по одной орбите GPS и ГЛОНАСС; спутники не вращаются вокруг Земли; Canvas полностью изменен в соответствии с модулем; Scripts имеет другой набор скриптов

2. *Скрипты*

Главным скриптом на сцене является *History*, который отвечает за переключение UI и 3D объектов при нажатии соответствующих кнопок (рисунок 25). В нем реализованы следующие ключевые методы:

– отслеживание коллизии курсора на объектах и контурное подсвечивание этих объектов – *OnMouseOver(), OnMouseExit(), OnMouseUp(), SetOutlines()*;
– переключение основных UI окон при нажатиях на кнопки – *PanelMain()*;
– переключение визуальных 3D объектов спутников и орбит при вызове соответствующего окна – *EnableChronoOrbit(), ClearOrbits();*
– выход в главное меню – *MainMenu()*.

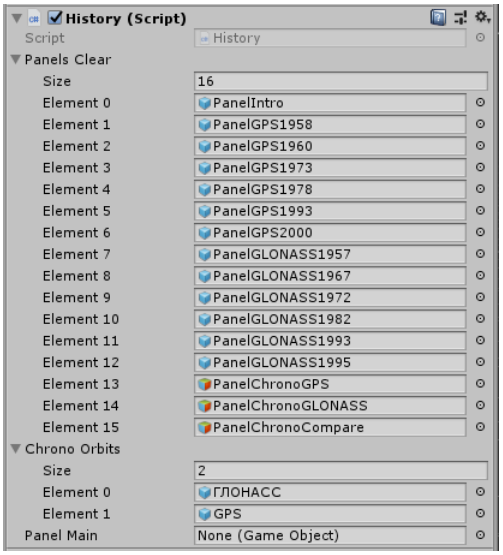


Рисунок 25 – Скрипт "History" как компонент объекта

Rotate_the_object вращает Землю на фоне сцены.

3. *Взаимодействие с пользователем*

Принцип взаимодействия с пользователем происходит аналогичным образом, как и на сцене «Главное меню». С помощью верхних кнопок происходит переключение между 3 подмодулями: история GPS, история ГЛОНАСС, сравнение периодов на временной шкале (рисунок 26).

Два первых подмодуля представляют из себя текстовую информацию о выбранном периоде и изображение разработок спутников в этот период. Периоды выбираются с помощью нижних кнопок или при нажатии на спутники под кнопками.

В третьем подмодуле на панелях представлены кнопки с годами разработок, при наведении на которые появляется всплывающее окно с ключевой информацией о разработках в этом году соответствующей системы (рисунок 27).

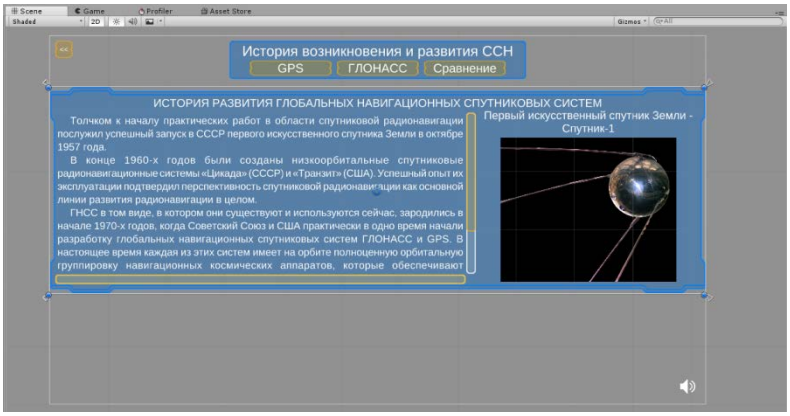


Рисунок 26 – UI элементы на Canvas сцены «История ССН»

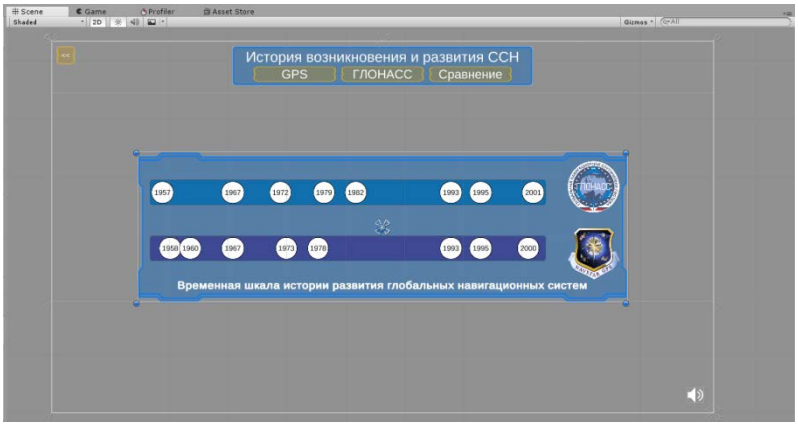


Рисунок 27 – 3-й подмодуль "Сравнение"

3.2.3. Сцена «Структура и отличия ССН» – 2 уровень

На данную сцену можно попасть из сцены «Главное меню». Она представляет из себя интерактивный модуль, посвященный структуре ССН GPS и ГЛОНАСС, их отличию и преимуществам и недостаткам данных систем. Демонстрируется взаимное орбитальное расположение спутников, их вращение вокруг Земли, сами орбиты и спутники, составляющие сигнала спутника.

1. *Объекты сцены.*

В данной сцене представлены следующие ключевые объекты (рисунок 28):

– камера;
– спутник GPS;
– спутник ГЛОНАСС;

– орбита GPS (с визуализацией);
– орбита ГЛОНАСС (с визуализацией);
– наземная станция;
– главная станция управления;
– наземная антенна;
– планета Земля;
– холст (англ. «Canvas») с объектами UI;
– scripts;
– audioSetter;
– визуализированное «облако» сигнала;
– направленный сигнал в виде потокового луча.

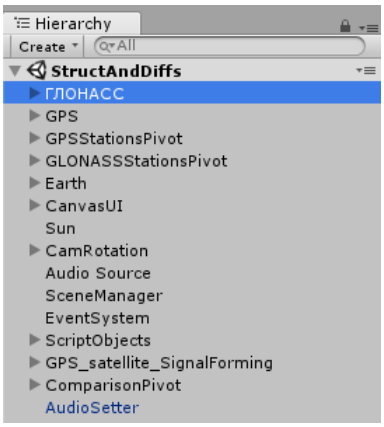


Рисунок 28 – Список объектов на сцене "Структура и отличия ССН"

Спутники вращаются аналогичным образом, как и на сцене «Главное меню», однако их *орбиты* теперь имеют визуализацию в виде шейдеров.

Камера меняет свою позицию и угол вращения в соответствии с выбранным подмодулем и текущим действием.

Наземные объекты (наземные станции, главная станция управления, наземные антенны) располагаются на поверхности Земли, вращаются в системе координат Земли и подсвечиваются контурными линиями (Outline). Появляются и исчезают при соответствующих действиях на сцене.

«Облака» сигналов появляются и меняют в своей цвет в соответствии со сценарием.

Направленный сигнал в виде потокового луча реализован с помощью системы частиц Unity.

2. *Скрипты*

SatellitesRotation синхронизировано вращает спутники относительно друг друга с равным расстоянием и с исключением возможности пересечения друг с другом.

Угол вращения спутника вычисляется по формуле:

, где – время между текущим и предыдущем кадром, – скорость вращения спутника, – последний рассчитанный угол вращения спутника.

В скрипте это выглядит следующим образом:

```
satRotationAngle += Time.deltaTime * satellitesRotationSpeed;
```

Если угол вращения принимает значение больше 360 градусов, то его значение сбрасывается до нуля. Таким образом соблюдается изменение угла в пределах [0;360] градусов:

```
if (satRotationAngle > 360f) satRotationAngle -= 360f;
```

В функциях *UpdateGlonnassSatPos() и UpdateGpsSatPos()* реализовано изменение вращения спутников относительно центра Земли (рисунок 29).

```
40  void UpdateGlonassSatPos()
41  {
42      for (int sat = 0; sat < 8; sat++)
43      {
44          Quaternion localSatRotation = Quaternion.Euler(0f, satRotationAngle + sat * 45f, 0f);
45          glonassSatellites[sat + 0].localRotation = localSatRotation;
46          glonassSatellites[sat + 8].localRotation = localSatRotation;
47          glonassSatellites[sat + 16].localRotation = localSatRotation;
48      }
49  }
50
51  void UpdateGpsSatPos()
52  {
53      for (int sat = 0; sat < 4; sat++)
54      {
55          Quaternion localSatRotation = Quaternion.Euler(0f, satRotationAngle + sat * 90f, 0f);
56          gpsSatellites[sat + 0].localRotation = localSatRotation;
57          gpsSatellites[sat + 4].localRotation = localSatRotation;
58          gpsSatellites[sat + 8].localRotation = localSatRotation;
59          gpsSatellites[sat + 12].localRotation = localSatRotation;
60          gpsSatellites[sat + 16].localRotation = localSatRotation;
61          gpsSatellites[sat + 20].localRotation = localSatRotation;
62      }
63  }
```

Рисунок 29 – Функции UpdateGlonnassSatPos() и UpdateGpsSatPos() в скрипте StructAndDiffs

Скрипт *SceneElementsSwitcher* позволяет с помощью кнопок переключать группы объектов в соответствии с выбранным пользователем подмодулем.

Скрипт *UIElementsSwitcher* переключает UI объекты на сцене.

Скрипт *DiffsFlowController* является главным скриптом управления, контролирующим синхронизированную работу сгруппированных объектов и связанных с ними скриптов.

Скрипт *CamTargeting* задает необходимую позицию и угол поворота камеры относительно точки захвата. Функция *CalculateAngles()* вычисляет угол вращения камеры (рисунок 30).

```
46  Vector2 CalculateAngles()
47  {
48      Vector3 pos = baseRotTr.worldToLocalMatrix.MultiplyPoint(target.position);
49      float angleX = Mathf.Rad2Deg * Mathf.Asin(pos.y / pos.magnitude);
50      float angleY = Mathf.Rad2Deg * Mathf.Atan2(-pos.x, -pos.z);
51      return new Vector2(angleX, angleY);
52  }
```

Рисунок 30 – Функция CalculateAngles() в скрипте CamTargeting

Скрипт *SunRotationControl* меняет угол поворота источника освещения (Солнца). Это необходимо, так как в большинстве подмодулей достаточно показать естественную симуляцию работы Солнца, а в подмодуле «Сегмент

управления» необходимо осветить всю поверхность Земли для наглядности расположения наземных объектов.

Скрипт *GroundWaves* осуществляет работу системы частиц направленного сигнала в виде потокового луча. Система частиц отрабатывает 4 раза в соответствии с четырьмя шагами перемещения частиц: спутник – наземная станица; наземная станция – главная станция; главная станция; – антенна; антенна – спутник (рисунок 31).

```
79   IEnumerator WavesMover()
80   {
81     // satellite - station (1st step)-----------------------------------
82     Vector3 dir1 = wavesNavPoints[1] - wavesNavPoints[0];
83     float dist1 = dir1.magnitude;
84     float delta1 = linearWavesDiff * (numWaves - 1);
85     float step1Time = (dist1 + delta1) / linearSignalSpeed;
86     float step1Timer = 0f;
87     Quaternion rot1 = Quaternion.LookRotation(dir1);
88     for (int i = 0; i < numWaves; i++) waves[i].transform.rotation = rot1;
89
90     while (step1Timer < step1Time)
91     {
92       float progr = step1Timer / step1Time;
93
94       for (int i = 0; i < numWaves; i++)
95       {
96         float pos = (progr * (dist1 + delta1) - linearWavesDiff * i)/ dist1;
97         bool waveActive = (pos >= 0f) && (pos < 1f);
98         waves[i].transform.position = pos * wavesNavPoints[1] + (1f - pos) * wavesNavPoints[0];
99         if (waves[i].activeSelf != waveActive) waves[i].SetActive(waveActive);
100      }
101      step1Timer += Time.deltaTime;
102      yield return null;
103    }
```

Рисунок 31 – Пример реализации одного из шагов работы системы частиц в скрипте *GroundWaves*

Скрипт *SignalFormingController* отвечает за изменение цвета системы частиц объекта визуализированного «облака» сигнала, который применяет в подмодулея «Формирование сигнала».

3. *Взаимодействие с пользователем*

Данный модуль разделен на 6 подмодулей. Переключение происходит с помощью кнопок в боковом меню слева. В большинстве подмодулей демонстрируется пошаговая работа подмодуля. Переключение шагов происходит с помощью кнопок снизу холста. Сопровождающая текстовая информация располагается в правой части экрана (рисунок 32).

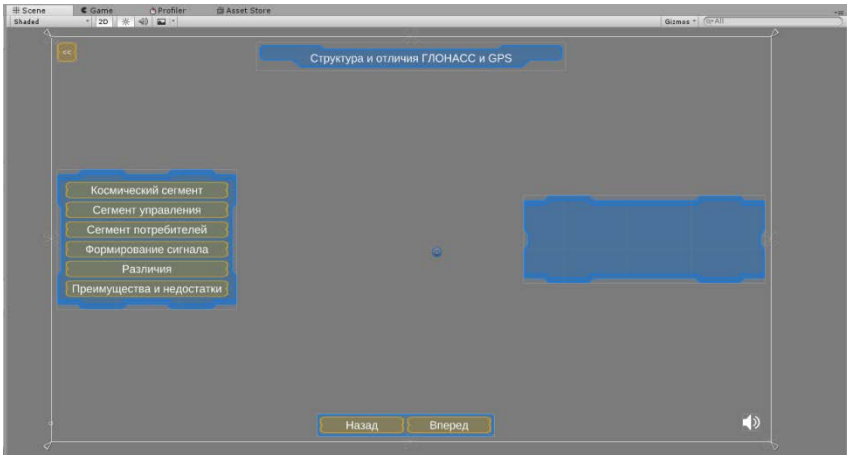


Рисунок 32 – UI элементы на Canvas сцены «Структура и отличия ССН»

3.2.4. Сцена «Принципы определения местоположения» – 2 уровень

На данную сцену можно попасть из сцены «Главное меню». Она представляет из себя интерактивный модуль, посвященный принципам определения местоположения объекта. Демонстрируется пошаговый алгоритм трилатерации (метод определения местоположения объекта) с возможностью выбора пользователем местоположения условного приемника на поверхности Земли.

1. *Объекты сцены.*

В данной сцене представлены следующие ключевые объекты (рисунок 33):

- камера;
- спутник GPS;
- спутник ГЛОНАСС;
- орбита GPS (с визуализацией);
- орбита ГЛОНАСС (с визуализацией);
- планета Земля;
- холст (англ. «Canvas») с объектами UI;
- TrilatProcessFlowController;
- audioSetter;

– условный объект-приемник в виде красной сферы;
– полупрозрачная полусфера разных цветов;
– полупрозрачный конус разных цветов;
– полупрозрачный сегмент полусферы разных цветов;
– круговая линия пересечения;

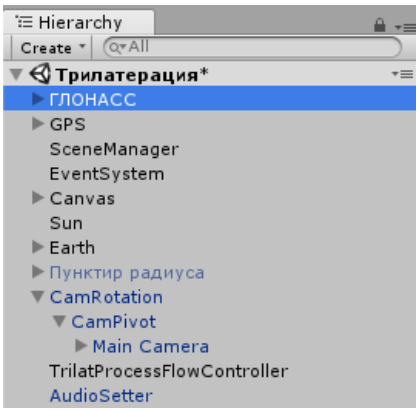


Рисунок 33 – Список объектов на сцене "Принципы определения местоположения"

Камеру можно вращать вокруг Земли аналогичным образом, как и в сцене «Главное меню».

Спутники и орбиты работают по такому же принципу, как и в сцене «Структура и отличия ССН». Однако на определенном шаге проигрывания сцены все спутники останавливаются. Также в зависимости от действий пользователя на сцене активированы либо спутники и орбиты GPS, либо ГЛОНАСС.

Земля вращается аналогичным образом, как и в предыдущих сценах. Однако, также, как и спутники, она останавливается во время определенного шага проигрывания сцены. Атмосфера Земли двигается в течение всего воспроизведения сцены. Важную роль на данной сцене имеет компонент Sphere Collider модели Земли. Благодаря ему появляется возможность обрабатывать клик мыши по координате клика в зависимости от места клика на поверхности Земли.

TrilatProcessFlowController является управляющим объектом сцены, в основе которого лежит одноименный скрипт.

Условный объект-приемник в виде красной сферы визуализирует клик мыши пользователем по поверхности Земли с дальнейшим сохранением позиции до конечного шага сцены.

Полупрозрачная полусфера разных цветов появляется рядом с антенной спутника и динамически увеличивается в размерах до касания условного объекта-приемника. Таким образом демонстрируется интересующая целевая область распространения сигнала спутника. Появляется несколько последовательных полусфер разных цветов в соответствии со сценарием проигрывания сцены. Разные цвета представлены для наглядности происходящего.

Полупрозрачный конус разных цветов появляется рядом с антенной спутника и направлен своим основанием к объекту-приемнику. Конус появляется в двух случаях. В первом случае после выбора точки объекта-приемника появляется широкий полупрозрачный конус, вершина которого выходит из антенны спутника, демонстрирующий, что объект находится в зоне распространения сигнала спутника. Во втором случае, после исчезновения вышеописанной полусферы появляется конус, проецирующий часть данной полусферы. Через несколько секунд конус исчезает.

Полупрозрачный сегмент полусферы разных цветов появляется после исчезновения вышеописанного конуса во втором случае. Сегмент проецирует основание конуса и часть поверхности полусферы. Таким образом демонстрируется малая часть поверхности полусферы, в центре которой располагается объект-приемник.

Круговая линия пересечения генерируется в местах пересечения сегментов полусферы и, таким образом, наглядно демонстрирует, что сегменты действительно пересекаются, в каких именно местах они

пересекаются и что точка пересечения четырех спутников действительно соответствует координатам объекта-приемника.

2. *Скрипты*

Главным скриптом является *TrilatProcessFlowController*, который управляет всей работой данной сцены, а именно интерактивным взаимодействием с пользователем, вычислениями на основе действий пользователя и последовательным процессом демонстрации принципа определения местоположения. Управляющий скрипт как компонент располагается в соответствующем объекте сцены (рисунок 34).

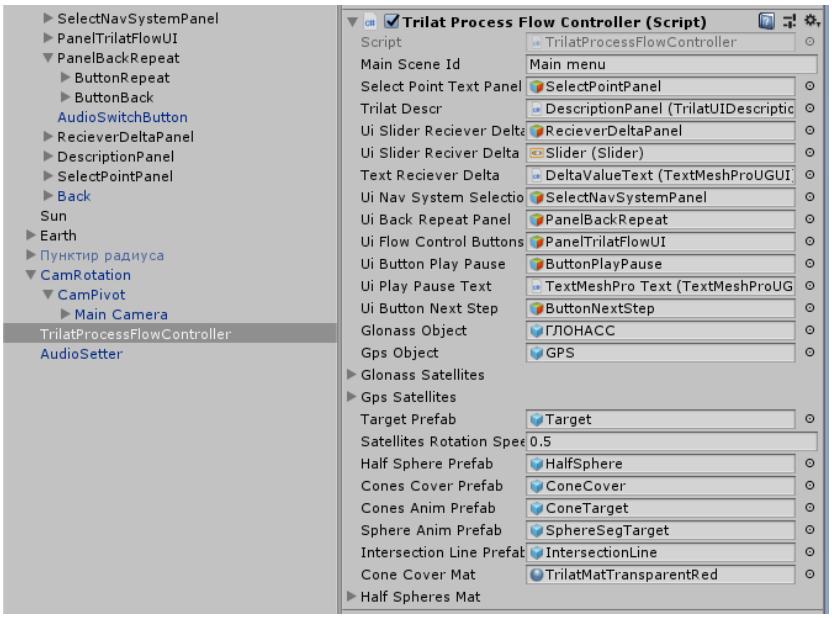


Рисунок 34 – Скрипт TrilatProcessFlowController как компонент на сцене

Синхронизированное и равноудаленное вращение спутников вокруг Земли, их углы поворотов реализованы аналогичным образом, как и на сцене «Структура и отличия ССН» с помощью методов *UpdateRotationSatelliteValue(), UpdateGlonassSatPos(), UpdateGpsSatPos()*.

Методы *SelectGLONASS() и SelectGPS()* активируют объекты соответствующих ССН. Выбор системы осуществляется пользователем при старте сцены. Изначально обе группы объектов отключены.

Метод *PointSelect()* является ключевым в данном скрипте. Выбор ближайших спутников к объекту-приемнику происходит следующим образом:

где *sat => Vector3.SqrMagnitude(sat.GetPosition() - target.transform.position)* это лямбда-функция, которая сортирует спутники по квадрату дальности между спутником и объектом-приемником;

метод сортировки спутников, в качестве аргумента принимающий вышеописанную лямбда-функцию;

метод, который выбирает 4 первых спутника из отсортированных .

Далее происходит генерация линий пересечения секторов (рисунок 35)

```
160     intersections = new GameObject[6];
161     for (int i = 0; i < 6; i++) intersections[i] = Instantiate(intersectionLinePrefab);
162
163     intersections[0].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].transform.position, closestSats[1].transform.position, target.transform.position);
164     intersections[1].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].transform.position, closestSats[2].transform.position, target.transform.position);
165     intersections[2].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[1].transform.position, closestSats[2].transform.position, target.transform.position);
166     intersections[3].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].transform.position, closestSats[3].transform.position, target.transform.position);
167     intersections[4].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[1].transform.position, closestSats[3].transform.position, target.transform.position);
168     intersections[5].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[2].transform.position, closestSats[3].transform.position, target.transform.position);
```

Рисунок 35 – Генерация линий пересечения секторов полусфер

Затем запускается цикл обработки каждого выбранного спутника из четырех. Задается подсветка спутников с помощью контуров:

```
closestSats[i].SetOutline(true);
```

Рассчитывается радиус диска для определения пересечения конуса с Землей:

```
float discr = rEarth * rEarth - rSat * rSat * Mathf.Sin(Mathf.Deg2Rad * satAngle) * Mathf.Sin(Mathf.Deg2Rad * satAngle);
```

На основании этого дискриминанта рассчитывается длина конуса, который соответствует вышеописанному конусу.

```
float coneLength = (discr > 0) ? (rSat * Mathf.Cos(Mathf.Deg2Rad * satAngle) - Mathf.Sqrt(discr)) : rSat;
```

Далее генерируются вышеописанные полусферы и конусы. Запускаются анимации этих фигур с помощью метода *SatAnimation()*.

Метод *SwitchPausePlay()* отвечает за изменение текста кнопки «Пауза» и за активность кнопки «Следующий шаг».

Метод *ReturnToMain()* позволяет после прохождения всех шагов повторно проиграть сцену с возможностью выбора другой системы (ГЛОНАСС, GPS)/

Метод *SelectNewPoint()* позволяет после прохождения всех шагов повторно проиграть сцену с той же выбранной системой, но с повторным выбором новой точки местоположения объекта-приемника.

Метод *ClearScene()* очищает сцену от ненужных объектов, сбрасывает значение некоторых переменных. Вызывается в двух предыдущих описанных методах.

Метод *ChangeRadiuses()* изменяет радиусы сгенерированных линий пересечений сегментов в зависимости от выбранного пользователем значения ошибки хода часов приемника с помощью ползунка.

3. Взаимодействие с пользователем

Данный модуль разбит на последовательные шаги. Сначала пользователю предлагается выбрать, с какой ССН он будет работать. После выбора системы появляются соответствующие орбиты и спутники выбранной системы. Затем пользователю предлагается выбрать точку местоположения условного объекта-приемника на поверхности Земли с помощью нажатия курсора мыши. Пошагово демонстрируется трилатерация. Шаги переключаются с помощью кнопки в нижней части экрана. Во время динамических действий на сцене у пользователя есть возможность поставить на паузу проигрывание сцены. После прохождения всех шагов пользователю предлагается повторно выбрать систему или выбрать другую точку на поверхности Земли. Кнопка «<<» позволяет вернутся в главное меню приложения (рисунок 36).

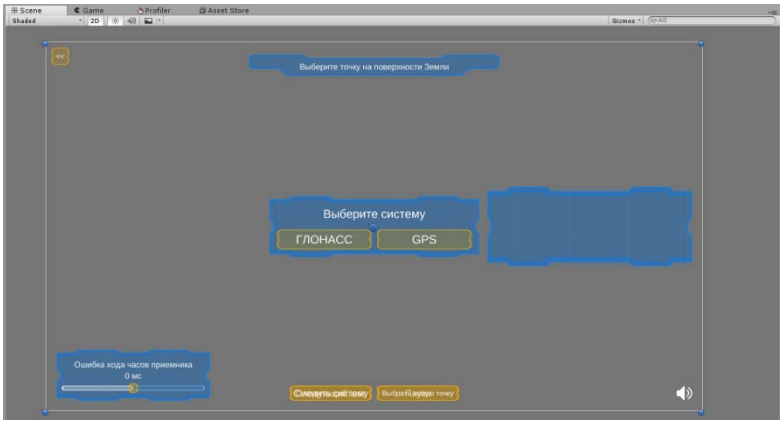


Рисунок 36 – UI элементы на Canvas сцены «Принцип определения местоположения»

3.2.5. Сцена «Факторы, влияющие на точность определения координат» – 2 уровень

На данную сцену можно попасть из сцены «Главное меню». Она представляет из себя интерактивный модуль, посвященный факторам, влияющим на точность определения координат приемника. Модуль разделен на 5 подмодулей, соответствующих 5 ключевым группам факторов, и являющихся сценами 3-го уровня.

Сцена по своей логике и функциональности повторяет сцену «Главное меню». По сути, сцена является таким же меню для факторов. Отличием является стилизованное 3D меню в отличие от классического меню на Canvas. Кнопки переключения на сцены 3-го уровня расположены в 3D пространстве сцены и всегда повернуты «лицом» по отношению к камере (рисунок 37).

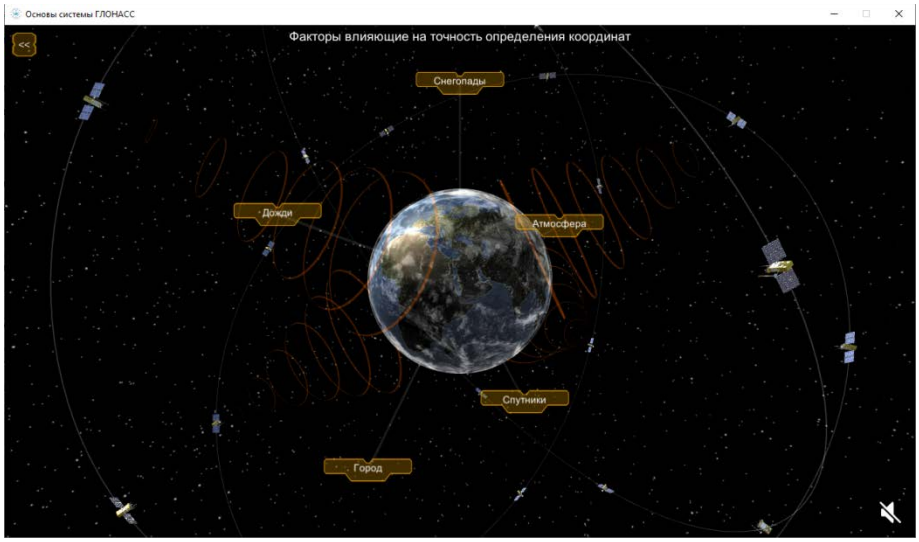


Рисунок 37 – Работа сцены "Факторы, влияющие на точность определения координат"

3.2.6. Сцена «Дожди» – 3 уровень

На данную сцену можно попасть из сцены «Факторы, влияющие на точность определения координат». Она представляет из себя интерактивный модуль, посвященный группе факторов атмосферных осадков, а именно дождю. Демонстрируется влияние разных степеней количества дождевых осадков на распространение сигнала со спутника. Для наглядности сцена представлена в виде острова, окруженного водой.

1. *Объекты сцены.*

В данной сцене представлены следующие ключевые объекты (рисунок 38):

- камера;
- холст (англ. «Canvas») с объектами UI;
- audioSetter;
- terrain (местность);
- вода;
- дождь;
- weather;
- radioWaves;

– audioSetter;

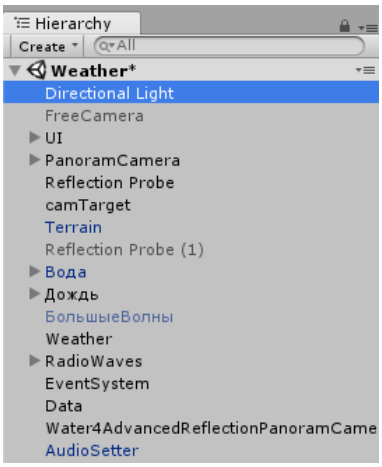


Рисунок 38 – Список объектов на сцене "Дожди"

Камеру можно вращать вокруг точки, привязанной к центру острова. Вращение происходит аналогичным образом, как и в предыдущих сценах.

Terrain имеет одноименное название в соответствии со своими компонентами. Это ландшафт, представляющий из себя остров.

Вода представляет из себя динамическую текстуру, симулирующую работу водной глади моря. На данной сцене был использован готовый Unity пакет с водой, реализованный сторонним автором.

Дождь представляет из себя систему частиц. Представлен в трех вариантах: слабый, средний и сильный дождь. В зависимости от действий пользователя выбирается соответствующий дождь. Каждый вариант дождя имеет свою систему частиц.

Weather в основе содержит одноименный скрипт и является управляющим объектом сцены.

RadioWaves аналогично дождю представляет из себя систему частиц. Каждая частица выглядит как желтое кольцо разной степени прозрачности в зависимости от выбранных пользователем погодных условий. Данная система частиц наглядно показывает абстрактный сигнал со спутника как множество колец, в совокупности напоминающих фигуру цилиндра. Кольца

друг за другом движутся снизу вверх, как бы показывая направление распространения сигнала от спутника.

2. *Скрипты*

Скрипт *Weather* отвечает за переключение погодных условий, сигнала распространения радиоволн спутника, скайбоксов (рисунок 39).

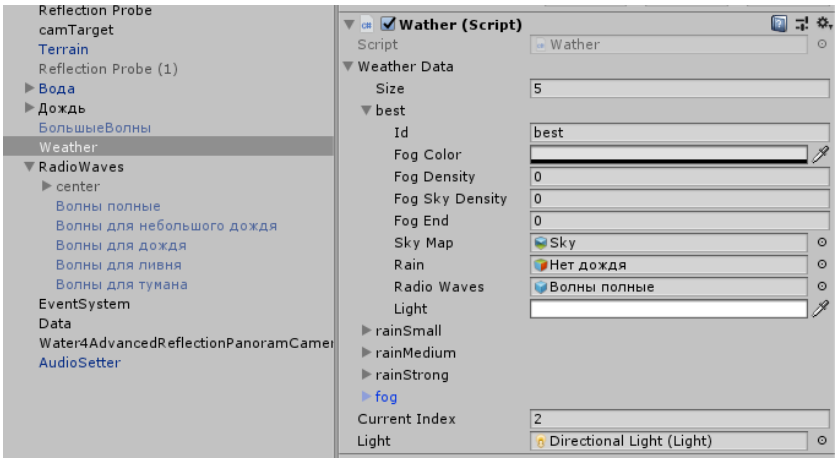


Рисунок 39 – Скрипт Weather как компонент объекта

3. *Взаимодействие с пользователем*

На данной сцене пользователь с помощью переключателей в нижней части экрана может изменять степень дождевых осадков. Также возможно переключение на туман. В соответствии с выбранными осадками включается соответствующая система частиц осадков, скайбокс, степень туманности и звук. Также в правом верхнем углу дается краткое пояснение в виде UI текстового поля о влиянии выбранных осадков на распространение сигнала (рисунок 40).

Рисунок 40– UI элементы на Canvas сцены «Дожди»

3.2.7. Сцена «Снегопады» – 3 уровень

На данную сцену можно попасть из сцены «Факторы, влияющие на точность определения координат». Она представляет из себя интерактивный модуль, посвященный группе факторов атмосферных осадков, а именно снегопаду. Демонстрируется влияние разных степеней количества снежных осадков на распространение сигнала со спутника. Для наглядности сцена представлена в виде снежных гор (рисунок 41).

Сцена по своей логике и функциональности повторяет сцену «Дожди». Отличие лишь в том, что угол вращения камеры на данной сцене ограничен в соответствии с особенностями ландшафта.

Аналогично можно выбрать 4 степени осадков: отличная погода, небольшой снег, метель, сильный снег. Также демонстрируется степень ослабления сигнала в зависимости от интенсивности осадков.

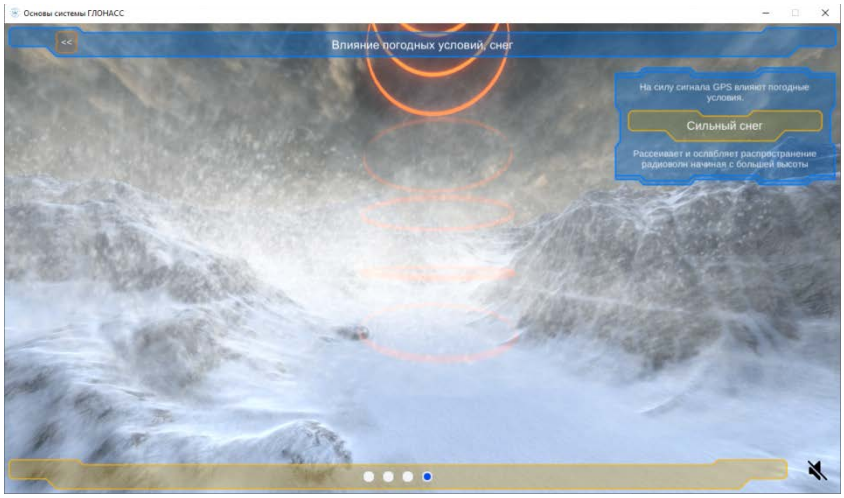


Рисунок 41 – Работа сцены "Снегопады"

3.2.8. Сцена «Атмосфера» – 3 уровень

На данную сцену можно попасть из сцены «Факторы, влияющие на точность определения координат». Она представляет из себя интерактивный модуль, посвященный группе факторов атмосферы. Демонстрируется влияние ионосферы и тропосферы на распространение сигнала со спутника.

1. *Объекты сцены.*

В данной сцене представлены следующие ключевые объекты (рисунок 42):

– камера;
– холст (англ. «Canvas») с объектами UI;
– audioSetter;
– спутник;
– Земля;
– ионосфера;
– тропосфера;
– радиоволны в виде системы частиц;

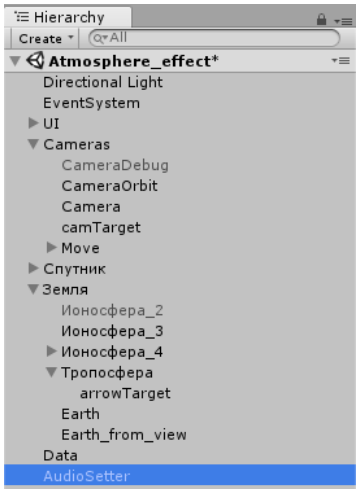


Рисунок 42 – Список объектов на сцене "Атмосфера"

На сцене находится *3 камеры*. Одна камера является основной и в её кадрах демонстрируется основная работа сцены, а именно прохождение радиоволн через космос, ионосферу и тропосферу. Вторая камера предназначена для наблюдения за испусканием радиоволн со спутника. Она работает параллельно с основной камерой и демонстрируется в небольшой окне поверх основной камеры. Третья камера является динамической и включается пользователем. При включении срабатывает анимация движения камеры, и она начинает следить за перемещением радиоволны до поверхности Земли. При этом основная камера отключается.

Спутник статично располагается на одной позиции под наблюдением одной из камер.

Радиоволны в виде системы частиц испускаются из антенны спутника и долетают до поверхности Земли. При этом на сцене участвуют два варианта волн. Первый непосредственно начинает свою работу рядом с антенной спутника и имеет скорость в 20 юнитов в секунду (для наглядности работы волн), а второй как бы продолжает работу первой и запускается в верхней части ионосферы в момент исчезновения первого кольца первого варианта волн на поверхности ионосферы со скоростью 6 юн/сек. Также в режиме прохождения волны через тропосферу у второго варианта волн кольца при начале прохождения через тропосферу становятся полупрозрачными.

Разница скоростей радиоволн в космосе и в ионосфере; прозрачность при прохождении через тропосферу реализована для наглядной демонстрации того, как влияет атмосфера на прохождение радиоволны через неё.

Ионосфера представляет из себя модель сферы с наложенным шейдером Shield. Это шейдер со специальным визуальным эффектом для создания щитовидных форм. На данной сцене ионосфера в зависимости от выбора режима пользователем имеет разные размеры. В одном случае пропорции высоты ионосферы относительно поверхности Земли подобраны в соответствии с реальными значениями. Второй режим демонстрации увеличивает ионосферу в 16 раз.

Тропосфера взята из оригинальной комплексной модели Земли, которая используется во многих модулях данного проекта. Используется аналогичным образом, как и ионосфера. В пропорциях к ионосфере в реальности тропосфера располагается значительно ближе к поверхности Земли. При правильных пропорциях атмосферы к Земле в первом режиме практически не видно ни тропосферу, ни то, как через неё проходят

радиоволны. Для этого реализован второй режим, в котором в 16 раз относительно реальных пропорций увеличивается размер ионосферы и тропосферы. Таким образом, можно наглядно увидеть прохождение радиоволн через тропосферу.

2. *Скрипты*

Скрипт *Rotate_the_object* медленно вращает ионосферу вокруг Земли.

Скрипт *AtmosphereEffectManager* отвечает за работу данной сцены, а именно за работу выбранных пользователем режимов.

Скрипт *Distance* выполняет работу перемещения главной камеры.

3. *Взаимодействие с пользователем*

На данной сцене пользователь с помощью переключателей в нижней части экрана может изменять режим демонстрации прохождения радиоволн через атмосферу. В соответствии с выбранным режимом камера переключается или изменяет свою позицию. При втором режиме работа ионосфера и тропосфера масштабируется в 16 раз, камера фокусируется на тропосфере, а радиоволны начинают становится полупрозрачными при прохождении через тропосферу. При переключении на третий режим включается динамическая камера, которая следит за перемещением радиоволны от спутника до поверхности Земли.

В левой верхней части экрана располагается небольшой окно с дополнительной камерой, которая демонстрирует испускание волн спутником. В правом верхнем углу дается краткое пояснение в виде UI текстового поля по каждому режиму работу (рисунок 43).

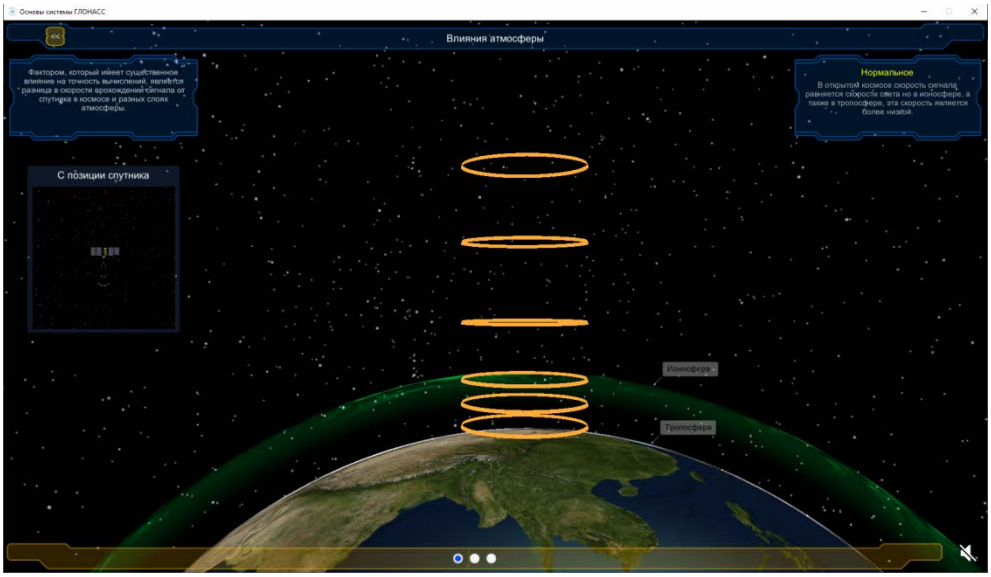


Рисунок 43 – Работа сцены "Атмосфера"

3.2.9. Сцена «Спутники» – 3 уровень

На данную сцену можно попасть из сцены «Факторы, влияющие на точность определения координат». Она представляет из себя интерактивный модуль, посвященный группе факторов качества геометрии спутников. Демонстрируется влияние расположения спутников относительно друг друга на качество сигнала в 2D и 3D режимах.

1. *Объекты сцены*

В данной сцене представлены следующие ключевые объекты (рисунок 44):

– камера;
– холст (англ. «Canvas») с объектами UI;
– audioSetter;
– спутник;
– Земля;
– полупрозрачный 3D конус;
– полупрозрачный 2D конус;
– полупрозрачная 3D фигура в виде объемного сегмента сферы;
– полупрозрачная 2D фигура в виде широкого сегмента сферы;
– полупрозрачные плоские геометрические четырехугольные фигуры;

– объемные геометрические фигуры;

Рисунок 44 – Список объектов на сцене "Спутники"

Камера имеет два режима работы: перспективная и ортографическая проекция. В зависимости от выбранного режима сцены 2D или 3D выставляется соответствующий режим работы камеры: ортографическая проекция в 2D и перспективная проекция в 3D.

Два спутника располагаются над поверхностью Земли и изменяют свое положение относительно друг друга в зависимости от выбранного пользователем шага.

В зависимости от выбранного режима 2D / 3D на сцене изображаются *полупрозрачные 2D / 3D конусы* белого цвета, вершины которых располагаются у антенн спутников. Конусы проецируют исходящий сигнал от спутников. Они изменяют свой угол относительно условного объекта-приемника на поверхности Земли в зависимости от выбранного пользователем шага, то есть при изменении позиции спутников конусы также изменяют свое положение.

В зависимости от выбранного режима 2D / 3D на сцене изображаются *полупрозрачные 2D / 3D фигуры в виде широкого / объемного сегмента сферы* зеленого цвета с контурными линиями (Outline) на границах фигур. Данные фигуры являются продолжением вышеописанных конусов и, по сути,

являются их основанием. Они демонстрируют, что у каждого спутника есть погрешность измерений сигнала, то есть нельзя с максимальной точностью оценить на какой именно высоте находится приемник относительно спутника. Данная область и показывает данную погрешность. Она не имеет точных значений по ширине, а лишь информирует о наличии такой погрешности.

В зависимости от выбранного режима 2D / 3D на сцене изображаются п*олупрозрачные 2D / 3D симметричные геометрические фигуры зеленого цвета.* Фигуры изменяют свою форму в зависимости от выбранного пользователем шага. Они соответствуют пересечениям вышеописанных сегментов сферы, исходящих от спутников. Фигуры не генерируются автоматически, а были вручную созданы заранее.

2. Скрипты

DragMouseOrbitOrigin позволяет вращать камеру в 3D режиме вокруг точки, соответствующей объекту-приемнику.

CameraMode меняет режим и настройки камеры в соответствии с выбранным режимом просмотра 2D / 3D.

RayIntersect управляет изменением позиций и углов спутников геометрических объектов, а также изменением геометрических фигур в соответствии с выбранным шагом или режимом работы 2D / 3D.

3. Взаимодействие с пользователем

На данной сцене пользователь с помощью переключателей в нижней части экрана может менять шаги демонстрации разных положений спутников относительно друг друга, а также режим просмотра в 2D и 3D с помощью соответствующей кнопки в правом нижнем углу экрана. В 3D режиме появляется возможность вращать камерой вокруг объекта-приемника (рисунок 45). В правой части экрана дается сопровождающая пояснительная информация.

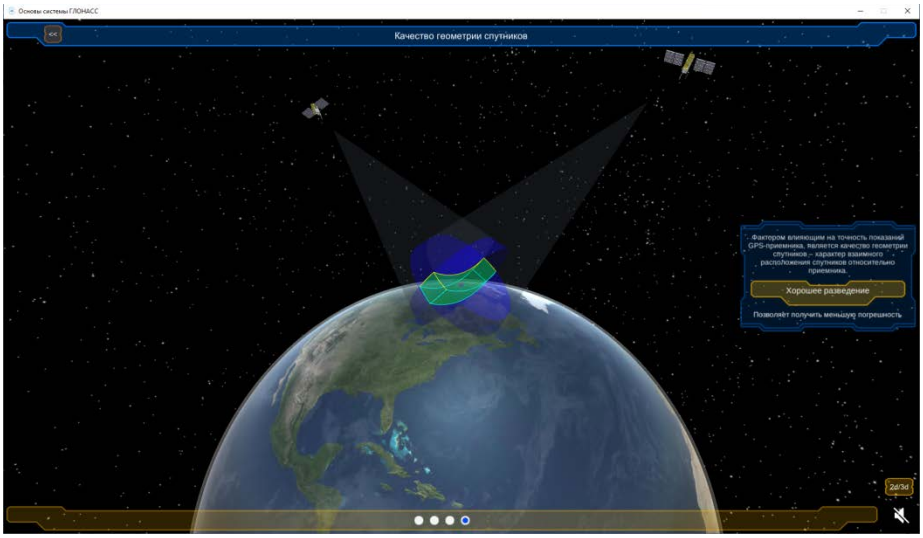


Рисунок 45 – Работа сцены "Спутники"

3.2.10. Сцена «Город» – 3 уровень

На данную сцену можно попасть из сцены «Факторы, влияющие на точность определения координат». Она представляет из себя интерактивный модуль, посвященный группе факторов влияния городской среды. Демонстрируются различные ситуации в условиях городской среды, которые могут влиять на распространение сигнала со спутника.

1. *Объекты сцены*

В данной сцене представлены следующие ключевые объекты (рисунок 46):

– камера;
– холст (англ. «Canvas») с объектами UI;
– audioSetter;
– грунт;
– точки;
– различные волны в виде систем частиц;
– городские объекты;

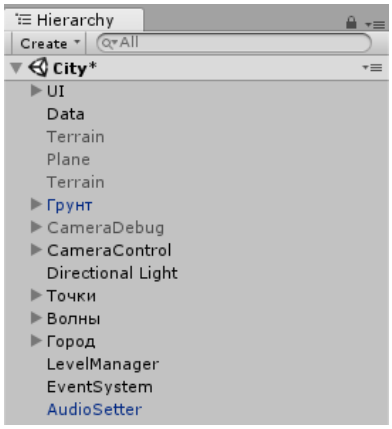


Рисунок 46 – Список объектов на сцене "Город"

Камера перемещается между заранее заданными точками на сцене в зависимости от выбранного пользователем шага. На некоторых точках камеру можно вращать в ограниченном секторе в зависимости от особенности её расположения.

Грунт является декоративным объектом и представляет из себя плоский ландшафт местности. В нем сгруппированы различные виды деревьев, которые также являются декоративными объектами.

Точки являются пустыми объектами, которые представляют из себя наборы координат, необходимые для перемещения камеры по позициям, соответствующие этим координатам.

Волны распространения сигналов в виде систем частиц представлены в двух вариантах в зависимости от выбранной пользователем ситуации: в виде конусов, встречающихся в вышеописанных сценах; в виде плоскости, напоминающей распространение волн на поверхности воды от центральной точки. Второй вид волн всегда сопровождается работой первого вида, демонстрируя в упрощенном виде отражения волн второго вида от поверхностей.

Городские объекты являются декоративной группой различных объектов, создающих имитацию городской среды. На сцене встречаются такие городские объекты, как многоэтажные жилые дома, комната в здании,

подземный переход, дорога, внутренний двор в виде асфальтированного участка, вышка сотовой связи, дорожные знаки, модели человека, деревья,

2. *Скрипты*

CityManager задает работу данной сцены. В зависимости от выбранной ситуации на сцене, он изменяет координаты камеры с помощью линейной интерполяции. За счет этого достигается плавное перемещение и вращение камеру. Также данный скрипт переключает наборы объектов, соответствующих данной ситуации (рисунок 47).

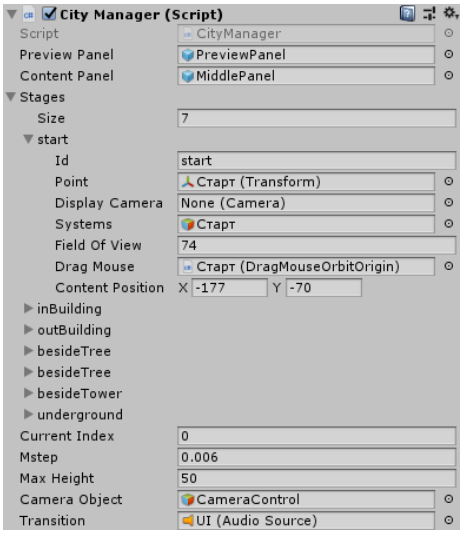


Рисунок 47 – Скрипт CityManager как компонент объекта

3. *Взаимодействие с пользователем*

На данной сцене пользователь с помощью переключателей в нижней части экрана может выбирать различные ситуации городской среды, при которых происходит ухудшение принимаемого сигнала со спутника. В правой части экрана дается пояснительная информация для каждой ситуации (рисунок 48).

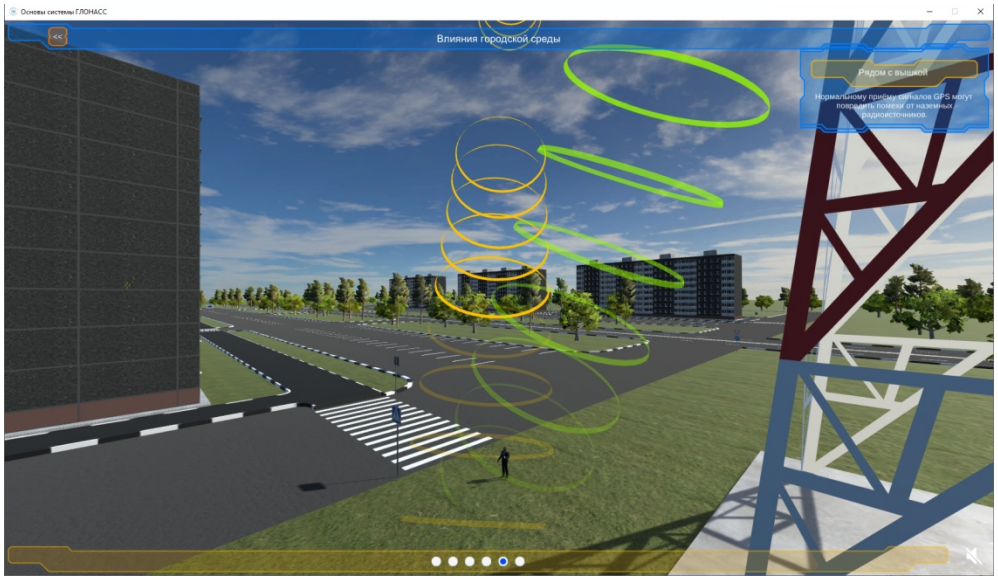


Рисунок 48 – Работа сцены "Город"

3.2.11. Сцена «Показатели точности» – 2 уровень

На данную сцену можно попасть из сцены «Главное меню». Она представляет из себя интерактивный модуль, посвященный показателям точности определения координат. В данном модуле представлена преимущественно стилизованная текстовая информация с возможностью выбора глав (рисунок 49).

Сцена по своей логике и функциональности повторяет сцену «История ССН». Переключение между главами осуществляется кнопками «Вперед» и «Назад», либо с помощью выпадающего меню для быстрого переключения на интересующую главу.

Данный модуль дополняет собой модуль «Факторы, влияющие на точность сигнала». Приводит теоретическую справку и более конкретные величины и формулы, связанные с показателями точности координат и их погрешностями, а также дает дополнительную полезную теоретическую информацию об ССН.

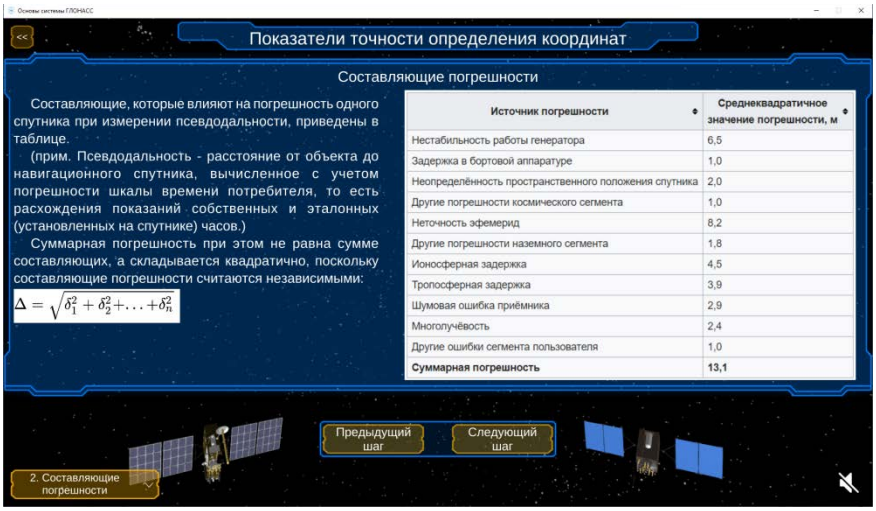


Рисунок 49 – Работа сцены "Показатели точности"

3.2.12. Сцена «Практическое применение ССН» – 2 уровень

На данную сцену можно попасть из сцены «Главное меню». Она представляет из себя интерактивный модуль, посвященный практическому применению ССН. Описываются сферы области, в которых применяются ССН, и демонстрируются абстрактные модели данных сфер.

1. *Объекты сцены*

В данной сцене представлены следующие ключевые объекты (рисунок 50):

– камера;
– холст (англ. «Canvas») с объектами UI;
– audioSetter;
– городские объекты;
– остров;
– модели;
– points;

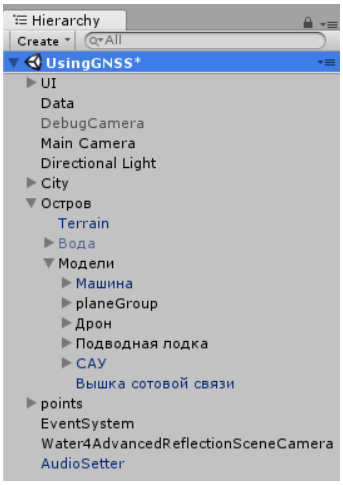


Рисунок 50 – Список объектов на сцене "Практическое применение ССН"

Камера перемещается между заранее заданными точками на сцене в зависимости от выбранного пользователем шага. На некоторых точках камеру можно вращать в ограниченном секторе в зависимости от особенности её расположения. Работа камеры аналогична работе камеры на сцене «Город».

Городские объекты включают в себя часть декораций, взятых со сцены «Город». Они необходимы для визуального вписывания в интерьер модели автомобиля.

Остров взят со сцены «Дожди». В него сгруппирована вода, которая также взята со сцены «Дожди», и абстрактные технические модели, а именно: автомобиль, пассажирский самолет, дрон, подводная лодка, САУ, вышка сотовой связи. Модели представлены для формирования образа у пользователя сферы применения ССН.

Points являются пустыми объектами, которые представляют из себя наборы координат, необходимые для перемещения камеры по позициям, соответствующие этим координатам.

2. *Скрипты*

Основным скриптом сцены является *UsingGNSSManager*. Работа данного скрипта аналогична работе скрипта *CityManager*, который задает работу сцены «Город». В зависимости от выбранной ситуации на сцене

данный скрипт изменяет координаты камеры с помощью линейной интерполяции. За счет этого достигается плавное перемещение и вращение камеру. Однако при переключении пользователем шага никакие объекты не переключаются.

Скрипт *Rotation_the_object* позволяет имитировать движение самолета вокруг точки, расположенной примерно над центром острова. Также данный скрипт отвечает за вращение камеры вокруг объектов.

3. *Взаимодействие с пользователем*

На данной сцене пользователь с помощью переключателей в нижней части экрана может выбирать различные примеры применения ССН. В правой части экрана дается описание для каждого примера (рисунок 51).

Последний переключатель показывает сводную информацию по всем сферам применения ССН.

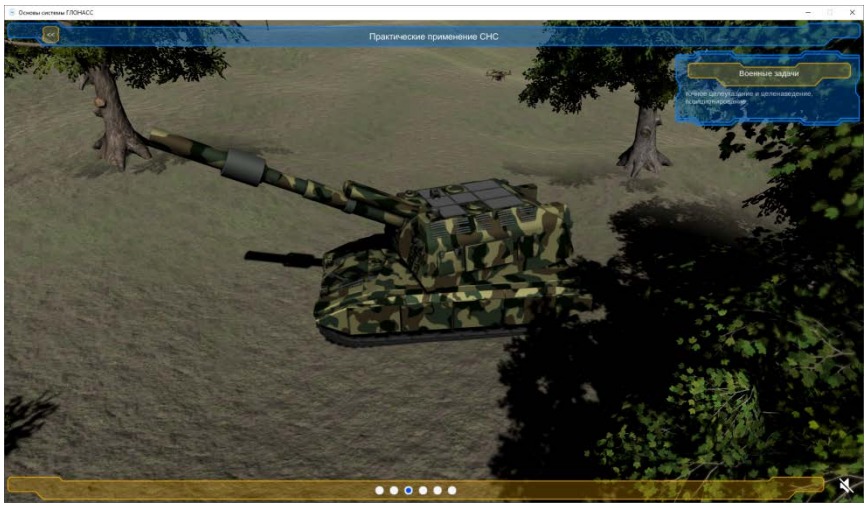


Рисунок 51 – Работа сцены "Практическое применение ССН"

4. РЕАЛИЗАЦИЯ

4.1. СБОРКА И ЗАПУСК ПРИЛОЖЕНИЯ

После проектирования программного комплекса Unity проект был собран в рабочую сборку под операционную систему Windows (рисунок 52).

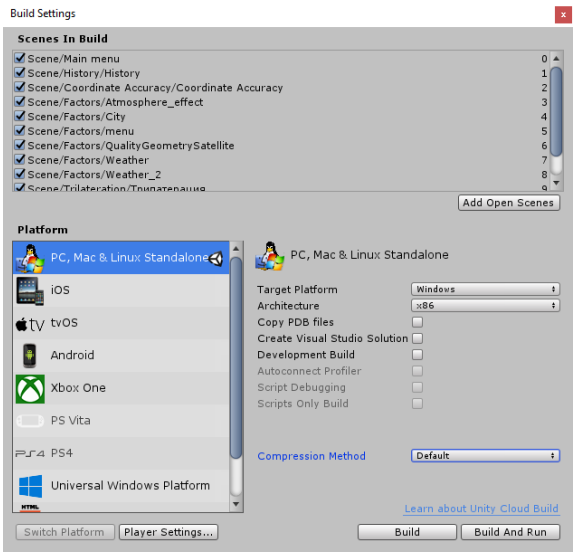


Рисунок 52 – Сборка Unity проекта в приложение

После выполнения исполняемого файла запускается стандартное стартовое окно настроек Unity. Пользователь может выбрать разрешение экрана, режим окна, пресет настроек графики и монитор, на котором будет запущено приложение (рисунок 53).

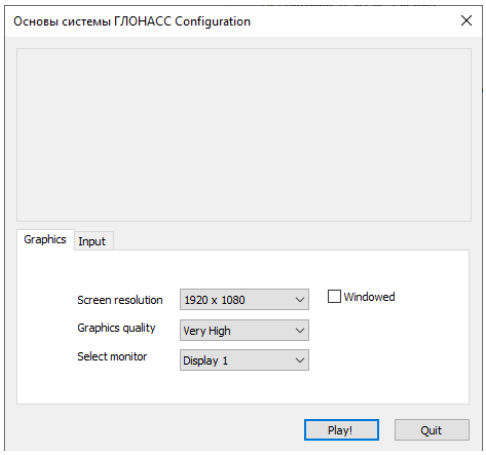


Рисунок 53 – Окно настроек приложения

4.2. СЦЕНАРИЙ РАБОТЫ ПРИЛОЖЕНИЯ

Приложение разделено на 6 модулей, описывающих и демонстрирующих ключевые темы ССН.

Запуск программы начинается с главного меню, представляющего из себя навигацию по интересующим модулям (рисунок 54). На фоне вращается Земля и спутники ГЛОНАСС и GPS. Сцена является интерактивной и позволяет рассмотреть Землю и спутники с разных ракурсов.

Кнопки перемещения по модулям расположены в порядке, рекомендуемом к изучению.

Вся работа программы наполнена звуковым сопровождением. Играет фоновая музыка, которая не отвлекает от изучения. При необходимости её можно отключить соответствующей кнопкой на экране. Нажатие на кнопки также озвучены. В соответствии с логикой различных сцен на некоторых из них имеются дополнительные звуки, а звуковое сопровождение может отличаться.

Далее рассмотрены сценарии работы модулей.

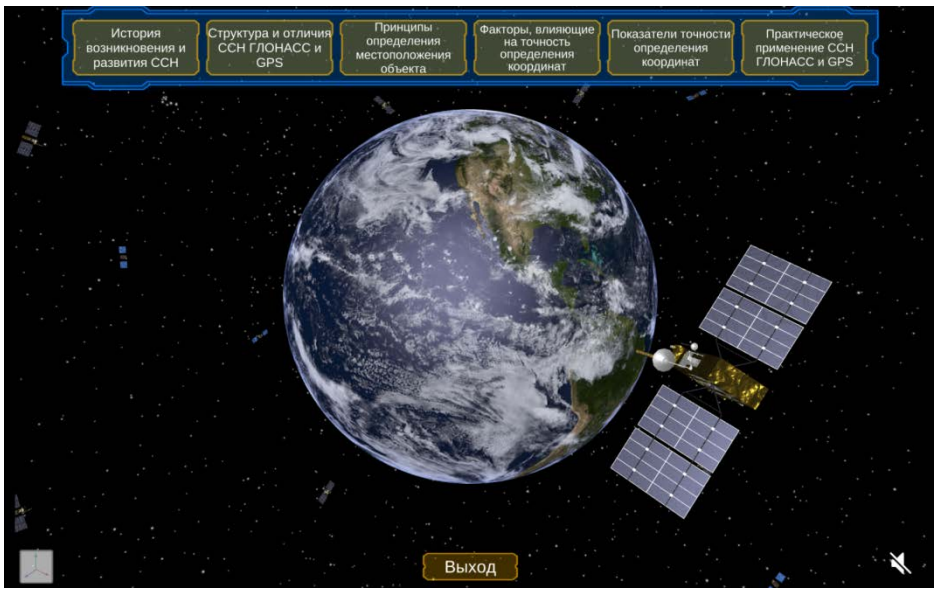


Рисунок 54 – Главное меню

4.2.1. «История возникновения и развития ССН»

Первый модуль включает в себя описание возникновения ССН, этапы их развития вплоть до текущего времени и представляет собой модель с возможностью выбора исследуемого периода.

Хронология каждой системы разбита на ключевые года разработок данных систем, переключаться между которыми можно с помощью соответствующих стилизованных кнопок (рисунок 55).

В окне сравнения развития систем появляется временная шкала, наглядно демонстрирующая хронологию развития обоих систем (рисунок 56).

Рисунок 55 – История развития ГЛОНАСС

Рисунок 56 – Сравнение развития ССН по временной шкале

4.2.2. «Структура и отличия ГЛОНАСС и GPS»

Второй модуль представляет из себя структуру и отличия ГЛОНАСС и GPS. Слева располагается меню, которая разделяет данный модуль на подмодули:

- «Космический сегмент» наглядно показывает структуру этих навигационных систем, то есть их орбиты, количество спутников, высоту над поверхностью Земли, а также модели самих спутников, которые в настоящее время введены в эксплуатацию (рисунок 57).
- «Сегмент управления» показывает места расположения наземного комплекса управления данными системами (в него входят главная станция управления, станции слежения и наземные антенны) (рисунок 58), а также наглядно демонстрирует и описывает процесс передачи данных со спутников на Землю и наоборот (рисунок 59).
- «Сегмент потребителей» вкратце описывает практическое применение данных систем. Более подробное применение этих систем описывается в модуле «Практическое применение ССН»
- В подмодуле «Формирование сигнала» описывается из чего состоит сигнал, отправляемый со спутников на Землю (рисунок 60).
- В «Различиях» приведено сравнительное окно численных характеристик систем ГЛОНАСС и GPS, а также визуальное сравнение моделей спутников.
- В «Преимуществах и недостатках» кратко описаны плюсы и минусы обоих систем.

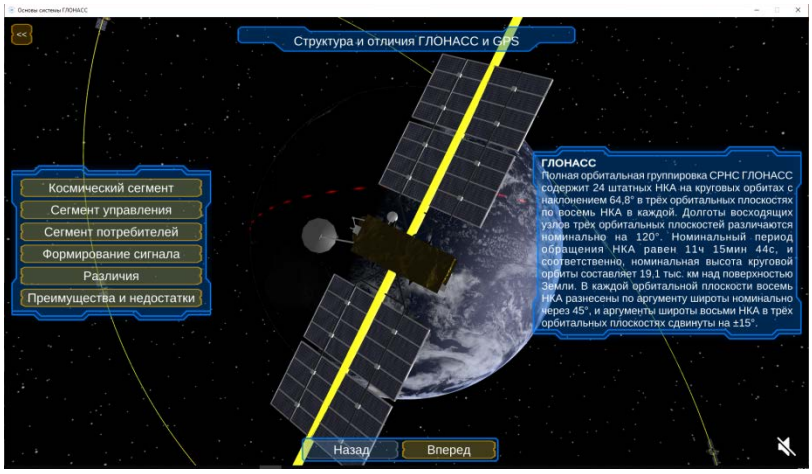


Рисунок 57 – Космический сегмент ГЛОНАСС

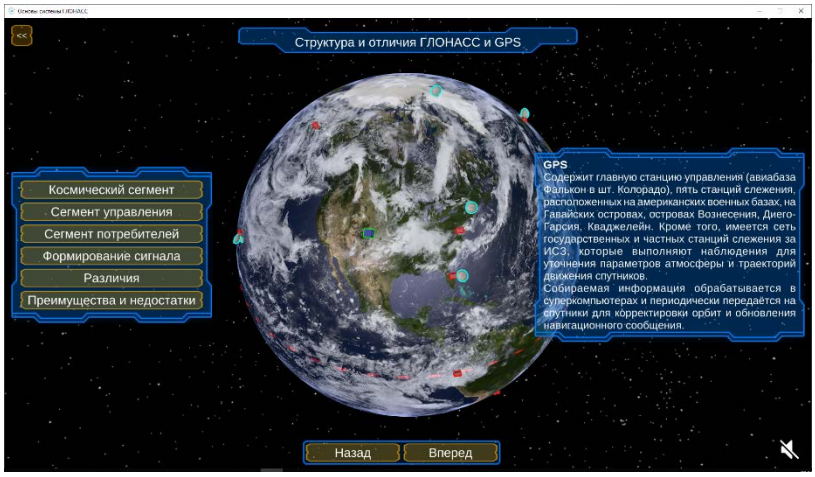


Рисунок 58 – Сегмент управления GPS на Земле

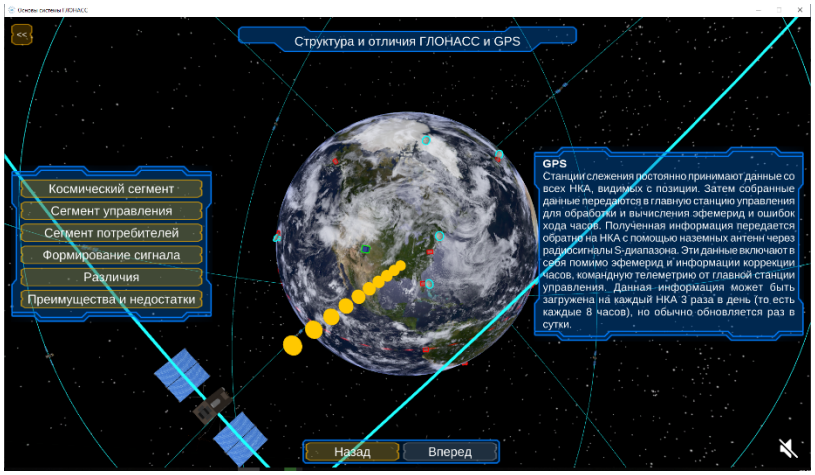


Рисунок 59 – Процесс передачи сигнала на Землю и обратно

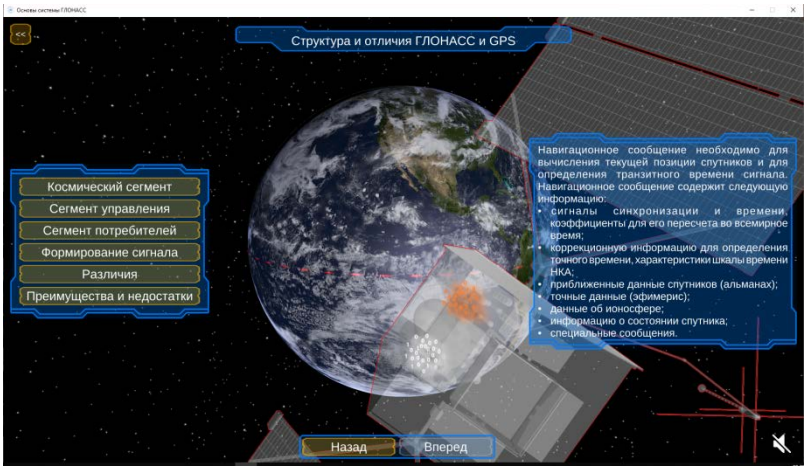


Рисунок 60 – Составляющие сигнала спутника

4.2.3. «Принципы определения местоположения»

Модуль «Принципы определения местоположения объекта» представляет собой интерактивную модель с возможностью выбора расположения объекта-приемника на поверхности Земли с описанием основных этапов определения местоположения.

Данный модуль является ключевым из всех остальных, так как описывает и демонстрирует самый важный теоретический пункт навигационных систем, а именно трилатерацию, то есть метод определения положения приемника.

Процесс работы с модулем можно описать следующим образом:

1. Пользователь выбирает сначала ССН, на примере которой будет показано определения положения, а затем точку на поверхности Земли, которая является объектом-приемником (рисунок 61).
2. Подсвечиваются ближайшие 4 спутника, в зону видимости сигнала которых входит приемник.
3. Далее последовательно с помощью анимации демонстрируются сферические и конусные сигналы связи между приемником и ближайшим к нему спутником (рисунок 62).
4. Аналогично происходит для последующих трех спутников (в порядке удаленности от приемника).

5. Вместе с этим объясняется принцип определения местоположения (рисунок 63).
6. При появлении последнего четвертого спутника можно сменить систему, выбрать другую точку, а также посмотреть, на что повлияет ошибка хода часов приемника с помощью ползунка (рисунок 64).

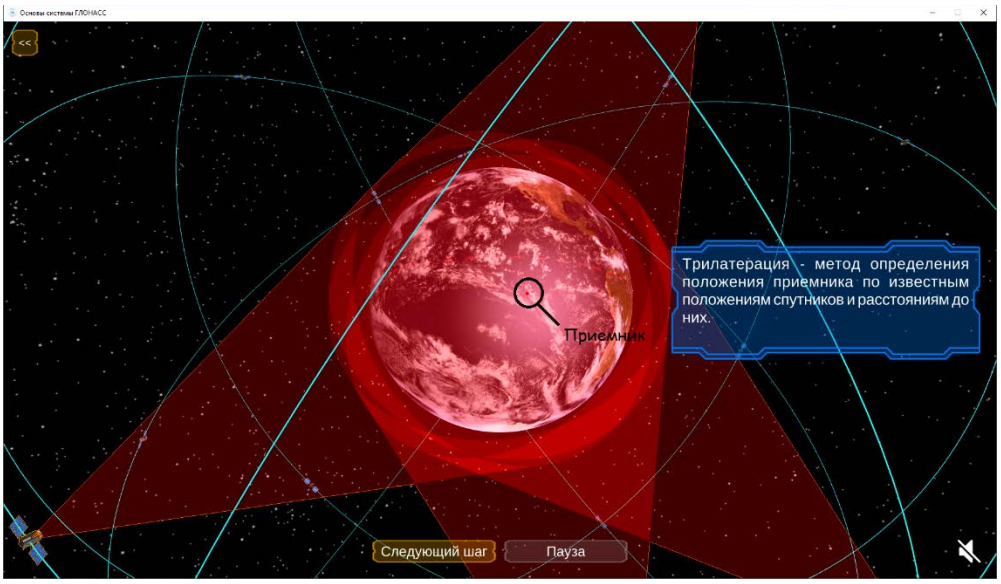


Рисунок 61 – Выбор точки местоположения объекта-приемника

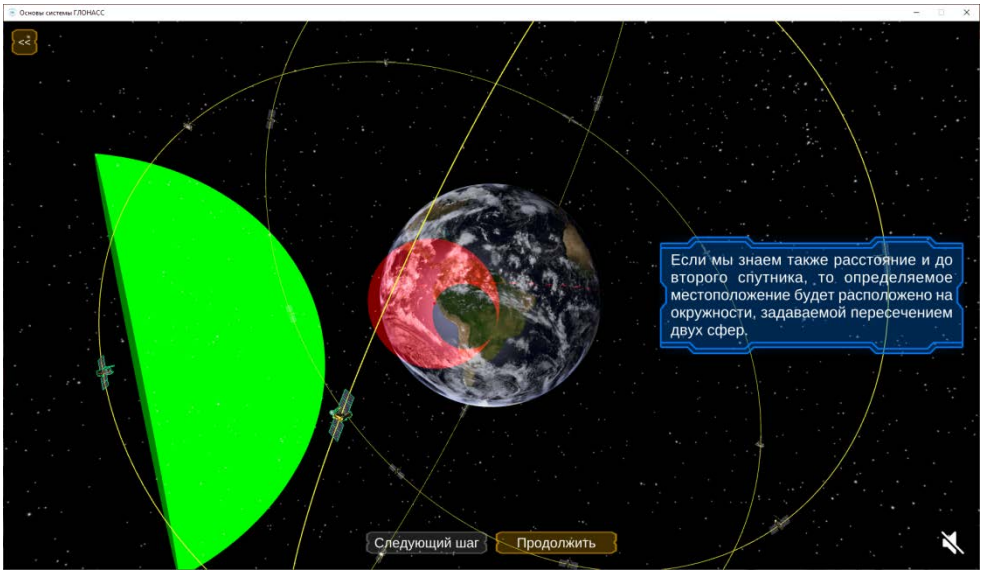


Рисунок 62 – Определение местоположения от второго спутника

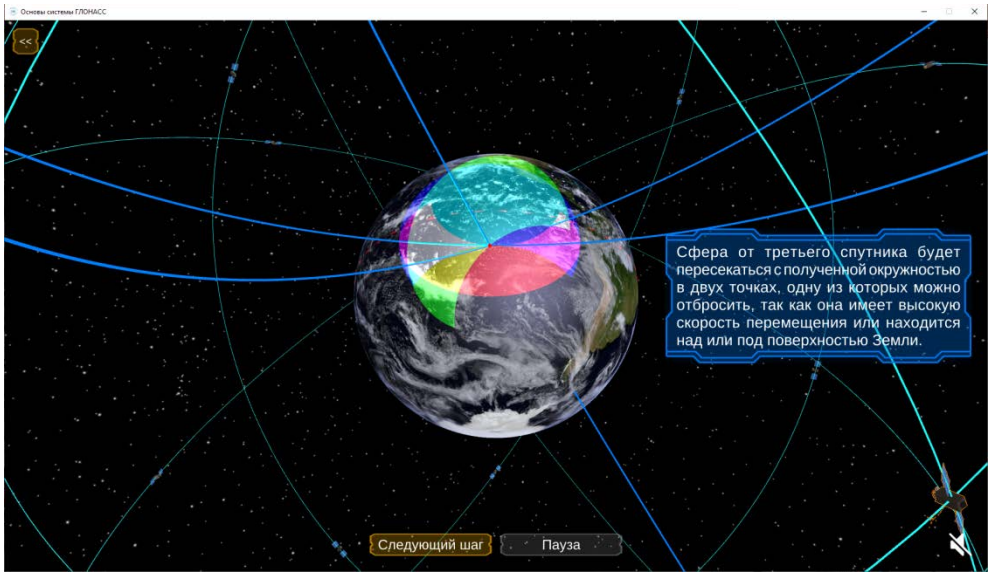


Рисунок 63 – Пересечение трех сфер от трех спутников

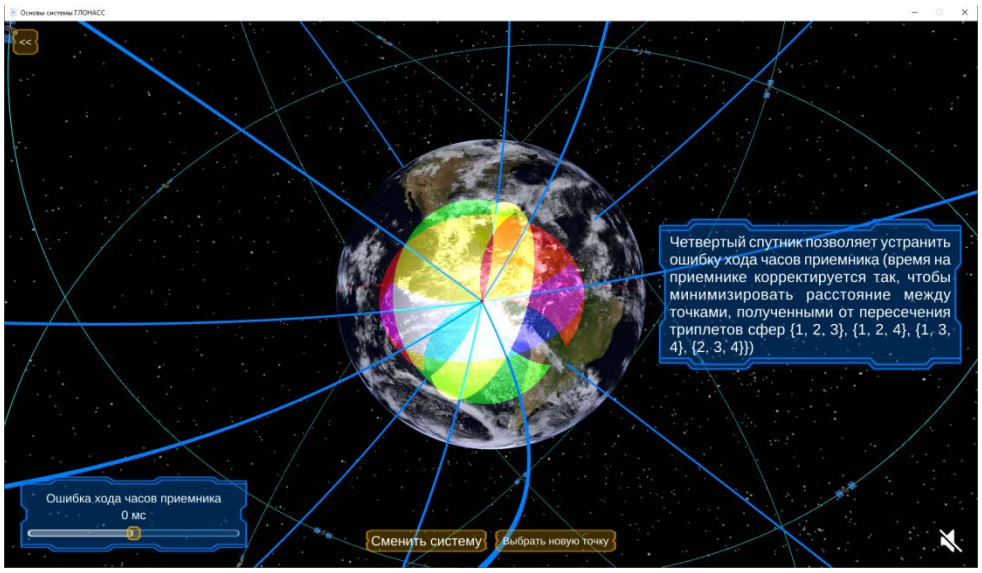


Рисунок 64 – Результат трилатерации

4.2.4. «Факторы, влияющие на точность определения координат»

Модуль «Факторы, влияющие на точность определения координат» представляет собой интерактивную модель с возможностью выбора различных условий приема сигнала от спутников с описанием их влияния на определение местоположения.

Представлены 5 групп факторов в виде подмодулей, на которые можно перейти через меню факторов:

- Дождевые погодные условия. В данном подмодуле демонстрируется влияние разных уровней интенсивности дождя на распространение радиоволн (рисунок 65). Уровень интенсивности можно выбрать с помощью переключателей.
- Снежные погодные условия. Аналогично предыдущему подмодулю (рисунок 66).
- Влияние ионосферы. В третьей группе демонстрируется, как атмосфера Земли значительно замедляет прохождение радиоволн через неё, а тропосфера заметно ослабляет сигнал (рисунок 67).
- Влияние городской среды. В данном подмодуле показано, как различные городские условия влияют на распространение сигнала (отражение волны от твердых поверхностей, влияние других радиоволн на сигнал спутника) (рисунок 68).
- Качество геометрии спутников. В данном группе показаны то, как взаимное расположение спутников относительно приемника влияет на погрешность вычисления местоположения.

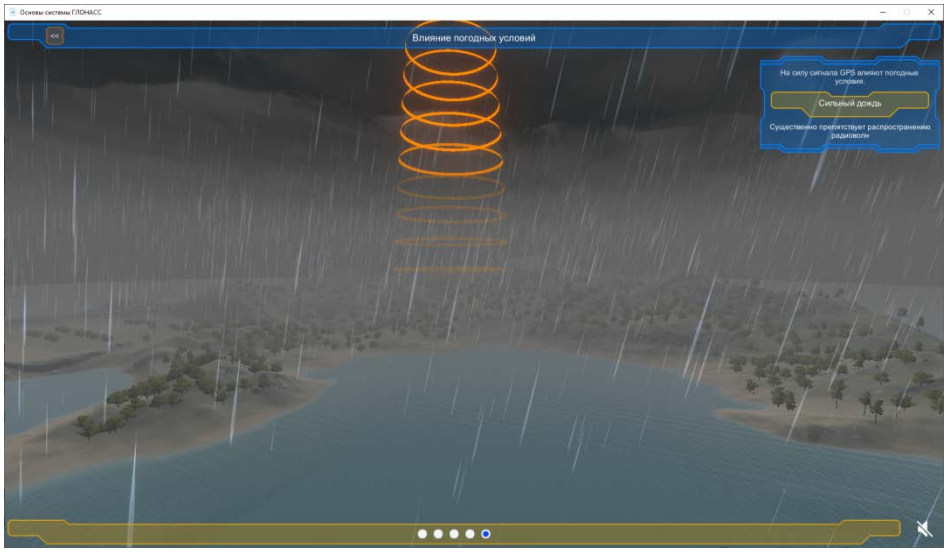


Рисунок 65 – Влияние сильного дождя на распространение сигнала

Рисунок 66 – Влияние сильного снега на распространение сигнала

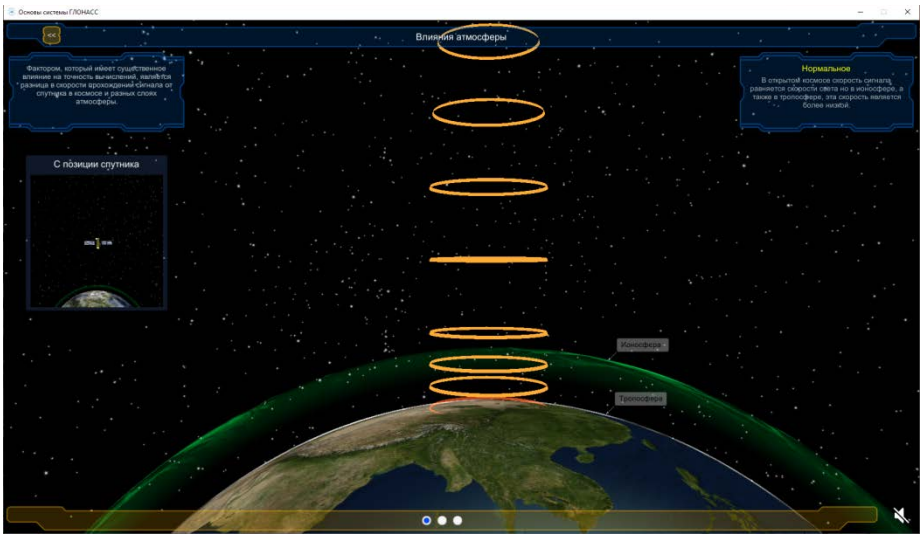


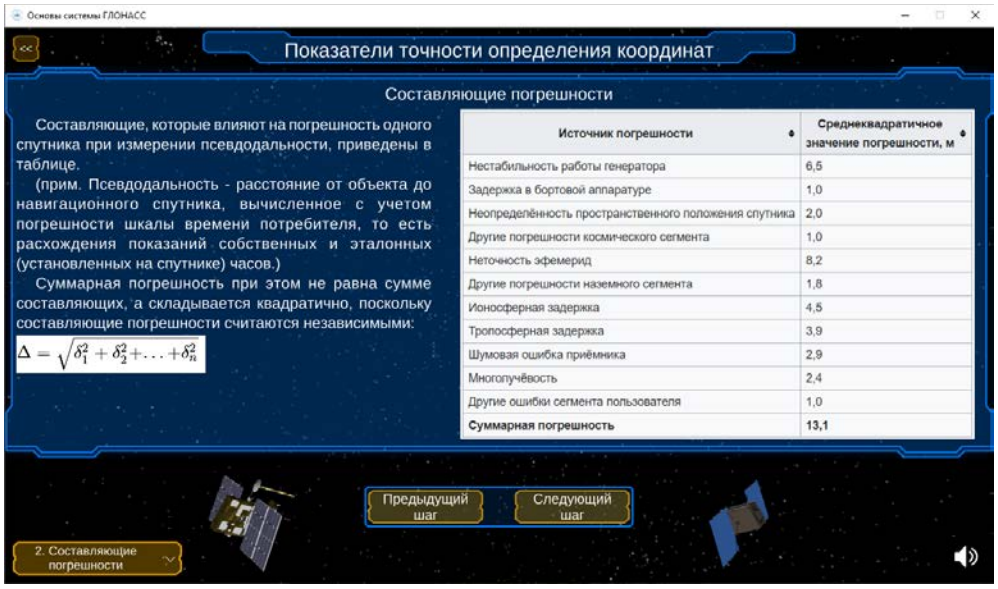
Рисунок 67 – Влияние атмосферы на распространение сигнала

Рисунок 68 – Влияние излучения радиовышки на распространение сигнала

4.2.5. «Показатели точности определения координат»

Модуль «Показатели точности определения координат» включает в себя описание характеристик точности определения местоположения, а также примеры их вычисления.

Данный модуль аналогичен первому модулю. Можно переключаться между главами. В каждой главе описывается своя тема, связанная с показателями точности (рисунок 69 и рисунок 70).



Источник погрешности	Среднеквадратичное значение погрешности, м
Нестабильность работы генератора	6,5
Задержка в бортовой аппаратуре	1,0
Неопределённость пространственного положения спутника	2,0
Другие погрешности космического сегмента	1,0
Неточность эфемерид	8,2
Другие погрешности наземного сегмента	1,8
Ионосферная задержка	4,5
Тропосферная задержка	3,9
Шумовая ошибка приёмника	2,9
Многолучёвость	2,4
Другие ошибки сегмента пользователя	1,0
Суммарная погрешность	**13,1**

Рисунок 69 – Глава 2. Составляющие погрешности

Рисунок 70 – Глава 7. Широкозонные системы дифференциальной коррекции

4.2.6. «Практическое применение ССН»

Модуль «Практическое применение ССН ГЛОНАСС и GPS» включает в себя описание сценариев применения ССН в реальной жизни. Показываются абстрактные модели объектов, для которых применяются ССН (рисунок 71 и рисунок 72).

С помощью переключателей можно посмотреть в каких сферах применяются ССН. Последний переключатель показывает итоговую сводку всех областей применения ССН.

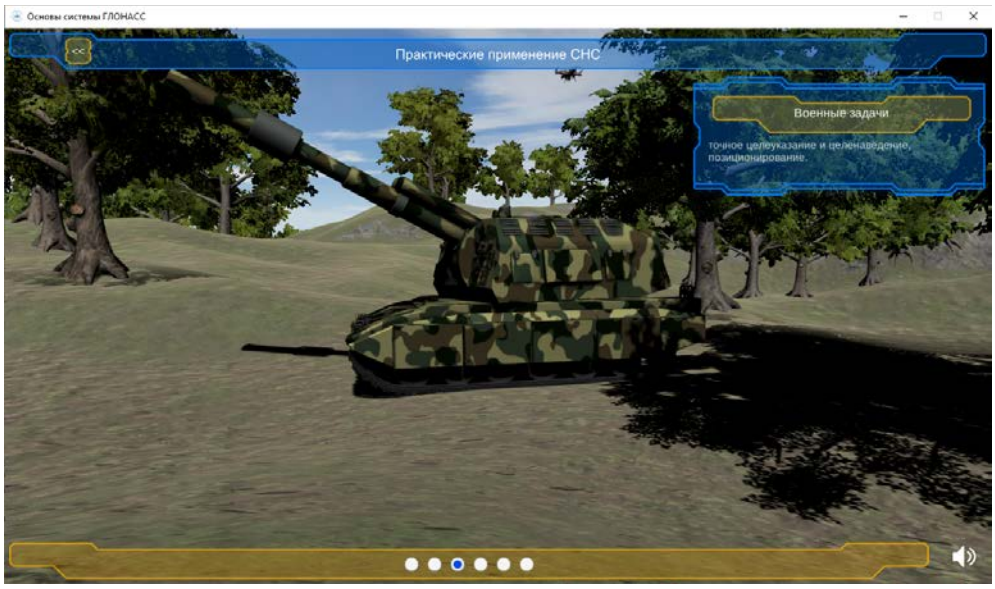


Рисунок 71 – Военные задачи

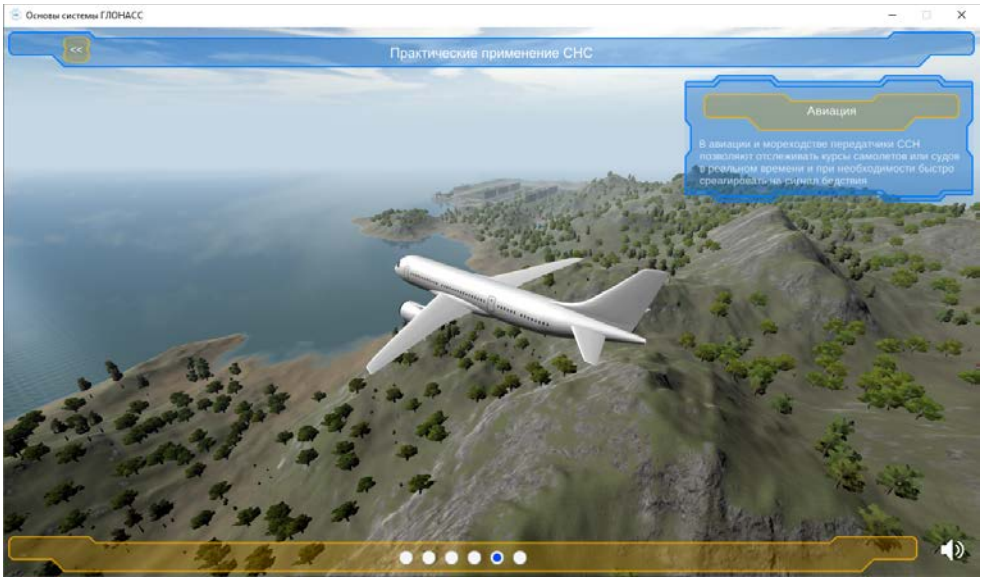


Рисунок 72 – Авиация

4.3. ЛИЦЕНЗИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Одним из требований технического задания является применение защитных средств на готовый программный продукт, а именно электронный ключ фирмы Guardant [13].

Фирма Guardant предоставляет на выбор большое количество различных моделей ключей. Для выбора правильного ключа, удовлетворяющего требованиям заказчика необходимо проанализировать предлагаемые решения Guardant.

Выбор модели ключа основывается, по большей части на особенностях предполагаемой лицензионной политики и способе эксплуатации защищаемого приложения.

В частности, если приложение предназначено для работы в локальной сети, то лицензирование копий приложения можно осуществлять по сети. Это может быть реализовано на основе сетевых ключей – Guardant Sign Net и Guardant Time Net.

При необходимости ограничения работы приложения по времени имеет смысл использовать ключ Guardant Time. С его помощью можно организовать активацию/деактивацию функционала защищаемого приложения по времени.

Потенциально может возникнуть необходимость надежной защиты некоторого алгоритма обработки данных от копирования. Такая возможность предоставляется ключом серии Guardant Code, на основе которого можно создавать уникальные схемы защиты с выполнением загружаемого кода непосредственно внутри ключа [14].

Если рассматривать программу с точки зрения функций представленных ключей (рисунок 73), то для разработанного приложения можно отметить следующие особенности:

– лицензирование происходит локально, в рамках одного компьютера;

– ограничений по времени работы приложений нет;
– не предполагается выполнение дополнительного кода в ключе;

На основании данных особенностей можно сделать вывод, что наиболее подходящим ключом для разработанного приложения является *Guardant Sign*.

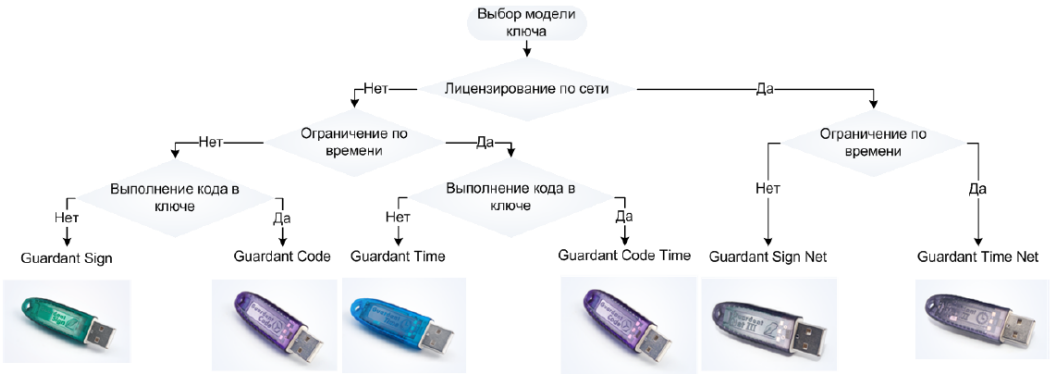


Рисунок 73 – Выбор модели ключа

После выбора электронного ключа необходимо произвести «вакцинацию» приложения. Для этого используется специальная утилита производителя ключа, которая автоматически производит «вакцинацию» программы. Утилита вписывает в тело исполняемого файла модуль – внутреннюю вакцину. В момент запуска приложения данная вакцина обращается к внешней вакцине, расположенной в отдельном файле, которая в свою очередь занимается исполнением защищенного кода приложения и проведением необходимых проверок и преобразований (рисунок 74).

Рисунок 74 – Принцип работы автозащиты

5. ТЕСТИРОВАНИЕ

Благодаря достаточно последовательной работе приложения невозможно вызвать необрабатываемые ошибки. Часть служебных ошибок обрабатывается непосредственно самим движком приложения.

В соответствии с требованиями технического задания программа должна удовлетворять минимальным системным требованиям. Для этого заказчиком был предоставлен компьютер с минимальными системными требованиями под управлением Windows 7. Программа стабильно работала на пресете графических настроек «Ультра» и выдавала за все время работы программы более 60 кадров в секунду.

Было проведено бета-тестирование приложения на компьютерах сотрудников предприятия-заказчика. Не на всех компьютерах программа стабильно исполнялась. Однако такие компьютеры не удовлетворяли минимальным системным требованиям программы. Но на более низких пресетах графических настроек программа работала плавно (более 60 кадров в секунду).

На основании этого на данный момент программное средство полностью отлажено и сдано в эксплуатацию заказчика.

6. ЗАКЛЮЧЕНИЕ

В результате работы поставленная цель была полностью выполнена. Реализован интерактивный программный комплекс «Спутниковые системы навигации» в среде разработки Unity.

Был проведен обзор одного найденного аналога, выделены его преимущества и недостатки, проведен сравнительный анализ аналога с разработанным решением.

Проведено исследование технических специальностей, связанных с радиоэлектроникой; исследование актуальности и необходимости данного проекта в виде электронного опроса среди студентов и выпускников, на основании которых можно сделать вывод, что программный комплекс будет иметь практическую значимость среди студентов, обучающихся на специальностях, имеющих отношение к радиоэлектронике (частично или полностью), и специалистов, изучающих область радиоэлектроники и ведущих разработки в этой области.

Проведен анализ сред разработки, на основании которого наилучшим инструментом для разработки была определена Unity.

Были определены требования заказчика в виде технического задания, на основании которых был разработан программный комплекс.

Спроектирована и разработана архитектура проекта, на основании которой получен граф структуры сцен проекта. Описаны и реализованы алгоритмы решения задач каждой сцены.

После разработки проект был отлажен и собран. Описан сценарий работы приложения по каждому модулю. Проведен комплекс мер по защите программного обеспечения в виде «прошивки» электронным ключом. Затем были проведены тесты, которые подтвердили положительный результат работы программы на различных компьютерах с различными ОС Windows при различных настройках графики.

Программный комплекс имеет потенциал в расширении, модифицировании и улучшении его работы. Например, можно создать модуль в виде «песочницы», в которой пользователь может сам проектировать различные ситуации приема сигнала спутников. Можно создать сложную и большую математическую модель распространения сигнала на основании реальных математических и физических моделей и воспроизвести крайне реалистичную симуляцию распространения радиоволны со спутника.

Всего в данном проекте насчитывается 336 скриптов. Разработанное приложение занимает 1,25 Гб на дисковом пространстве.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Класс навигационного оборудования "Глонасс". – https://zarnitza.ru/catalog/shkolnaya-podgotovka-i-detskiy-sad/shkola/klassy-po-drugim-obrazovatelnym-disciplinam/klass-navigatsionnogo-oborudovaniya-glonass/?sphrase_id=152584. Дата обращения: 11.02.2019.
2. ВШ ЭКН. – https://eecs.susu.ru/ru/entrant/programs. Дата обращения: 13.02.2019.
3. Обзор самых популярных движков для разработки игр. – https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e. Дата обращения: 19.02.2019.
4. Developing MOBA games using the Unity game engine. – https://ieeexplore.ieee.org/abstract/document/7973661. Дата обращения: 19.02.2019.
5. Simulating visual impairments using the Unreal Engine 3 game engine. – https://ieeexplore.ieee.org/abstract/document/6165430. Дата обращения: 19.02.2019.
6. Implementing a low-cost CAVE system using the CryEngine2. – https://www.sciencedirect.com/science/article/pii/S1875952110000108. Дата обращения: 19.02.2019.
7. Game Engine Solutions. – https://www.intechopen.com/books/simulation-and-gaming/game-engine-solutions. Дата обращения: 19.02.2019.
8. The Effectiveness of Usability Evaluation Methods: Determining the Appropriate Criteria. – https://journals.sagepub.com/doi/abs/10.1177/154193129904302007. Дата обращения: 19.02.2019.
9. Scenes. – https://docs.unity3d.com/ru/current/Manual/CreatingScenes.html. Дата обращения: 07.03.2019.

10. GameObject. – https://docs.unity3d.com/Manual/class-GameObject.html. Дата обращения: 07.03.2019.
11. Using Components. – https://docs.unity3d.com/Manual/UsingComponents.html. Дата обращения: 07.03.2019.
12. Glossary. – https://docs.unity3d.com/Manual/Glossary.html. Дата обращения: 09.03.2019.
13. Электронные ключи Guardant. – https://www.guardant.ru. Дата обращения: 26.04.2019.
14. Обучающие материалы. – https://www.guardant.ru/support/lessons. Дата обращения: 26.04.2019.
15. НПП «Учтех-профи». Теоретические пособие. Учебно-лабораторный стенд «Системы спутниковой навигации». – 2014 – 98 с.
16. Руководство Unity. – https://docs.unity3d.com/Manual/index.html. Дата обращения: 07.03.2019.
17. В.В. Конин. Национальный авиационный университет. Системы спутниковой радионавигации / В.В. Конин, В.П. Харченко. – Киев: Холтех, 2010. – 520 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД КЛЮЧЕВЫХ СКРИПТОВ ПРОГРАММЫ

1. Исходный код скрипта MainMenu.cs. Работа сцены «Главное меню».

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

public class MainMenu : MonoBehaviour
{
    public void SceneLoad(string sceneName)
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene(sceneName);
    }
    public void ApplicationQuit()
    {
        Application.Quit();
    }
    private void Update()
    {
        if (Input.GetKey("escape"))
            Application.Quit();
    }
}
```

2. Исходный код скрипта History.cs. Работа сцены «История ССН».

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

public class History : MonoBehaviour {

    [SerializeField] GameObject[] panelsClear;
    [SerializeField] GameObject[] chronoOrbits;
    [SerializeField] GameObject panelMain;
    cakeslice.Outline[] outlines;
    private void OnMouseOver()
    {
        SetOutlines(true);
    }
    private void OnMouseExit()
    {
        SetOutlines(false);
    }
    private void OnMouseUp()
    {
        PanelMain(panelMain);
    }

    private void SetOutlines(bool on)
    {
        for (int i = 0; i < outlines.Length; i++)
        {
            outlines[i].enabled = on;
        }
```

```
    }
    public void PanelMain (GameObject panelMain)
    {
        for (int i = 0; i < panelsClear.Length; i++)
        {
            panelsClear[i].SetActive(false);
        }
        panelMain.SetActive(true);

    }

    public void MainMenu()
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene("Main menu");
    }

    public void EnableChronoOrbit(GameObject orbit)
    {
        for (int i = 0; i < chronoOrbits.Length; i++)
        {
            chronoOrbits[i].SetActive(false);
        }
        orbit.SetActive(true);
    }

    public void ClearOrbits()
    {
        for (int i = 0; i < chronoOrbits.Length; i++)
        {
            chronoOrbits[i].SetActive(false);
        }
    }
    void Start()
    {
        outlines = GetComponentsInChildren<cakeslice.Outline>();
        SetOutlines(false);
    }
}
```

3. Исходный код скрипта SatellitesRotation.cs. Синхронизированное вращение спутников вокруг Земли.

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

namespace StructAndDiffs
{
  public class SatellitesRotation : MonoBehaviour
  {
    [SerializeField] float satellitesRotationSpeed;

    [SerializeField] Transform[] glonassSatellites;
    [SerializeField] Transform[] gpsSatellites;

    float satRotationAngle = 0f;
    bool rotating = true;



    private static SatellitesRotation instance = null;
```

```
    public static SatellitesRotation Instance { get { return instance; } }

    public bool Rotating { get { return rotating; } set { rotating = value; } }

    private void Awake()
    {
      instance = this;
    }

    private void OnDestroy()
    {
      instance = null;
    }

    void UpdateRotationSatelliteValue()
    {
      satRotationAngle += Time.deltaTime * satellitesRotationSpeed;
      if (satRotationAngle > 360f) satRotationAngle -= 360f;
    }

    void UpdateGlonassSatPos()
    {
      for (int sat = 0; sat < 8; sat++)
      {
        Quaternion localSatRotation = Quaternion.Euler(0f, satRotationAngle + sat * 45f,
0f);
        glonassSatellites[sat + 0].localRotation = localSatRotation;
        glonassSatellites[sat + 8].localRotation = localSatRotation;
        glonassSatellites[sat + 16].localRotation = localSatRotation;
      }
    }

    void UpdateGpsSatPos()
    {
      for (int sat = 0; sat < 4; sat++)
      {
        Quaternion localSatRotation = Quaternion.Euler(0f, satRotationAngle + sat * 90f,
0f);
        gpsSatellites[sat + 0].localRotation = localSatRotation;
        gpsSatellites[sat + 4].localRotation = localSatRotation;
        gpsSatellites[sat + 8].localRotation = localSatRotation;
        gpsSatellites[sat + 12].localRotation = localSatRotation;
        gpsSatellites[sat + 16].localRotation = localSatRotation;
        gpsSatellites[sat + 20].localRotation = localSatRotation;
      }
    }

    void Update()
    {
      if (rotating)
      {
        UpdateRotationSatelliteValue();
        UpdateGlonassSatPos();
        UpdateGpsSatPos();
      }
    }
  }
}
```

4. Исходный код скрипта SceneElementsSwitcher.cs. Переключение наборов объектов на сцене.

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

namespace StructAndDiffs
{
  public class SceneElementsSwitcher : MonoBehaviour
  {
    [SerializeField] GameObject[] objToDisable;

    [SerializeField] GameObject glonassSatellites;
    [SerializeField] GameObject gpsSatellites;

    [SerializeField] GameObject glonassLandObjects;
    [SerializeField] GameObject gpsLandObjects;

    [SerializeField] GameObject earth;
    [SerializeField] GameObject satelliteFormationSignals;
    [SerializeField] GameObject comparisonObject;

    private static SceneElementsSwitcher instance = null;
    public static SceneElementsSwitcher Instance { get { return instance; } }

    private void Awake()
    {
      instance = this;
    }

    private void OnDestroy()
    {
      instance = null;
    }

    public void HideAll()
    {
      for (int i = 0; i < objToDisable.Length; i++)
      {
        objToDisable[i].SetActive(false);
      }
    }

    public void ShowGlonassSatellites()
    {
      glonassSatellites.SetActive(true);
    }

    public void ShowGpsSatellites()
    {
      gpsSatellites.SetActive(true);
    }

    public void ShowEarth()
    {
      earth.SetActive(true);
    }

    public void ShowGlonassLandObjects()
    {
```

```
      glonassLandObjects.SetActive(true);
    }

    public void ShowGpsLandObjects()
    {
      gpsLandObjects.SetActive(true);
    }

    public void ShowFormingSignalsSatellite()
    {
      satelliteFormationSignals.SetActive(true);
    }

    public void ShowComarisonObject()
    {
      comparisonObject.SetActive(true);
    }
  }
}
```

5. Исходный код скрипта UIElementsSwitcher.cs. Переключение наборов UI объектов на сцене.

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

namespace StructAndDiffs
{
  public class UIElementsSwitcher : MonoBehaviour
  {
    [SerializeField] GameObject[] objToDisable;
    [SerializeField] GameObject backForwardControlPanel;
    [SerializeField] Button backButton;
    [SerializeField] Button forwardButton;
    [SerializeField] GameObject descrPanel;
    [SerializeField] TMPro.TextMeshProUGUI descrText;

    private static UIElementsSwitcher instance = null;
    public static UIElementsSwitcher Instance { get { return instance; } }

    private void Awake()
    {
      instance = this;
    }

    private void OnDestroy()
    {
      instance = null;
    }

    public void HideAll()
    {
      for (int i = 0; i < objToDisable.Length; i++)
      {
        objToDisable[i].SetActive(false);
      }
    }

    public void ShowBackForward(int currentStep, int maxSteps)
```

```
    {
      backForwardControlPanel.SetActive(true);
      backButton.interactable = !(currentStep == 0);
      forwardButton.interactable = (currentStep < maxSteps - 1);
    }

    public void ShowDescription(string text)
    {
      descrPanel.SetActive(true);
      descrText.text = text;
    }

  }
}
```

6. Исходный код скрипта DiffsFlowController.cs. Управление работой сцены «Структура и отличия ССН».

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

namespace StructAndDiffs
{
  public class DiffsFlowController : MonoBehaviour
  {
    public enum DiffSubmodules
    {
      None,
      Space,
      Control,
      Consumers,
      Signals,
      Diffs,
      ProsCons,
      Prospects
    }

    [SerializeField] Transform satelliteGlonass;
    [SerializeField] Transform satelliteGps;
    [SerializeField] Transform targetFormingSignals;

    [SerializeField] Transform camPivot;

    [Multiline]
    [SerializeField] string startDescription;

    [Multiline]
    [SerializeField] string[] spaceSegTexts;

    [Multiline]
    [SerializeField] string[] controlSegTexts;

    [Multiline]
    [SerializeField] string[] consumerSegTexts;

    [Multiline]
    [SerializeField] string[] signalsTexts;

    [Multiline]
```

```
[SerializeField] string[] diffsTexts;

[Multiline]
[SerializeField] string[] prosConsTexts;

[SerializeField] Transform diffsSubTarget = null;

DiffsCamControl camControl;

int currentStep = 0;
int maxSteps = 1;

DiffSubmodules currentSub = DiffSubmodules.None;

delegate void StepAction();
Dictionary<DiffSubmodules, StepAction> stepActions;

delegate void StepChange();
event StepChange onStepChange;

void Start()
{
  stepActions = new Dictionary<DiffSubmodules, StepAction>();
  stepActions.Add(DiffSubmodules.Space, ChangeStepSpace);
  stepActions.Add(DiffSubmodules.Control, ChangeStepControl);
  stepActions.Add(DiffSubmodules.Consumers, ChangeStepConsumers);
  stepActions.Add(DiffSubmodules.Signals, ChangeStepSignals);
  stepActions.Add(DiffSubmodules.ProsCons, ChangeStepProsCons);
  stepActions.Add(DiffSubmodules.Diffs, ChangeStepDiffs);

  camControl = FindObjectOfType<DiffsCamControl>();

  SceneElementsSwitcher.Instance.HideAll();
  SceneElementsSwitcher.Instance.ShowEarth();

  UIElementsSwitcher.Instance.HideAll();

  SunRotationControl.Instance.SetRotation(Quaternion.identity);
  SunRotationControl.Instance.SetTargetTransform(null);
}

public void StepForward()
{
  if (onStepChange != null) onStepChange();
  currentStep = Mathf.Clamp(currentStep + 1, 0, maxSteps - 1);
  if (stepActions.ContainsKey(currentSub)) stepActions[currentSub]();
}

public void StepBack()
{
  if (onStepChange != null) onStepChange();
  currentStep = Mathf.Clamp(currentStep - 1, 0, maxSteps - 1);
  if (stepActions.ContainsKey(currentSub)) stepActions[currentSub]();
}

public void StartSpaceSegment()
{
  if (onStepChange != null) onStepChange();
  currentSub = DiffSubmodules.Space;
  maxSteps = spaceSegTexts.Length;
  currentStep = 0;
  ChangeStepSpace();
}
```

```
public void StartControlSegment()
{
  if (onStepChange != null) onStepChange();
  currentSub = DiffSubmodules.Control;
  maxSteps = controlSegTexts.Length;
  currentStep = 0;
  ChangeStepControl();
}

public void StartConsumersSegment()
{
  if (onStepChange != null) onStepChange();
  currentSub = DiffSubmodules.Consumers;
  maxSteps = consumerSegTexts.Length;
  currentStep = 0;
  ChangeStepConsumers();
}

public void StartSignalsSegment()
{
  if (onStepChange != null) onStepChange();
  currentSub = DiffSubmodules.Signals;
  maxSteps = signalsTexts.Length;
  currentStep = 0;
  ChangeStepSignals();
}

public void StartPorosConsSegment()
{
  if (onStepChange != null) onStepChange();
  currentSub = DiffSubmodules.ProsCons;
  maxSteps = prosConsTexts.Length;
  currentStep = 0;
  ChangeStepProsCons();
}

public void StartDiffsSegment()
{
  if (onStepChange != null) onStepChange();
  currentSub = DiffSubmodules.Diffs;
  maxSteps = diffsTexts.Length;
  currentStep = 0;
  ChangeStepDiffs();
}

private void ChangeStepDiffs()
{
  const float targetDist = 80f;

  UIElementsSwitcher.Instance.HideAll();
  UIElementsSwitcher.Instance.ShowDescription(diffsTexts[currentStep]);

  SceneElementsSwitcher.Instance.HideAll();

  SunRotationControl.Instance.SetTargetTransform(null);
  SunRotationControl.Instance.SetRotation(Quaternion.identity);

  SatellitesRotation.Instance.Rotating = true;

  GroundWaves.Instance.StopSignalsAnimation();

  SceneElementsSwitcher.Instance.ShowEarth();
```

```
      SceneElementsSwitcher.Instance.ShowComarisonObject();

      CamTargeting.Instance.StartTargeting(diffsSubTarget, camControl.BaseTr,
targetDist);
      onStepChange += OnStepChangeCamStopTargeting;
    }
    private void ChangeStepProsCons()
    {
      UIElementsSwitcher.Instance.HideAll();
      UIElementsSwitcher.Instance.ShowDescription(prosConsTexts[currentStep]);
      UIElementsSwitcher.Instance.ShowBackForward(currentStep, maxSteps);

      SceneElementsSwitcher.Instance.HideAll();

      SunRotationControl.Instance.SetTargetTransform(null);
      SunRotationControl.Instance.SetRotation(Quaternion.identity);

      SatellitesRotation.Instance.Rotating = true;

      GroundWaves.Instance.StopSignalsAnimation();

      SceneElementsSwitcher.Instance.ShowEarth();
    }

    void ChangeStepSignals()
    {
      const float targetingDist = 65f;

      UIElementsSwitcher.Instance.HideAll();
      UIElementsSwitcher.Instance.ShowDescription(signalsTexts[currentStep]);
      UIElementsSwitcher.Instance.ShowBackForward(currentStep, maxSteps);

      SceneElementsSwitcher.Instance.HideAll();

      SunRotationControl.Instance.SetTargetTransform(null);
      SunRotationControl.Instance.SetRotation(Quaternion.identity);

      SatellitesRotation.Instance.Rotating = true;

      GroundWaves.Instance.StopSignalsAnimation();

      SceneElementsSwitcher.Instance.ShowEarth();
      SceneElementsSwitcher.Instance.ShowFormingSignalsSatellite();

      CamTargeting.Instance.StartTargeting(targetFormingSignals, camControl.BaseTr,
targetingDist);
      onStepChange += OnStepChangeCamStopTargeting;

      switch (currentStep)
      {
      case 0:
        SignalFormingController.Instance.SetNavMessageState(false);
        SignalFormingController.Instance.SetPseudoRandomCodeState(false);
        break;
      case 1:
        SignalFormingController.Instance.SetNavMessageState(false);
        SignalFormingController.Instance.SetPseudoRandomCodeState(true);
        break;
      case 2:
        SignalFormingController.Instance.SetNavMessageState(true);
        SignalFormingController.Instance.SetPseudoRandomCodeState(false);
        break;
```

```
    default:
      break;
    }
}

void ChangeStepConsumers()
{
    UIElementsSwitcher.Instance.HideAll();
    UIElementsSwitcher.Instance.ShowDescription(consumerSegTexts[currentStep]);

    SceneElementsSwitcher.Instance.HideAll();

    SunRotationControl.Instance.SetTargetTransform(null);
    SunRotationControl.Instance.SetRotation(Quaternion.identity);

    SatellitesRotation.Instance.Rotating = true;

    GroundWaves.Instance.StopSignalsAnimation();

    SceneElementsSwitcher.Instance.ShowEarth();
}

void ChangeStepControl()
{
    UIElementsSwitcher.Instance.HideAll();
    UIElementsSwitcher.Instance.ShowBackForward(currentStep, maxSteps);
    UIElementsSwitcher.Instance.ShowDescription(controlSegTexts[currentStep]);

    SceneElementsSwitcher.Instance.HideAll();

    SunRotationControl.Instance.SetTargetTransform(camPivot);

    SatellitesRotation.Instance.Rotating = true;

    GroundWaves.Instance.StopSignalsAnimation();

    switch (currentStep)
    {
    case 0:
      SceneElementsSwitcher.Instance.ShowEarth();
      SceneElementsSwitcher.Instance.ShowGlonassLandObjects();
      camControl.MoveCamToPosition(-151f, 33f, 80f, 1f);
      break;
    case 1:
      SceneElementsSwitcher.Instance.ShowEarth();
      SceneElementsSwitcher.Instance.ShowGpsLandObjects();
      camControl.MoveCamToPosition(-8.6f, 21f, 80f, 1f);
      break;
    case 2:
      SceneElementsSwitcher.Instance.ShowEarth();
      SceneElementsSwitcher.Instance.ShowGpsLandObjects();
      SceneElementsSwitcher.Instance.ShowGpsSatellites();
      camControl.MoveCamToPosition(-8.6f, 21f, 150f, 1f);
      SatellitesRotation.Instance.Rotating = false;
      GroundWaves.Instance.StartSignalsAnimation();
      break;
    default:
      break;
    }
}

void ChangeStepSpace()
{
```

```
      const float targetingGlonassDist = 65f;
      const float targetingGpsDist = 68f;

      UIElementsSwitcher.Instance.HideAll();
      UIElementsSwitcher.Instance.ShowBackForward(currentStep, maxSteps);
      UIElementsSwitcher.Instance.ShowDescription(spaceSegTexts[currentStep]);

      SceneElementsSwitcher.Instance.HideAll();

      SunRotationControl.Instance.SetTargetTransform(null);
      SunRotationControl.Instance.SetRotation(Quaternion.identity);

      GroundWaves.Instance.StopSignalsAnimation();

      switch (currentStep)
      {
      case 0:
        SceneElementsSwitcher.Instance.ShowEarth();
        SceneElementsSwitcher.Instance.ShowGlonassSatellites();
        CamTargeting.Instance.StartTargeting(satelliteGlonass, camControl.BaseTr,
targetingGlonassDist);
        onStepChange += OnStepChangeCamStopTargeting;
        break;
      case 1:
        SceneElementsSwitcher.Instance.ShowEarth();
        SceneElementsSwitcher.Instance.ShowGlonassSatellites();
        camControl.MoveCamToPosition(-151f, 33f, 150f, 1f);
        break;
      case 2:
        SceneElementsSwitcher.Instance.ShowEarth();
        SceneElementsSwitcher.Instance.ShowGpsSatellites();
        CamTargeting.Instance.StartTargeting(satelliteGps, camControl.BaseTr,
targetingGpsDist);
        onStepChange += OnStepChangeCamStopTargeting;
        break;
      case 3:
        SceneElementsSwitcher.Instance.ShowEarth();
        SceneElementsSwitcher.Instance.ShowGpsSatellites();
        camControl.MoveCamToPosition(-8.6f, 21f, 150f, 1f);
        break;
      default:
        break;
      }

      SceneElementsSwitcher.Instance.ShowEarth();
    }

    private void OnStepChangeCamStopTargeting()
    {
      CamTargeting.Instance.StopTargeting();
      onStepChange -= OnStepChangeCamStopTargeting;
    }
  }
}
```

7. Исходный код скрипта CamTargeting.cs. Позиционирование и вращение камеры относительно целевой точки.

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

namespace StructAndDiffs
{
  public class CamTargeting : MonoBehaviour
  {
    DiffsCamControl camControl;
    Transform target;
    Transform baseRotTr;
    float camDist;

    private static CamTargeting instance = null;
    public static CamTargeting Instance { get { return instance; } }

    void Start()
    {
      instance = this;
      camControl = FindObjectOfType<DiffsCamControl>();
    }

    private void OnDestroy()
    {
      instance = null;
    }

    public void StartTargeting(Transform _target, Transform _baseRotationTransform, float
_camDist)
    {
      camDist = _camDist;
      target = _target;
      baseRotTr = _baseRotationTransform;
      StartCoroutine(UpdateCamMovePoint());
    }

    IEnumerator UpdateCamMovePoint()
    {
      while (true)
      {
        Vector2 targetAngle = CalculateAngles();
        camControl.MoveCamToPosition(targetAngle.y, targetAngle.x, camDist, 0.2f,
AnimationCurve.Linear(0, 0, 1, 1));
        yield return null;
      }
    }

    Vector2 CalculateAngles()
    {
      Vector3 pos = baseRotTr.worldToLocalMatrix.MultiplyPoint(target.position);
      float angleX = Mathf.Rad2Deg * Mathf.Asin(pos.y / pos.magnitude);
      float angleY = Mathf.Rad2Deg * Mathf.Atan2(-pos.x, -pos.z);
      return new Vector2(angleX, angleY);
    }

    public void StopTargeting()
    {
      StopAllCoroutines();
```

```
    }
  }
}
```

8. Исходный код скрипта SunRotationControl.cs. Вращение источника освещения (Солнца).

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

namespace StructAndDiffs
{
  public class SunRotationControl : MonoBehaviour
  {
    [SerializeField] Transform sunTr;

    Transform target = null;

    private static SunRotationControl instance = null;
    public static SunRotationControl Instance { get { return instance; } }

    void Start()
    {
      instance = this;
    }

    private void OnDestroy()
    {
      instance = null;
    }

    public void SetRotation(Quaternion rotation)
    {
      sunTr.rotation = rotation;
    }

    public void SetTargetTransform(Transform _target)
    {
      target = _target;
    }

    private void Update()
    {
      if (target != null)
      {
        sunTr.rotation = target.rotation;
      }
    }
  }
}
```

9. Исходный код скрипта GroundWaves.cs. Работа системы частиц направленного сигнала со спутника в виде потокового луча.

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

namespace StructAndDiffs
{
  public class GroundWaves : MonoBehaviour
  {
    [SerializeField] Transform monitorStation, mainControlStation, groundAntenna;
    [SerializeField] Vector3 groundOffset;
    [SerializeField] Vector3 satelliteOffset;

    [SerializeField] float linearSignalSpeed = 10f;
    [SerializeField] float linearWavesDiff = 2f;

    const float linearToAngularMult = 2.85f;

    float angularSignalSpeed = 28.5f;
    float angularWavesDiff = 5.7f;

    [SerializeField] GameObject wavePrefab = null;
    [SerializeField] int numWaves = 5;

    [SerializeField] Transform[] gpsSatellites = null;

    Transform sat = null;

    Vector3[] wavesNavPoints;
    GameObject[] waves = null;
    int wavesStep = 0;

    Vector3 axis;
    float angle;

    private static GroundWaves instance = null;
    public static GroundWaves Instance { get { return instance; } }

    private void Awake()
    {
      instance = this;
      angularSignalSpeed = linearToAngularMult * linearSignalSpeed;
      angularWavesDiff = linearToAngularMult * linearWavesDiff;
    }

    private void OnDestroy()
    {
      instance = null;
    }

    public void StartSignalsAnimation()
    {
      StopSignalsAnimation();
      wavesStep = 0;
      sat = gpsSatellites.OrderBy(sat => Vector3.SqrMagnitude(sat.position -
monitorStation.transform.position)).First();
      wavesNavPoints = new Vector3[4];
      wavesNavPoints[0] = sat.position + sat.rotation * satelliteOffset;
```

```
      wavesNavPoints[1] = monitorStation.position + monitorStation.rotation *
groundOffset;
      wavesNavPoints[2] = mainControlStation.position + mainControlStation.rotation *
groundOffset;
      wavesNavPoints[3] = groundAntenna.position + groundAntenna.rotation * groundOffset;

      waves = new GameObject[numWaves];

      for (int i = 0; i < numWaves; i++)
      {
        waves[i] = Instantiate(wavePrefab);
        waves[i].SetActive(false);
      }
      StartCoroutine(DelayedWaves());
    }

    IEnumerator DelayedWaves()
    {
      const float startDelay = 2f;
      yield return new WaitForSeconds(startDelay);
      StartCoroutine(WavesMover());
    }

    IEnumerator WavesMover()
    {
      // satellite - station (1st step)-------------------------------------
      Vector3 dir1 = wavesNavPoints[1] - wavesNavPoints[0];
      float dist1 = dir1.magnitude;
      float delta1 = linearWavesDiff * (numWaves - 1);
      float step1Time = (dist1 + delta1) / linearSignalSpeed;
      float step1Timer = 0f;
      Quaternion rot1 = Quaternion.LookRotation(dir1);
      for (int i = 0; i < numWaves; i++) waves[i].transform.rotation = rot1;

      while (step1Timer < step1Time)
      {
        float progr = step1Timer / step1Time;

        for (int i = 0; i < numWaves; i++)
        {
          float pos = (progr * (dist1 + delta1) - linearWavesDiff * i)/ dist1;
          bool waveActive = (pos >= 0f) && (pos < 1f);
          waves[i].transform.position = pos * wavesNavPoints[1] + (1f - pos) *
wavesNavPoints[0];
          if (waves[i].activeSelf != waveActive) waves[i].SetActive(waveActive);
        }
        step1Timer += Time.deltaTime;
        yield return null;
      }

      // staition - main station (2nd step)---------------------------------
      SetPoints(wavesNavPoints[1], wavesNavPoints[2], Vector3.zero);
      Vector3 rad2 = wavesNavPoints[1];
      float dist2 = Mathf.Abs(angle);
      float delta2 = angularWavesDiff * (numWaves - 1);
      float step2Time = (dist2 + delta2) / angularSignalSpeed;
      float step2Timer = 0f;

      while (step2Timer < step2Time)
      {
        float progr = step2Timer / step2Time;

        for (int i = 0; i < numWaves; i++)
```

```
        {
          float pos = (progr * (dist2 + delta2) - angularWavesDiff * i) / dist2;
          bool waveActive = (pos >= 0f) && (pos < 1f);
          Vector3 newPos = Quaternion.AngleAxis(pos * angle, axis) * rad2;
          waves[i].transform.position = newPos;
          waves[i].transform.rotation = Quaternion.LookRotation(Vector3.Cross(axis,
newPos));
          if (waves[i].activeSelf != waveActive) waves[i].SetActive(waveActive);
        }
        step2Timer += Time.deltaTime;
        yield return null;
      }

      // main staition - ground antenna (3rd step)--------------------------
      SetPoints(wavesNavPoints[2], wavesNavPoints[3], Vector3.zero);
      Vector3 rad3 = wavesNavPoints[2];
      float dist3 = Mathf.Abs(angle);
      float delta3 = angularWavesDiff * (numWaves - 1);
      float step3Time = (dist3 + delta3) / angularSignalSpeed;
      float step3Timer = 0f;

      while (step3Timer < step3Time)
      {
        float progr = step3Timer / step3Time;

        for (int i = 0; i < numWaves; i++)
        {
          float pos = (progr * (dist3 + delta3) - angularWavesDiff * i) / dist3;
          bool waveActive = (pos >= 0f) && (pos < 1f);
          Vector3 newPos = Quaternion.AngleAxis(pos * angle, axis) * rad3;
          waves[i].transform.position = newPos;
          waves[i].transform.rotation = Quaternion.LookRotation(Vector3.Cross(axis,
newPos));
          if (waves[i].activeSelf != waveActive) waves[i].SetActive(waveActive);
        }
        step3Timer += Time.deltaTime;
        yield return null;
      }

      // ground antenna - satellite (4th step)------------------------------
      Vector3 dir4 = wavesNavPoints[0] - wavesNavPoints[3];
      float dist4 = dir1.magnitude;
      float delta4 = linearWavesDiff * (numWaves - 1);
      float step4Time = (dist4 + delta4) / linearSignalSpeed;
      float step4Timer = 0f;
      Quaternion rot4 = Quaternion.LookRotation(dir4);
      for (int i = 0; i < numWaves; i++) waves[i].transform.rotation = rot4;

      while (step4Timer < step4Time)
      {
        float progr = step4Timer / step4Time;

        for (int i = 0; i < numWaves; i++)
        {
          float pos = (progr * (dist4 + delta4) - linearWavesDiff * i) / dist4;
          bool waveActive = (pos >= 0f) && (pos < 1f);
          waves[i].transform.position = pos * wavesNavPoints[0] + (1f - pos) *
wavesNavPoints[3];
          if (waves[i].activeSelf != waveActive) waves[i].SetActive(waveActive);
        }
        step4Timer += Time.deltaTime;
        yield return null;
      }
```

```
      yield return null;
      StartCoroutine(WavesMover());
    }

    public void StopSignalsAnimation()
    {
      StopAllCoroutines();
      if (waves != null)
      {
        for (int i = 0; i < waves.Length; i++)
        {
          if (waves[i] != null) Destroy(waves[i]);
        }
        waves = null;
      }
    }


    void SetPoints(Vector3 _startPos, Vector3 _endPos, Vector3 _center)
    {
      Vector3 rad1 = _startPos - _center;
      Vector3 rad2 = _endPos - _center;

      axis = Vector3.Cross(rad1, rad2).normalized;
      angle = Vector3.SignedAngle(rad1, rad2, axis);
    }

  }
}
```

10. Исходный код скрипта SignalFormingController.cs. Изменение цвета системы частиц объекта визуализированного «цифрового облака» сигнала.

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

namespace StructAndDiffs
{
  public class SignalFormingController : MonoBehaviour
  {
    [SerializeField] ParticlesColorChanger colorChangerPseudo, colorChangerNavMessage;

    private static SignalFormingController instance = null;
    public static SignalFormingController Instance { get { return instance; } }

    void Start()
    {
      instance = this;
    }

    private void OnDestroy()
    {
      instance = null;
    }

    public void SetPseudoRandomCodeState(bool state)
```

```
    {
      colorChangerPseudo.SetColor(state);
    }

    public void SetNavMessageState(bool state)
    {
      colorChangerNavMessage.SetColor(state);
    }
  }
}
```

11. Исходный код скрипта TrilatProcessFlowController.cs. Управление работой сцены «Принципы определения местоположения».

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

public class TrilatProcessFlowController : MonoBehaviour
{
  public enum NavSystem
  {
    GLONASS,
    GPS
  }

  public enum TrilaterationStep
  {
    NavSystemSelection,
    PositionSelection,
    AnimationStart,
    Anim1stSat,
    Anim2ndSat,
    Anim3ndSat,
    Anim4thSat,
    End
  }

  TrilaterationStep currentStep = TrilaterationStep.NavSystemSelection;
  NavSystem navSystem = NavSystem.GLONASS;
  bool satellitesStopped = false;
  bool animPaused = true;

  [SerializeField] string mainSceneId = "Main menu";

  [SerializeField] GameObject selectPointTextPanel;
  [SerializeField] TrilatUIDescription trilatDescr;
  [SerializeField] GameObject uiSliderRecieverDeltaPanel;
  [SerializeField] Slider uiSliderReciverDelta;
  [SerializeField] TMPro.TextMeshProUGUI textRecieverDelta;

  [SerializeField] GameObject uiNavSystemSelection;
  [SerializeField] GameObject uiBackRepeatPanel;
  [SerializeField] GameObject uiFlowControlButtonsPanel;
  [SerializeField] GameObject uiButtonPlayPause;
  [SerializeField] TMPro.TextMeshProUGUI uiPlayPauseText;
  [SerializeField] GameObject uiButtonNextStep;
```

```
  [SerializeField] GameObject glonassObject, gpsObject;

  [SerializeField] Transform[] glonassSatellites;
  [SerializeField] Transform[] gpsSatellites;

  [SerializeField] GameObject targetPrefab;

  TrilatSatellite[] glonassSats;
  TrilatSatellite[] gpsSats;

  [SerializeField] float satellitesRotationSpeed = 0.5f;
  float satRotationAngle = 0f;

  GameObject target = null;

  TrilatSatellite[] currentSats;
  TrilatSatellite[] closestSats = null;

  [SerializeField] GameObject halfSpherePrefab;
  [SerializeField] GameObject conesCoverPrefab;
  [SerializeField] GameObject conesAnimPrefab;
  [SerializeField] GameObject sphereAnimPrefab;
  [SerializeField] GameObject intersectionLinePrefab;

  [SerializeField] Material coneCoverMat;
  [SerializeField] Material[] halfSpheresMat;

  GameObject[] halfSpheres;
  GameObject[] conesCover;
  GameObject[] conesAnim;
  GameObject[] spheresAnim;
  GameObject[] intersections;

  const int numUsedSats = 4;
  const float glonassAngle = 19f;
  const float gpsAngle = 23.5f;

  void UpdateRotationSatelliteValue()
  {
    satRotationAngle += Time.deltaTime * satellitesRotationSpeed;
    if (satRotationAngle > 360f) satRotationAngle -= 360f;
  }

  void UpdateGlonassSatPos()
  {
    for (int sat = 0; sat < 8; sat++)
    {
      Quaternion localSatRotation = Quaternion.Euler(0f, satRotationAngle + sat * 45f,
0f);
      glonassSatellites[sat + 0].localRotation = localSatRotation;
      glonassSatellites[sat + 8].localRotation = localSatRotation;
      glonassSatellites[sat + 16].localRotation = localSatRotation;
    }
  }

  void UpdateGpsSatPos()
  {
    for (int sat = 0; sat < 4; sat++)
    {
      Quaternion localSatRotation = Quaternion.Euler(0f, satRotationAngle + sat * 90f,
0f);
      gpsSatellites[sat + 0].localRotation = localSatRotation;
      gpsSatellites[sat + 4].localRotation = localSatRotation;
```

```
      gpsSatellites[sat + 8].localRotation = localSatRotation;
      gpsSatellites[sat + 12].localRotation = localSatRotation;
      gpsSatellites[sat + 16].localRotation = localSatRotation;
      gpsSatellites[sat + 20].localRotation = localSatRotation;
    }
  }

  public void SelectGLONASS()
  {
    navSystem = NavSystem.GLONASS;
    glonassObject.SetActive(true);
    gpsObject.SetActive(false);
    uiNavSystemSelection.SetActive(false);
    currentSats = glonassSats;
    currentStep = TrilaterationStep.PositionSelection;
    selectPointTextPanel.SetActive(true);
  }

  public void SelectGPS()
  {
    navSystem = NavSystem.GPS;
    glonassObject.SetActive(false);
    gpsObject.SetActive(true);
    uiNavSystemSelection.SetActive(false);
    currentSats = gpsSats;
    currentStep = TrilaterationStep.PositionSelection;
    selectPointTextPanel.SetActive(true);
  }

  void PointSelect(Vector3 pos, Vector3 normal)
  {
    selectPointTextPanel.SetActive(false);
    uiSliderReciverDelta.value = 0f;
    currentStep = TrilaterationStep.AnimationStart;

    satellitesStopped = true;
    animPaused = true;

    uiFlowControlButtonsPanel.SetActive(true);
    uiButtonNextStep.GetComponent<Button>().interactable = true;
    uiButtonPlayPause.GetComponent<Button>().interactable = false;

    target = Instantiate(targetPrefab);
    target.transform.position = pos;
    target.transform.rotation = Quaternion.LookRotation(normal);

    closestSats = currentSats.OrderBy(sat => Vector3.SqrMagnitude(sat.GetPosition() -
target.transform.position)).Take(numUsedSats).ToArray();
    halfSpheres = new GameObject[numUsedSats];
    conesCover = new GameObject[numUsedSats];
    conesAnim = new GameObject[numUsedSats];
    spheresAnim = new GameObject[numUsedSats];

    intersections = new GameObject[6];
    for (int i = 0; i < 6; i++) intersections[i] = Instantiate(intersectionLinePrefab);


intersections[0].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].trans
form.position, closestSats[1].transform.position, target.transform.position);

intersections[1].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].trans
form.position, closestSats[2].transform.position, target.transform.position);
```

```
intersections[2].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[1].trans
form.position, closestSats[2].transform.position, target.transform.position);

intersections[3].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].trans
form.position, closestSats[3].transform.position, target.transform.position);

intersections[4].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[1].trans
form.position, closestSats[3].transform.position, target.transform.position);

intersections[5].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[2].trans
form.position, closestSats[3].transform.position, target.transform.position);

    intersections[0].SetActive(false);
    intersections[1].SetActive(false);
    intersections[2].SetActive(false);
    intersections[3].SetActive(false);
    intersections[4].SetActive(false);
    intersections[5].SetActive(false);

    for (int i = 0; i < closestSats.Length; i++)
    {
      closestSats[i].SetOutline(true);

      const float rEarth = 20f;
      const float targetRadius = 10f;

      float satAngle = navSystem == NavSystem.GLONASS ? glonassAngle : gpsAngle;
      float rSat = closestSats[i].GetPosition().magnitude;
      float discr = rEarth * rEarth - rSat * rSat * Mathf.Sin(Mathf.Deg2Rad * satAngle) *
Mathf.Sin(Mathf.Deg2Rad * satAngle);
      float coneLength = (discr > 0) ? (rSat * Mathf.Cos(Mathf.Deg2Rad * satAngle) -
Mathf.Sqrt(discr)) : rSat;

      conesCover[i] = Instantiate(conesCoverPrefab);
      conesCover[i].transform.position = closestSats[i].transform.position;
      conesCover[i].transform.rotation = closestSats[i].transform.rotation;
      conesCover[i].GetComponent<TrilatConeCover>().GenerateMesh(satAngle);
      conesCover[i].GetComponent<TrilatConeCover>().SetMaterial(coneCoverMat);
      conesCover[i].transform.localScale = Vector3.one * coneLength;

      halfSpheres[i] = Instantiate(halfSpherePrefab);
      halfSpheres[i].transform.position = closestSats[i].transform.position;
      halfSpheres[i].transform.rotation = closestSats[i].transform.rotation;
      halfSpheres[i].GetComponent<TrilatHalfSphere>().SetMaterial(halfSpheresMat[i]);
      halfSpheres[i].SetActive(false);

      conesAnim[i] = Instantiate(conesAnimPrefab);
      conesAnim[i].transform.position = closestSats[i].transform.position;
      conesAnim[i].transform.rotation = Quaternion.LookRotation(target.transform.position
- closestSats[i].transform.position);
      conesAnim[i].GetComponent<TrilatConeToTarget>().GenerateMesh((pos -
closestSats[i].transform.position).magnitude, targetRadius, false);
      conesAnim[i].GetComponent<TrilatConeToTarget>().SetMaterial(halfSpheresMat[i]);
      conesAnim[i].SetActive(false);

      spheresAnim[i] = Instantiate(sphereAnimPrefab);
      spheresAnim[i].transform.position = closestSats[i].transform.position;
      spheresAnim[i].transform.rotation =
Quaternion.LookRotation(target.transform.position - closestSats[i].transform.position);
      spheresAnim[i].GetComponent<TrilatSphereSegToTarget>().GenerateMesh((pos -
closestSats[i].transform.position).magnitude, targetRadius, false);
```

```
spheresAnim[i].GetComponent<TrilatSphereSegToTarget>().SetMaterial(halfSpheresMat[i]);
      spheresAnim[i].SetActive(false);
    }

    StartCoroutine(SatAnimation());
  }

  public void SwitchPausePlay()
  {
    animPaused = !animPaused;

    if (animPaused)
    {
      uiPlayPauseText.text = "Продолжить";
    }
    else
    {
      uiPlayPauseText.text = "Пауза";
      uiButtonNextStep.GetComponent<Button>().interactable = false;
      uiButtonPlayPause.GetComponent<Button>().interactable = true;
    }
  }

  public void ReturnToMain()
  {
    uiNavSystemSelection.SetActive(true);

    trilatDescr.SetText(0);
    trilatDescr.Hide();
    selectPointTextPanel.SetActive(false);
    satellitesStopped = false;

    glonassObject.SetActive(false);
    gpsObject.SetActive(false);
    currentStep = TrilaterationStep.NavSystemSelection;
    uiSliderRecieverDeltaPanel.SetActive(false);
    uiBackRepeatPanel.SetActive(false);

    ClearScene();
  }

  public void SelectNewPoint()
  {
    selectPointTextPanel.SetActive(true);
    trilatDescr.SetText(0);
    trilatDescr.Hide();
    satellitesStopped = false;
    uiSliderRecieverDeltaPanel.SetActive(false);
    currentStep = TrilaterationStep.PositionSelection;
    uiBackRepeatPanel.SetActive(false);
    ClearScene();
  }

  void ClearScene()
  {
    for (int i = 0; i < closestSats.Length; i++)
    {
      closestSats[i].SetOutline(false);
    }

    Destroy(target);
```

```
    for (int i = 0; i < halfSpheres.Length; i++)
    {
      Destroy(halfSpheres[i]);
      Destroy(conesCover[i]);
      Destroy(conesAnim[i]);
      Destroy(spheresAnim[i]);
    }

    for (int i = 0; i < intersections.Length; i++)
    {
      Destroy(intersections[i]);
    }

    target = null;
    halfSpheres = null;
    conesCover = null;
    conesAnim = null;
    spheresAnim = null;
    intersections = null;
  }

  IEnumerator SatAnimation()
  {
    trilatDescr.Show();
    trilatDescr.SetText(0);
    yield return null;
    for (int i = 0; i < closestSats.Length; i++)
    {
      while (animPaused) yield return null;
      trilatDescr.SetText(i + 1);
      if (i == 0) for (int j = 0; j < conesCover.Length; j++)
conesCover[j].SetActive(false);
      currentStep = TrilaterationStep.Anim1stSat + i;
      yield return AnimateSatellite(i);

      animPaused = true;
      uiButtonNextStep.GetComponent<Button>().interactable = true;
      uiButtonPlayPause.GetComponent<Button>().interactable = false;
    }

    uiFlowControlButtonsPanel.SetActive(false);
    uiBackRepeatPanel.SetActive(true);
    uiSliderRecieverDeltaPanel.SetActive(true);
    currentStep = TrilaterationStep.End;
  }

  IEnumerator AnimateSatellite(int num)
  {
    const float growthTime = 3f;
    const float sphereStayTime = 1f;
    const float coneStayTime = 1f;

    float maxLen = (target.transform.position -
closestSats[num].transform.position).magnitude;
    float timer = 0f;

    halfSpheres[num].transform.localScale = Vector3.zero;
    halfSpheres[num].SetActive(true);

    while (timer < growthTime)
    {
      if (!animPaused)
      {
```

```
        halfSpheres[num].transform.localScale = Vector3.one * Mathf.Lerp(0, maxLen, timer
/ growthTime);
        timer += Time.deltaTime;
      }
      yield return null;
    }
    halfSpheres[num].transform.localScale = Vector3.one * maxLen;

    timer = 0f;
    while (timer < sphereStayTime)
    {
      if (!animPaused) timer += Time.deltaTime;
      yield return null;
    }

    halfSpheres[num].SetActive(false);

    conesAnim[num].SetActive(true);
    conesAnim[num].transform.localScale = Vector3.one * maxLen;
    spheresAnim[num].SetActive(true);
    spheresAnim[num].transform.localScale = Vector3.one * maxLen;

    timer = 0f;
    while (timer < coneStayTime)
    {
      if (!animPaused) timer += Time.deltaTime;
      yield return null;
    }
    conesAnim[num].SetActive(false);

    switch (num)
    {
    case 1:
      intersections[0].SetActive(true);
      break;
    case 2:
      intersections[1].SetActive(true);
      intersections[2].SetActive(true);
      break;
    case 3:
      intersections[3].SetActive(true);
      intersections[4].SetActive(true);
      intersections[5].SetActive(true);
      break;
    }
  }

  private void Awake()
  {
    //init satellite arrays
    glonassSats = new TrilatSatellite[glonassSatellites.Length];
    for (int i = 0; i < glonassSatellites.Length; i++) glonassSats[i] =
glonassSatellites[i].GetComponentInChildren<TrilatSatellite>();

    gpsSats = new TrilatSatellite[gpsSatellites.Length];
    for (int i = 0; i < gpsSatellites.Length; i++) gpsSats[i] =
gpsSatellites[i].GetComponentInChildren<TrilatSatellite>();

    uiSliderRecieverDeltaPanel.SetActive(false);
    gpsObject.SetActive(false);
    glonassObject.SetActive(false);
    uiFlowControlButtonsPanel.SetActive(false);
    uiBackRepeatPanel.SetActive(false);
```

```
    selectPointTextPanel.SetActive(false);
    trilatDescr.Hide();
  }

  void Start ()
  {

      }

      void Update ()
  {
    if (currentStep == TrilaterationStep.PositionSelection) UpdatePosSelection();

    UpdateSatellitesPositions();
  }

  private void UpdateSatellitesPositions()
  {
    if (!satellitesStopped)
    {
      UpdateRotationSatelliteValue();
      UpdateGlonassSatPos();
      UpdateGpsSatPos();
    }
  }

  public void ChangeRadiuses(float delta)
  {
    textRecieverDelta.text = delta.ToString("0.000") + " мс";

    const float lightSpeed = 1.1065f; // (scene units/ms)
    delta = delta * lightSpeed;
    if (target != null && spheresAnim != null && intersections != null)
    {
      float[] rad = new float[spheresAnim.Length];
      for (int i = 0; i < spheresAnim.Length; i++)
      {
        rad[i] = (closestSats[i].transform.position -
target.transform.position).magnitude - delta;
        spheresAnim[i].transform.localScale = Vector3.one * rad[i];
      }


intersections[0].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].trans
form.position, closestSats[1].transform.position, rad[0], rad[1],
target.transform.position);

intersections[1].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].trans
form.position, closestSats[2].transform.position, rad[0], rad[2],
target.transform.position);

intersections[2].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[1].trans
form.position, closestSats[2].transform.position, rad[1], rad[2],
target.transform.position);

intersections[3].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[0].trans
form.position, closestSats[3].transform.position, rad[0], rad[3],
target.transform.position);

intersections[4].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[1].trans
form.position, closestSats[3].transform.position, rad[1], rad[3],
target.transform.position);
```

```
intersections[5].GetComponent<SphereIntersectionLine>().GenerateLine(closestSats[2].trans
form.position, closestSats[3].transform.position, rad[2], rad[3],
target.transform.position);
    }

  }

  void UpdatePosSelection()
  {
    if (Input.GetMouseButtonDown(0))
    {
      Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
      RaycastHit hit;
      if (Physics.Raycast(ray, out hit))
      {
        if (hit.collider.gameObject.name == "Earth")
        {
          PointSelect(hit.point, hit.normal);
        }
      }
    }
  }
}
```

12. Исходный код скрипта Weather.cs. Управление работой сцен «Дождь» и «Снегопад».

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

public class Wather : MonoBehaviour {
    [System.Serializable]
    public struct weatherData
    {
        public string id;
        public Color fogColor;
        public float fogDensity;
        public float fogSkyDensity;
        public float fogEnd;
        public Texture skyMap;
        public GameObject rain;
        public GameObject radioWaves;
        public Color light;
    }

    public List<weatherData> WeatherData;
    public int currentIndex;
    public Light light;

    private int _currentIndex = -1;
    private weatherData weather;

    private void Start()
    {
        setCurrentIndex(currentIndex);
    }

    private void Update()
```

```
    {
        setCurrentIndex(currentIndex);
    }

    private void setCurrentIndex(int a_index)
    {
        if (a_index != _currentIndex)
        {
            if (_currentIndex > -1)
            {
                if (weather.rain != null) weather.rain.SetActive(false);
                if (weather.radioWaves != null) weather.radioWaves.SetActive(false);
            }
            _currentIndex = currentIndex = Mathf.Min(a_index, WeatherData.Count - 1);

            weather = WeatherData[_currentIndex];
            RenderSettings.fogDensity = weather.fogDensity;
            RenderSettings.fogColor = weather.fogColor;

            RenderSettings.skybox.SetFloat("_FogEnd", weather.fogEnd);
            RenderSettings.skybox.SetFloat("_FogIntens", weather.fogSkyDensity);
            RenderSettings.skybox.SetColor("_FogCol", weather.fogColor);
            RenderSettings.skybox.SetTexture("_Tex", weather.skyMap);

            if (weather.rain != null) weather.rain.SetActive(true);
            if (weather.radioWaves != null) weather.radioWaves.SetActive(true);

            light.color = weather.light;
        }
    }

    public void ApplyData(StageManager manager)
    {
        setCurrentIndex(indexOf(manager.getCurrentData().id));
    }

    private int indexOf(string id)
    {
        for (int i=0; i< WeatherData.Count; i++)
        {
            if (WeatherData[i].id.Contains(id)) return i;
        }
        return 0;
    }
}
```

13. Исходный код скрипта Rotate_the_object.cs. Вращение объектов с задаваемой скоростью.

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

public class Rotate_the_object : MonoBehaviour {

       public float Rotation_X=0.0f;
       public float Rotation_Y=20.0f;
       public float Rotation_Z=0.0f;

       void Update () {
```

```
        transform.Rotate(new Vector3(Rotation_X,Rotation_Y,Rotation_Z) * Time.deltaTime);
       }
}
```

14. Исходный код скрипта AtmosphereEffectManager.cs. Управление работой сцены «Атмосфера».

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

public class AtmosphereEffectManager : MonoBehaviour {
    [System.Serializable]
    public struct afStageData
    {
        public string id;
        public float afScale;
        public float rspeed;
        public bool cameraMove;
    }

    public List<afStageData> afStages;
    public int currentIndex;

    public Earth earth;
    public ParticleSystem v1;
    public ParticleSystem v2;
    public ParticleSystem v3;
    public Animator cameraAnimator;
    public List<GameObject> animHidden;

    private int _currentIndex = -1;
    private afStageData stages;
    private Vector3 startPos;
    private Quaternion startRotate;
    private float esmback;

    private void Awake()
    {
        startPos = cameraAnimator.transform.position;
        startRotate = cameraAnimator.transform.rotation;

        esmback = earth.smooth;
    }

    private void setCurrentIndex(int a_index)
    {
        if (a_index != _currentIndex)
        {
            _currentIndex = a_index;

            afStageData item = afStages[_currentIndex];
            earth.rateTo = item.afScale;

            ParticleSystem.MainModule main = v1.main;
            main.simulationSpeed = item.rspeed;

            main = v2.main;
            main.simulationSpeed = item.rspeed;
```

```
            main = v3.main;
            main.simulationSpeed = item.rspeed;

            if (cameraAnimator.enabled = item.cameraMove)
            {
                cameraAnimator.Play("camera_move", -1, 0);
                earth.smooth = 0.5f;
            } else
            {
                cameraAnimator.transform.position = startPos;
                cameraAnimator.transform.rotation = startRotate;
                earth.smooth = esmback;
            }

            foreach (GameObject ga in animHidden)
                ga.SetActive(!item.cameraMove);
        }
    }

    public void ApplyData(StageManager manager)
    {
        setCurrentIndex(indexOf(manager.getCurrentData().id));
    }

    private int indexOf(string id)
    {
        for (int i = 0; i < afStages.Count; i++)
        {
            if (afStages[i].id.Contains(id)) return i;
        }
        return 0;
    }
}
```

15. Исходный код скрипта Distance.cs. Перемещение камеры на сцене «Атмосфера».

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

public class Distance : MonoBehaviour {
    public Transform target;
    public float distanceTo;
    public float smooth = 0.2f;

    private float _distance;
    private Vector3 start;

    private void Awake()
    {
        start = transform.position;
        _distance = distanceTo;
        updatePos();
    }

    private void Update()
    {
        distanceTo = Mathf.Clamp(distanceTo, 0, 1);
        if (distanceTo != _distance) updatePos();
```

```
    }

    private void updatePos()
    {
        transform.position = Vector3.Lerp(target.position, start, _distance);
        _distance += (distanceTo - _distance) * smooth;
    }
}
```

16. Исходный код скрипта DragMouseOrbitOrigin.cs. Управление камерой на некоторых сценах.

```
using UnityEngine;
using UnityEngine.EventSystems;
using System.Collections;

[AddComponentMenu("Camera-Control/DragMouseOrbitOrigin")]
public class DragMouseOrbitOrigin : MonoBehaviour
{
    [SerializeField]
    private Transform target;

    public Transform Target
    {
        get
        {
            return target;
        }
    }
    public float distance = 800.0f;
    public float xSpeed = 20.0f;
    public float ySpeed = 20.0f;
    public float yMinLimit = -90f;
    public float yMaxLimit = 90f;
    public float xMinLimit = 0f;
    public float xMaxLimit = 0f;
    public float distanceMin = 10f;
    public float distanceMax = 10f;
    public float smoothTime = 2f;
    public float ScrollWheel = 5f;
    public float backRotate = 0;
    public Vector2 startPulse = Vector2.zero;

    float rotationYAxis = 0.0f;
    float rotationXAxis = 0.0f;
    float distanceCurrent;
    Vector2 velocity = Vector2.zero;
    int xAroundKey = 0;
    int yAroundKey = 0;


    public OverControl justOverIt;

    private bool initialized = false;

    //Use this for initialization
    void Start()
    {
        if (!initialized) updateFromTarget();
        velocity += new Vector2(startPulse.x, startPulse.y);
```

```
        if (GetComponent<Rigidbody>()) GetComponent<Rigidbody>().freezeRotation = true;
    }

    void getInput()
    {
        if (Input.GetMouseButton(0))
            {
                velocity += new Vector2(xSpeed * Input.GetAxis("Mouse X") * distance *
0.02f, ySpeed * Input.GetAxis("Mouse Y") * 0.02f);
            }
            else {

                if (Input.GetKeyDown(KeyCode.LeftArrow)) xAroundKey = 1;
                else if (Input.GetKeyDown(KeyCode.RightArrow)) xAroundKey = -1;
                else if (Input.GetKeyUp(KeyCode.LeftArrow) ||
Input.GetKeyUp(KeyCode.RightArrow)) xAroundKey = 0;

                if (Input.GetKeyDown(KeyCode.UpArrow)) yAroundKey = 1;
                else if (Input.GetKeyDown(KeyCode.DownArrow)) yAroundKey = -1;
                else if (Input.GetKeyUp(KeyCode.UpArrow) ||
Input.GetKeyUp(KeyCode.DownArrow)) yAroundKey = 0;
            }

        distance = Mathf.Clamp(distance - Input.GetAxis("Mouse ScrollWheel") *
ScrollWheel, distanceMin, distanceMax);

    }

    public void setTarget(Transform a_target)
    {
        if (enabled = (target = a_target) != null) updateFromTarget();
    }

    public void updateFromTarget()
    {
        transform.LookAt(target.position, Vector3.up);
        distance = (target.position - transform.position).magnitude;

        Vector3 angles = transform.eulerAngles;
        rotationYAxis = angles.y;
        while ((xMaxLimit != 0) && (rotationYAxis > xMaxLimit)) rotationYAxis =
rotationYAxis - 360;
        while ((xMinLimit != 0) && (rotationYAxis < xMinLimit)) rotationYAxis =
rotationYAxis + 360;

        rotationXAxis = angles.x;
        while ((yMaxLimit != 0) && (rotationXAxis > yMaxLimit)) rotationXAxis =
rotationXAxis - 360;
        while ((yMinLimit != 0) && (rotationXAxis < yMinLimit)) rotationXAxis =
rotationXAxis + 360;

        distanceCurrent = distance;

        initialized = true;
    }

    void LateUpdate()
    {
        if (target) {

            if ((!justOverIt || justOverIt.isFocus) && (justOverIt ||
!EventSystem.current.IsPointerOverGameObject())) getInput();
            else
```

```
            {
                velocity += new Vector2(backRotate, 0);
            }

            velocity += new Vector2(xAroundKey * xSpeed * distance * 0.02f, yAroundKey *
xSpeed * distance * 0.02f);

            rotationYAxis += velocity.x;

            if ((xMinLimit != 0) && (rotationYAxis < xMinLimit))
            {
                rotationYAxis = xMinLimit;
                velocity.x = 0;
            }

            if ((xMaxLimit != 0) && (rotationYAxis > xMaxLimit))
            {
                rotationYAxis = xMaxLimit;
                velocity.x = 0;
            }

            rotationXAxis = rotationXAxis + velocity.y;

            if (rotationXAxis < yMinLimit)
            {
                rotationXAxis = yMinLimit;
                velocity.y = 0;
            }

            if (rotationXAxis > yMaxLimit)
            {
                rotationXAxis = yMaxLimit;
                velocity.y = 0;
            }

            Quaternion toRotation = Quaternion.Euler(rotationXAxis, rotationYAxis, 0);
            Quaternion rotation = toRotation;

            distanceCurrent += (distance - distanceCurrent) * Time.deltaTime * smoothTime
* 2f;

            Vector3 negDistance = new Vector3(0.0f, 0.0f, -distanceCurrent);
            Vector3 position = rotation * negDistance + target.position;

            transform.rotation = rotation;
            transform.position = position;

            velocity -= velocity * Time.deltaTime * smoothTime;
        }
    }
}
```

17. Исходный код скрипта CameraMode.cs. Изменение режима 2D / 3D и соответствующих настроек камеры.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class CameraMode : MonoBehaviour {

    public float ortoSize;

    private Camera camera;

    private Vector3 startPosition;
    private Quaternion startRotation;

    public UnityEvent on3dMode;
    public UnityEvent on2dMode;

    public bool mode3d
    {
        get
        {
            return !camera.orthographic;
        }
    }

    private void Start()
    {
        camera = GetComponent<Camera>();
        startPosition = transform.position;
        startRotation = transform.rotation;
        updateFromMode();
        invoceFromMode();
    }

    public void toggleMode()
    {
        if (mode3d)
        {
            camera.orthographic = true;
            updateFromMode();
            invoceFromMode();
        }
        else
        {
            camera.orthographic = false;
            updateFromMode();
            invoceFromMode();
        }
    }

    private void invoceFromMode()
    {
        if (mode3d) on3dMode.Invoke();
        else on2dMode.Invoke();
    }

    private void updateFromMode()
    {
```

```
        if (!mode3d)
        {
            camera.orthographicSize = ortoSize;
            transform.position = startPosition;
            transform.rotation = startRotation;
        }
        GetComponent<DragMouseOrbitOrigin>().enabled = mode3d;
    }
}
```

18. Исходный код скрипта RayIntersect.cs. Перемещение и вращение геометрических фигур на сцене «Спутники».

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

public class RayIntersect : MonoBehaviour {
    [System.Serializable]
    public struct itsData
    {
        public Transform tranform;
        public float angle;
    }
    [System.Serializable]
    public struct rayData
    {
        public Transform ray;
        public Transform target;
        private float slen;
        public float calcScale(float a_len)
        {
            if (slen == 0) slen = a_len;

            return a_len / slen;
        }
    }

    [SerializeField]
    private List<rayData> rays;
    [SerializeField]
    private List<itsData> intersect;
    [SerializeField]
    private Transform center;
    [SerializeField]
    private float scaleFactor;

    private float angle;
    private float _prevAngle = -360;


    private void Update()
    {
        float calc_angle = 0;
        Vector3 baseVector = Vector3.zero;
        foreach (rayData data in rays)
        {
            Transform a_center = center ? center : transform;
            Vector3 d = data.target.position - a_center.position;
```

```
            data.ray.rotation = Quaternion.LookRotation(d) * Quaternion.AngleAxis(90,
Vector3.forward);
            if (baseVector.magnitude == 0) baseVector = d;
            else calc_angle = Mathf.Max(Vector3.Angle(baseVector, d), calc_angle);

            float scale = data.calcScale(d.magnitude) * scaleFactor;
            data.ray.localScale = new Vector3(scale, scale, scale);
        }

        angle = discreteAngle(calc_angle);

        if (_prevAngle != angle)
        {

            foreach (itsData data in intersect)
            {
                if (data.angle == angle) data.tranform.gameObject.SetActive(true);
                else data.tranform.gameObject.SetActive(false);
            }
            _prevAngle = angle;
        }
    }

    private int discreteAngle(float a_angle)
    {
        return (int)Mathf.Round(Mathf.Round(a_angle / 5) * 5);
    }
}
```

19. Исходный код скрипта *CityManager*.cs. Управление работой сцены «Город».

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

public class CityManager : MonoBehaviour {
    [System.Serializable]
    public struct stageData
    {
        public string id;
        public Transform point;
        public Camera displayCamera;
        public GameObject systems;
        public float FieldOfView;
        public DragMouseOrbitOrigin dragMouse;
        public Vector2 contentPosition;
    }
    public GameObject previewPanel;
    public GameObject contentPanel;
    public List<stageData> stages;
    public int currentIndex;
    public float mstep = 0.01f;
    public float maxHeight = 40f;
    public GameObject cameraObject;
    public AudioSource transition;

    private int _currentIndex = -1;
    private float ainx = 0;
```

```
    private Vector3[] top = new Vector3[2];
    private Vector3 start;
    private Quaternion startRotate;
    private stageData prev;

    private void Awake()
    {
        previewPanel.SetActive(false);
    }

    private void FixedUpdate()
    {
        if (_currentIndex > -1)
        {
            if (stages[_currentIndex].point && (ainx <= 1)) {
                stageData item = stages[_currentIndex];

                int stage = (int)(ainx * 4) % 4;
                float istg = ainx * 4 - stage;

                Quaternion toDirect = Quaternion.LookRotation(item.point.position -
start);

                switch (stage)
                {
                    case 0: cameraObject.transform.rotation =
Quaternion.Lerp(startRotate, toDirect, istg);
                        break;
                    case 1: cameraObject.transform.position = Vector3.Lerp(start, top[0],
istg);
                        break;
                    case 2: cameraObject.transform.position = Vector3.Lerp(top[0],
top[1], istg);
                        break;
                    case 3: cameraObject.transform.position = Vector3.Lerp(top[1],
item.point.position, istg);
                        cameraObject.transform.rotation = Quaternion.Lerp(toDirect,
item.point.rotation, istg);
                        break;
                }


                Camera camera = cameraObject.GetComponentInChildren<Camera>();
                camera.fieldOfView = Mathf.Lerp(camera.fieldOfView, item.FieldOfView,
ainx);

                ainx += mstep;
                if (ainx > 1) afterChange();
            }
        }
    }

    private void afterChange()
    {
        stageData item = stages[_currentIndex];
        if (item.systems) item.systems.SetActive(true);

        if (item.dragMouse)
        {

            if (prev.systems) prev.systems.SetActive(false);

            cameraObject.GetComponent<TransformCopy>().other = item.dragMouse.transform;
```

```
            item.dragMouse.enabled = true;
            item.dragMouse.updateFromTarget();
        }
    }

    private void setCurrentIndex(int a_index)
    {
        if (a_index != _currentIndex)
        {
            stageData item = stages[a_index];

            previewPanel.SetActive(item.displayCamera != null);
            if (item.displayCamera != null)
                previewPanel.GetComponentInChildren<RawImage>().texture =
item.displayCamera.targetTexture;


            if (_currentIndex > -1)
            {
                prev = stages[_currentIndex];

                top[0] = new Vector3(prev.point.position.x, maxHeight,
prev.point.position.z);
                top[1] = new Vector3(item.point.position.x, maxHeight,
item.point.position.z);

                if (item.dragMouse)
                    item.point.LookAt(item.dragMouse.Target.position, Vector3.up);

                start = cameraObject.transform.position;
                startRotate = cameraObject.transform.rotation;

                transition.Play();
                if (prev.dragMouse) prev.dragMouse.enabled = false;

                ainx = 0;
                cameraObject.GetComponent<TransformCopy>().other = null;
            }
            else ainx = 1;

            contentPanel.GetComponent<RectTransform>().anchoredPosition =
item.contentPosition;
            _currentIndex = a_index;
        }
    }

    public void ApplyData(StageManager manager)
    {
        setCurrentIndex(indexOf(manager.getCurrentData().id));
    }

    private int indexOf(string id)
    {
        for (int i = 0; i < stages.Count; i++)
        {
            if (stages[i].id.Contains(id)) return i;
        }
        return 0;
    }

}
```

20. Исходный код скрипта CoordAccuracy.cs. Управление работой сцены «Показатели точности определения координат».

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class CoordAccuracy : MonoBehaviour
{
    [SerializeField] GameObject[] panelsClear;
    [SerializeField] TMP_Dropdown dropdown;

    public void MainMenu()
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene("Main menu");
    }

    public void PanelMain(GameObject panelMain)
    {
        for (int i = 0; i < panelsClear.Length; i++)
        {
            panelsClear[i].SetActive(false);
            if (panelsClear[i] == panelMain) dropdown.value = i;
        }
        panelMain.SetActive(true);

    }
    public void ChooseStep()
    {
        switch (dropdown.value)
        {
            case 0:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[0].SetActive(true);
                break;
            case 1:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[1].SetActive(true);
                break;
            case 2:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[2].SetActive(true);
                break;
            case 3:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[3].SetActive(true);
```

```
                break;
            case 4:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[4].SetActive(true);
                break;
            case 5:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[5].SetActive(true);
                break;
            case 6:
                for (int i = 0; i < panelsClear.Length; i++)
                {
                    panelsClear[i].SetActive(false);
                }
                panelsClear[6].SetActive(true);
                break;
        }
    }
```

21. Исходный код скрипта UsingGNSSManager.cs. Управление работой сцены «Практическое применение ССН».

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

public class UsingGNSSManager : MonoBehaviour {
    [System.Serializable]
    public struct stageData
    {
        public string id;
        public Transform point;
        public float FieldOfView;
        public DragMouseOrbitOrigin dragMouse;
        public Vector2 contentPosition;
        public List<GameObject> objects;
        public bool toCourse;
        public float UiAddScale;
    }
    public GameObject contentPanel;
    public List<stageData> stages;
    public int currentIndex;
    public float mstep = 0.01f;
    public float maxHeight = 40f;
    public GameObject cameraObject;
    public AudioSource transition;

    private int _currentIndex = -1;
    private float ainx = 0;
    private Vector3[] top = new Vector3[2];
    private Vector3 start;
    private Quaternion startRotate;
    private stageData prev;
```

```
    private void FixedUpdate()
    {
        if (_currentIndex > -1)
        {
            if (stages[_currentIndex].point && (ainx <= 1))
            {
                stageData item = stages[_currentIndex];

                if (item.toCourse)
                {

                    int stage = (int)(ainx * 4) % 4;
                    float istg = ainx * 4 - stage;

                    Quaternion toDirect = Quaternion.LookRotation(item.point.position -
start);

                    switch (stage)
                    {
                        case 0:
                            cameraObject.transform.rotation =
Quaternion.Lerp(startRotate, toDirect, istg);
                            break;
                        case 1:
                            cameraObject.transform.position = Vector3.Lerp(start, top[0],
istg);
                            break;
                        case 2:
                            cameraObject.transform.position = Vector3.Lerp(top[0],
top[1], istg);
                            break;
                        case 3:
                            cameraObject.transform.position = Vector3.Lerp(top[1],
item.point.position, istg);
                            cameraObject.transform.rotation = Quaternion.Lerp(toDirect,
item.point.rotation, istg);
                            break;
                    }

                } else
                {
                    cameraObject.transform.position = Vector3.Lerp(start,
item.point.position, ainx);
                    cameraObject.transform.rotation = Quaternion.Lerp(startRotate,
item.point.rotation, ainx);
                }


                Camera camera = cameraObject.GetComponentInChildren<Camera>();
                camera.fieldOfView = Mathf.Lerp(camera.fieldOfView, item.FieldOfView,
ainx);

                ainx += mstep;
                if (ainx > 1) afterChange();
            }
        }
    }

    private void afterChange()
    {
```

```
        stageData item = stages[_currentIndex];

        if (item.dragMouse)
        {
            cameraObject.GetComponent<TransformCopy>().other = item.dragMouse.transform;
            item.dragMouse.enabled = true;
            item.dragMouse.updateFromTarget();
        }

        if (item.objects.Count > 0)
        {
            GetComponentInChildren<CanvasScaler>().scaleFactor = 1 + item.UiAddScale;
            foreach (GameObject ga in item.objects) ga.SetActive(true);
        }
    }

    private void setCurrentIndex(int a_index)
    {
        if (a_index != _currentIndex)
        {
            stageData item = stages[a_index];

            if (_currentIndex > -1)
            {
                prev = stages[_currentIndex];
                if ((prev.objects != null) && (prev.objects.Count > 0))
                    foreach (GameObject ga in prev.objects) ga.SetActive(false);

                top[0] = new Vector3(prev.point.position.x, maxHeight,
prev.point.position.z);
                top[1] = new Vector3(item.point.position.x, maxHeight,
item.point.position.z);

                if (item.dragMouse)
                    item.point.LookAt(item.dragMouse.Target.position, Vector3.up);
                start = cameraObject.transform.position;
                startRotate = cameraObject.transform.rotation;

                transition.Play();
                if (prev.dragMouse) prev.dragMouse.enabled = false;

                ainx = 0;
                cameraObject.GetComponent<TransformCopy>().other = null;
            }
            else ainx = 1;

            if (contentPanel) contentPanel.GetComponent<RectTransform>().anchoredPosition
= item.contentPosition;
            _currentIndex = a_index;
        }
    }

    public void ApplyData(StageManager manager)
    {
        setCurrentIndex(indexOf(manager.getCurrentData().id));
    }

    private int indexOf(string id)
    {
        for (int i = 0; i < stages.Count; i++)
        {
            if (stages[i].id.Contains(id)) return i;
        }
```

```
            return 0;
        }
    }
```