

**ФЕДЕРАЛЬНОЕ АГЕНТСТВО МОРСКОГО И РЕЧНОГО ТРАНСПОРТА**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Государственный университет морского и речного флота имени адмирала С.О. Макарова»**

**(ФГБОУ ВО «ГУМРФ имени адмирала С.О. Макарова»)**

**Институт водного транспорта**

**Кафедра вычислительных систем и информатики**

ОТЧЕТ ПО РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЕ

по дисциплине:

«Синтез программных систем»

на тему:

«Разработка игры жанра "Симулятор" - "Жизнь почтальона"»

Работу выполнил:  
студент группы ИС-42  
Дель А.А.

Работу принял: доцент кафедры ВСиИ,  
к.т.н., Журавлёв А.Е.

Санкт-Петербург

2024

Оглавление

[Введение 4](#_Toc31728)

[1. Общие сведения, обзор аналогов и формирование требований к проектируемому программному средству 6](#_Toc6762)

[1.1. Общие сведения об игре 6](#_Toc32024)

[1.2. Обзор аналогов 6](#_Toc9449)

[1.3. Сводная таблица оценки 12](#_Toc24409)

[1.4. Требования к проекту 13](#_Toc30222)

[1.5. Задачи 13](#_Toc3276)

[2. Логика живых объектов 15](#_Toc28974)

[2.1. Создание заготовки живых объектов 15](#_Toc24232)

[2.2. Реализация функционала 16](#_Toc18697)

[2.3. Подключение к мировой системе 16](#_Toc10606)

[3. Создание скриптов окружения 18](#_Toc16852)

[3.1. Создание заготовки игрового окружения 18](#_Toc17230)

[3.2. Реализация функционала игрового окружения 18](#_Toc14479)

[3.3. Создание скрипта окружения с общемировой системой 19](#_Toc7852)

[4. Создание скриптов объектов взаимодействия 20](#_Toc16869)

[4.1. Создание заготовок объектов взаимодействия 20](#_Toc8699)

[4.2. Реализаия функционала объектов взаимодействия 21](#_Toc1319)

[4.3. Налаживание связи между объектами взаимодействия, где это требуется. 22](#_Toc28487)

[5. Оптимизация игрового мира 23](#_Toc23597)

[5.1. Поиск проблемных областей кода 23](#_Toc8476)

[5.2. Измерение влияния и существенности проблемы на игровой мир и его системы 23](#_Toc18200)

[5.3. Оптимизация проблемной части кода 23](#_Toc28601)

[6. Руководство программиста 25](#_Toc31564)

[6.1. Игровые настройки 26](#_Toc2211)

[6.2. Игра 33](#_Toc20254)

[6.3. Диаграммы UML 35](#_Toc29722)

[Заключение 38](#_Toc4105)

[Приложение 39](#_Toc32518)

# Введение

Игры жанра «Симулятор» позволяют игрокам войти в виртуальный мир, который имитирует реальность или определенный аспект жизни. Они позволяют игрокам почувствовать себя в роли профессионала, испытать адреналин от опасных ситуаций или просто насладиться повседневной жизнью виртуального персонажа.

Большинство симуляторов фокусируются (полностью или частично) на имитации реальной жизненной деятельности. Тем не менее, многие симуляторы имитируют "фантастические" ситуации, например симуляторы космических сражений. Основными подвидами игровых симуляторов являются: узкотематические, широкотематические.

В основном узкотематические являются симуляторами технических средств, в частности: автомобилями, поездами, самолетами и вертолетами.

Широкотематическими являются симуляторы выполнения различных заданий. Пользователю предстоит выполнение определенных задач определенной профессии. Практически отсутствуют сложные технические средства. Управление становится дополнительным игровым действием вместо основного. Подобные проекты выделяются разноплановых геймплеем.

Широкотематические симуляторы условно разделяют на 3 основные группы:

- Конструкторы (Construction simulations) — моделирование и строительства городов, зданий, техники и тд;

- Симуляторы жизни (Life simulations) — имитация жизни человека;

- Симуляторы профессий (Professional simulations) и Социальные симуляторы (Social simulations) — имитация профессиональной и социальной деятельности;

«Жизнь почтальона» - однопользовательская игра в 3D-мире, в которой игрок берет на себя роль почтальона. Игра представляет собой симулятор профессии в фантастическом, основанном на реальном мире. Игроку предстоит доставлять посылки и письма по поселку «Торковичи», покупать вспомогательное оборудование и технику, сталкиваться с другими существами и, по необходимости, противостоять им.

# Общие сведения, обзор аналогов и формирование требований к проектируемому программному средству

## Общие сведения об игре

Игра «Postman life» представляет собой бродилку в открытом 3D мире, где игроки берут на себя роль почтальона в поселке городского типа.

В игровом мире будут функциональные здания, такие как почта и магазин, жилые и производственные здания, а также различные пасхалки и элементы окружения.

## Обзор аналогов

1. Totally Reliable Delivery Service (TRDS)

Totally Reliable Delivery Service - это основанная на физике игра-симулятор, разработанная We're Five Games и изданная tinyBuild. Действие игры разворачивается в открытом мире, где игроки берут на себя роль водителей доставки в хаотичном и непредсказуемом мире.

Одной из ключевых особенностей игры является многопользовательский режим, который позволяет до четырех игрокам играть вместе онлайн или локально. Игроки могут работать сообща, чтобы завершить доставку, или соревноваться друг с другом, чтобы узнать, кто сможет доставить больше посылок.

Игра доступна для покупки на нескольких платформах, включая Windows, Nintendo Switch, PlayStation 4 и Xbox One. Цена варьируется в зависимости от платформы, но обычно составляет от 15 до 20 долларов.

Минимальные требования для Windows:

Операционная система: Windows 7 или более поздняя версия

Процессор: Intel Core i5 2300 или эквивалент

Оперативная память: 4 ГБ ОЗУ

Графика: NVIDIA GeForce GTX 560 или эквивалент

DirectX: Версия 11

Объем памяти: 3 ГБ свободного места

В настоящее время игра не предлагает кроссплатформенную игру, что означает, что игроки на разных платформах не могут играть вместе. Однако игроки могут играть с другими на одной платформе, онлайн или локально.

В целом, Totally Reliable Delivery Service - это веселая и занимательная игра-симулятор, основанная на физике, в которую лучше всего играть с друзьями. Открытый мир, многопользовательский режим и низкие системные требования делают игру доступной для игроков на различных платформах и устройствах.



Рис. 1.1. Скриншот игры Totally Reliable Delivery Service

1. Route Me Mail and Delivery Co (RMMaD)

Route Me Mail and Delivery Co - это игра для доставки, разработанная TinyDino Games. Действие игры разворачивается в открытом 2D мире, где игроки берут на себя роль почтальона, доставляющего почту и посылки в различные места.

В игре нет многопользовательского режима. В игре представлены различные маршруты и режимы, включая новый режим под названием "Почтовая жизнь", который добавляет в игру основные боевые действия и новые карты.

Route Me Mail and Delivery Co доступна для покупки на платформе Windows через Steam. Цена игры составляет $9,99.

Минимальные системные требования для Windows:

ОС: Windows 7 или более поздняя версия

Процессор: Intel Core i3 2,4 ГГц или эквивалент

Оперативная память: 4 ГБ ОЗУ

Графика: NVIDIA GeForce 8800GT или эквивалент

DirectX: Версия 9.0c

Объем памяти: 1 ГБ свободного места

В целом, Route Me Mail and Delivery Co - это игра для доставки, которая предлагает игрокам множество маршрутов и режимов для изучения. Открытый мир и относительно низкие системные требования делают ее доступной для игроков на платформе Windows. Однако отсутствие многопользовательского режима и кроссплатформенности игры может быть недостатком для некоторых игроков.



Рис. 1.2. Скриншоты игры Route Me Mail and Delivery Co

1. Flying Neko Delivery (FND)

Flying Neko Delivery - это основанная на физике игра для доставки, разработанная Righteous Weasel Games. Действие игры разворачивается в красочном и причудливом мире, где игроки берут на себя роль летающего кота, доставляющего посылки в различные места.

В игре нет многопользовательского режима. В игре представлено множество уровней, каждый с различными задачами и препятствиями для преодоления.

Flying Neko Delivery доступна для покупки на нескольких платформах, включая Windows, Mac и Linux. Цена игры составляет 9,99 долларов.

Минимальные системные требования для Windows:

Операционная система: Windows 7 или более поздняя версия

Процессор: Intel Core 2 Duo 2,0 ГГц или эквивалент

Оперативная память: 2 ГБ ОЗУ

Графика: NVIDIA GeForce 8800GT или эквивалент

DirectX: Версия 9.0c

Объем хранилища: 500 МБ свободного места

В настоящее время игра не поддерживает кроссплатформенную игру, что означает, что игроки на разных платформах не могут играть вместе. Однако игроки могут играть с другими на той же платформе, онлайн или локально.

В целом, Flying Neko Delivery - это основанная на физике игра для доставки, которая предлагает игрокам множество уровней и задач для изучения. Причудливый мир и относительно низкие системные требования делают ее доступной для игроков на нескольких платформах. Однако отсутствие многопользовательского режима игры может быть недостатком для некоторых игроков.



Рис. 1.3. Скриншот игры Flying Neko Delivery

1. Lake

Lake - приключенческая игра, основанная на повествовании, разработанная Gamious. Действие игры разворачивается в маленьком городке Провиденс-Оукс, где игроки берут на себя роль Мередит Вайс, инженера-программиста, которая возвращается в свой родной город, чтобы на две недели взять на себя обязанности по доставке почты своего отца.

В игре присутствует среда открытого мира, позволяющая игрокам исследовать город и взаимодействовать с его жителями. Однако в игре нет многопользовательского режима.

Lake доступен для покупки на нескольких платформах, включая Windows, Mac и Nintendo Switch. Цена игры составляет $19,99.

Минимальные системные требования для Windows:

Операционная система: Windows 7 или более поздняя версия

Процессор: Intel Core i3 2,0 ГГц или эквивалент

Оперативная память: 4 ГБ ОЗУ

Графика: NVIDIA GeForce 560 или эквивалент

DirectX: Версия 11

Объем хранилища: 5 ГБ свободного места

В целом, Lake - это приключенческая игра, основанная на повествовании, которая предлагает игрокам возможность исследовать небольшой городок и делать выбор, влияющий на исход истории. Открытый мир и относительно низкие системные требования делают ее доступной и приятной для игроков на нескольких платформах. Однако отсутствие многопользовательского режима игры может быть недостатком для некоторых игроков.



Рис. 1.4. Скриншот игры Lake

1. Cloudpunk

Cloudpunk - приключенческая игра в стиле киберпанк, разработанная ION Lands. Действие игры разворачивается в городе Нивалис, где игроки берут на себя роль Нарнии, нового водителя доставки для Cloudpunk, компании по доставке, которая работает в подземном мире города.

В игре присутствует среда открытого мира, позволяющая игрокам исследовать город и его различные районы, выполняя миссии по доставке. Однако в игре нет многопользовательского режима.

Cloudpunk доступен для покупки на нескольких платформах, включая Windows, PlayStation 4, Xbox One и Nintendo Switch. Цена игры варьируется в зависимости от платформы, но обычно составляет от 24,99 до 29,99 долларов.

Минимальные системные требования для Windows:

Операционная система: Windows 7 или более поздняя версия

Процессор: Intel Core i3-3220 или эквивалент

Оперативная память: 4 ГБ RAM

Графика: NVIDIA GeForce GTX 560 или эквивалент

DirectX: Версия 11

Объем памяти: 6 ГБ свободного места

В целом, Cloudpunk - это приключенческая игра в стиле киберпанк, которая предлагает игрокам возможность исследовать огромный город и делать выбор, влияющий на исход истории. Открытый мир и относительно низкие системные требования делают ее доступной и приятной для игроков на нескольких платформах. Однако отсутствие многопользовательского режима игры может быть недостатком для некоторых игроков.



Рис. 1.5. Скриншот игры Cloudpunk

## Сводная таблица оценки

Таблица 1. Сводная таблица оценки аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | TRDS | RMMaD | FND | Lake | Cloudpunk |
| 1. Открытый мир | 10/10 | 08/10 | 10/10 | 10/10 | 10/10 |
| 2. Мультиплеер | 10/10 | 0/10 | 0/10 | 0/10 | 0/10 |
| 3. Цена | 05/10 | 09/10 | 07/10 | 07/10 | 07/10 |
| 4. Минимальные требования | 06/10 | 10/10 | 08/10 | 07/10 | 06/10 |
| 5. Кроссплатформенность | 07/10 | 02/10 | 05/10 | 05/10 | 07/10 |
| Итог | 7,6/10 | 5,8/10 | 6/10 | 5,8/10 | 6/10 |

## Требования к проекту

В связи с выставленными оценками и учётом всех позитивных и негативных аспектов каждого конкурента нашего проекта, предлагаем следующие решения:

Таблица 2. Требования к проекту

|  |  |
| --- | --- |
| 1. Открытый мир | Проект Postman Life подразумевает открытый мир изначально. Он будет огромным, разнообразным и полным живых существ. По этому фактору мы сможем дать отпор конкурентам благодаря движку - Godot Engine, в который предустановлена отличная оптимизация 3D пространства, что даст больше свободы в строительстве игрового мира. |
| 2. Цена | Наша игра будет практически бесплатной, как минимум - дешевле чем у конкурентов. |
| 3. Минимальные требования | Благодаря новому движку и его инструментарию есть широкий функционал оптимизации игры для слабых ПК. |
| 4. Кроссплатформенность | Godot Engine - среда разработки проекта Postman Life позволяет экспортировать его практически на любую платформу: Android, IOS, Web, Windows, VR. Проект Postman Life будет портирован на: Android и Windows. |
| Порт на IOS и VR не будет осуществлен, так как у команды разработки нет устройств для тестирования. |
| Порт на Web не будет осуществлен, так как это потребует создания параллельного проекта с перебором функционала, что не вписывается в график разработки. |

## Задачи

Ведущий разработчик (Дель Александр Андреевич)

Задача:

Создание и внедрение скриптов в проект игры «Жизнь почтальона» для оживления виртуальных сцен.

Шаги выполнения:

1. Логика живых объектов

* Создать заготовки живых объектов
* Реализовать функционал
* Подключить к общей мировой системе

1. Создание скриптов окружения:

* Создать заготовки объектов взаимодействия
* Реализовать функционал игрового окружения
* Связать скрипт окружения с общемировой системой

1. Создание скриптов объектов взаимодействия:

* Создать заготовки объектов взаимодействия
* Реализовать функционал объектов взаимодействия
* Наладить связь между объектами взаимодействия, где это требуется

1. Оптимизация игрового мира

* Найти проблемные области кода
* Измерить влияние и существенность проблемы на игровой мир и его системы
* Оптимизировать проблемную область кода

1. Руководство Разработчика

# Логика живых объектов

Список всех живых объектов с общим описанием функционала приведен в таблице ниже.

Таблица 1. Сущности игры

|  |  |
| --- | --- |
| Сущность | Функционал |
| НЛО | Летает, может украсть и уничтожить посылку |
| Чайка | Летает в воздухе, приземлиться не может. С определенным шансом может украсть посылку, находящуюся вне инвентаря игрока |
| Корова | Мычит |
| Собака | Лает |
| Курица | Орет |
| Призрак | Воспроизводит скример для игрока |
| Гигантское НЛО | Появляется раз в 10 дней, издает громкий звук |

## Создание заготовки живых объектов

Создание заготовки живого объекта включает в себя добавление модели в игру и его базовая настройка: настройка коллайдеров (область столкновений), рейкастов (по сути - зрение объекта), источники звука, таймеры и прочее. Для каждой сущности эти параметры будут индивидуальными. К примеру, сущность «Птица» имеет следующие базовые настройки (Рис. 2.1): модель, коллайдер, таймер, источники звука.

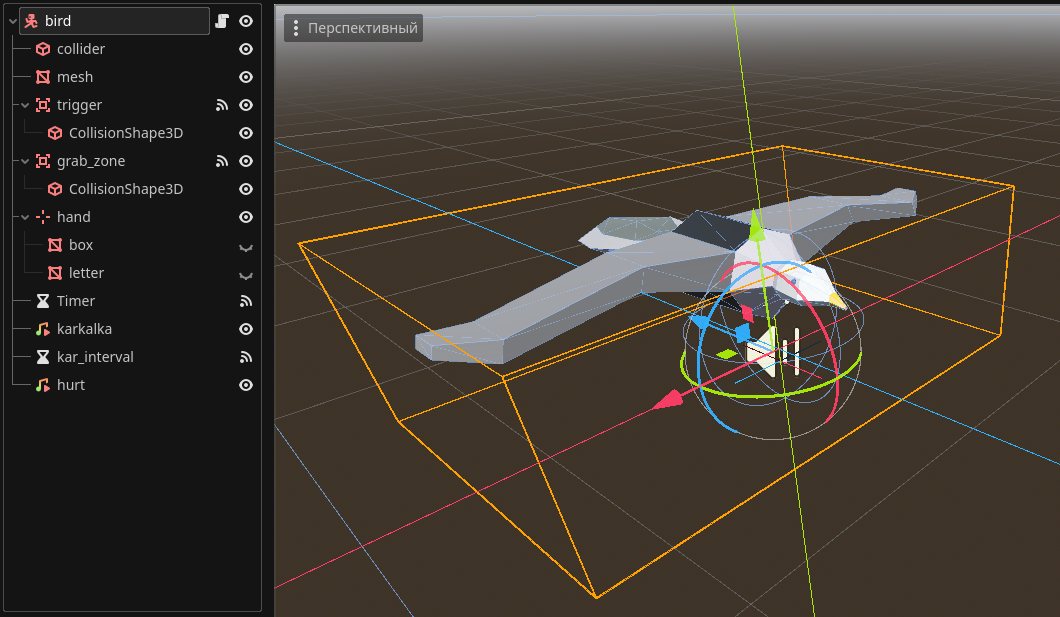


Рис. 2.1. Заготовка сущности - «Птица»

## Реализация функционала

Была написана логика всех живых объектов для игры «Жизнь почтальона». Реализация игрового функционала всех сущностей приведена в приложении в соответствующих скриптах. Реализация функционала включает в себя:

* Реализация появления и исчезновения сущностей в процессе игры;
* Реализация перемещения сущностей (по необходимости);
* Реализация звуков сущностей, в том числе вариативность;
* Балансировка количества сущностей.

## Подключение к мировой системе

Была написана логика подключения всех живых объектов к мировой системе для игры «Жизнь почтальона». Всего подобным образом было разработано 7 различных живых существ: Собака, Корова, Курица, НЛО, Гигантское НЛО, Призрак, Чайка.

Подключение сущностей к мировой системе включает в себя:

* Реализация появления сущностей в игровом мире;
* Реализация взаимодействия сущностей с игровым миром;
* Реализация взаимодействия сущностей с игроком.

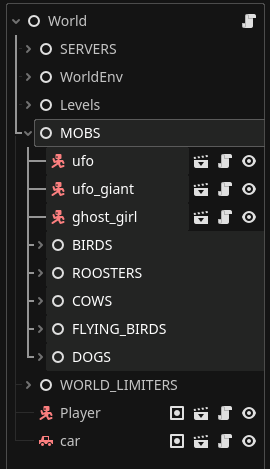


Рис. 2.2. Подключенные сущности к игровому миру

# Создание скриптов окружения

Игровое окружение — это очень важная составляющая любого проекта. Именно оно создает игровую атмосферу. С помощью окружения можно умело создавать разнообразные ситуации, погружающие игрока в виртуальный мир. Окружение — это своеобразное тело игры, в которое помещены персонажи.

## [Создание заготовки игрового окружения](#_Toc22522)

На Рис. 3.1 изображена заготовка окружения в игре «Жизнь почтальона». Заготовка игрового окружение подразумевает готовый игровой мир с точки зрения подключения неба, соответствующих шейдеров и анимации.

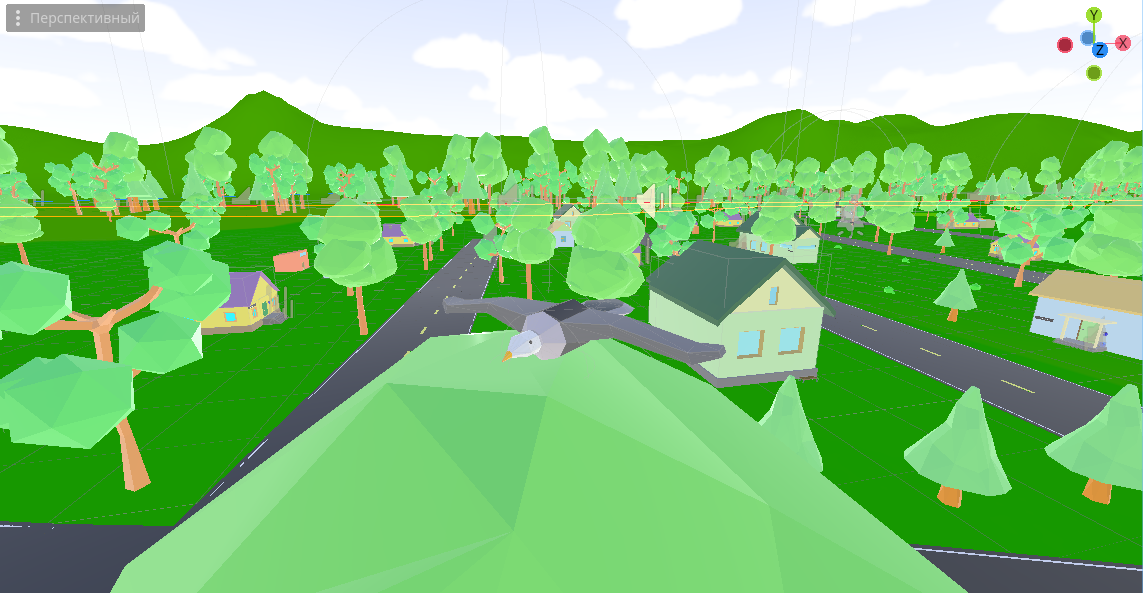


Рис. 3.1. Заготовка окружения в игре «Жизнь почтальона»

## [Реализация функционала игрового окружения](#_Toc21729)

На Рис. 3.2 отражен главный функционал окружения в игре «Жизнь почтальона» - смена дня и ночи, в виде анимации. Смена дня и ночи происходит постепенно в течении дня и является бесшовной, то есть игрок не заметит перехода между временем суток и сменой анимации.

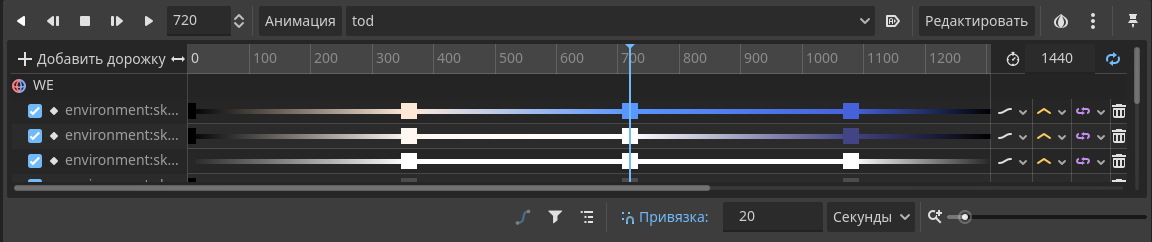


Рис. 3.2. Функционал окружения в игре «Жизнь почтальона»

## [Создание скрипта окружения с общемировой системой](#_Toc6105)

Была написана логика окружения для игры «Жизнь почтальона». Она включает в себя счет и отображение игрового времени и погода, а также изменение окружения в зависимости от них.



Рис. 3.3. Код окружения в игре «Жизнь почтальона»

# Создание скриптов объектов взаимодействия

## [Создание заготовок объектов взаимодействия](#_Toc13646)

Заготовка игрового объекта включает в себя добавление в проект модели или ее создание, а также нанесение на нее коллайдеров. Для каждого объекта заготовка будет представлять индивидуальный набор параметров. Для примера, так выглядит заготовка подбираемого предмета - «Молочка»:

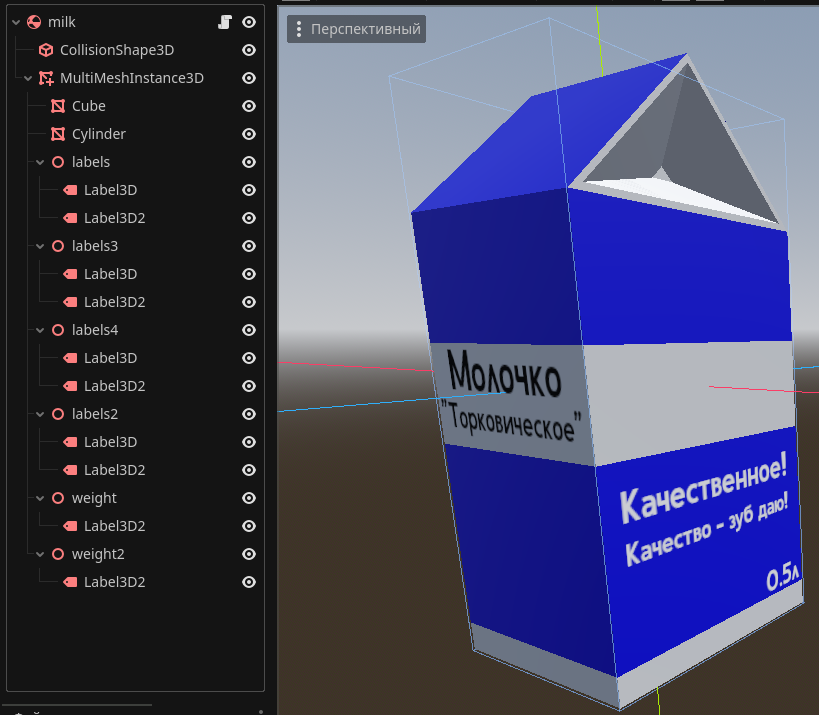


Рис. 4.1. Заготовка «Молочка» в игре «Жизнь почтальона»

## [Реализаия функционала объектов взаимодействия](#_Toc5762)

Реализация функционала объектов взаимодействия включает в себя следующие параметры:

* Описание параметров, таких как id, вес и имя;
* Описание процедуры взаимодействия (взятия).

Для каждого из объектов функционал будет индивидуальным. Простейший описан для объекта молочко:

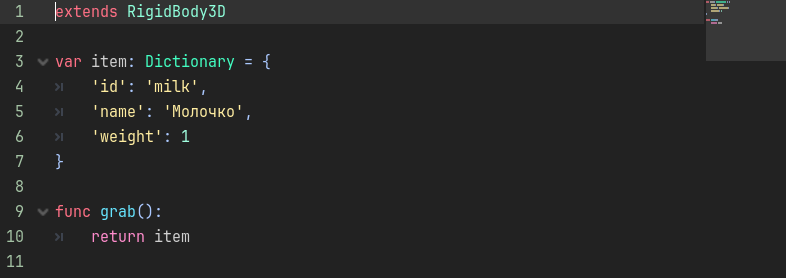


Рис. 4.2. Реализация функционала объекта «Молочко»

Пример реализации более сложного функционала для объекта «Мина»:

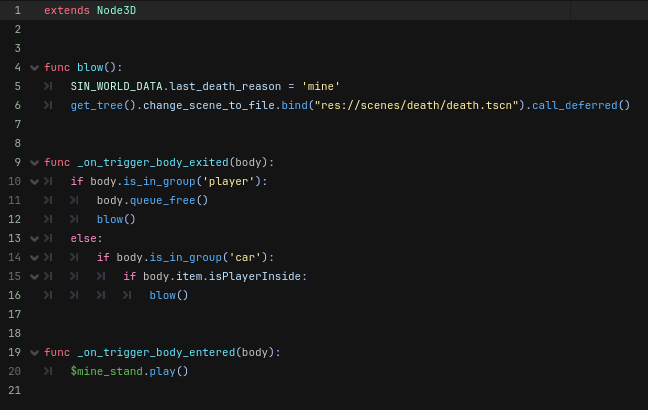


Рис. 4.3. Реализация функционала объекта «Мина»

## [Налаживание связи между объектами взаимодействия, где это требуется](#_Toc20573).

Налаживание связи подразумевает использование сигналов, которые позволяют из одного участка кода обращаться к другому, при условии что они никак не связаны (Рис. 4.3):

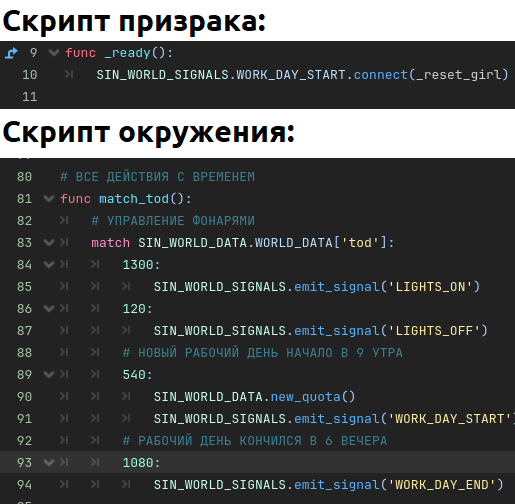


Рис. 4.4. Связь призрака с окружением в игре «Жизнь почтальона»

# Оптимизация игрового мира

## [Поиск проблемных областей кода](#_Toc6135)

В ходе разработки игры было создано и позже, обнаружено огромное множество неэффективного кода. В процессе оптимизации игрового мира эти проблемные области кода были исправлены.

Примером неэффективного кода может служить ошибка №121, отраженная на Рис. 5.1.

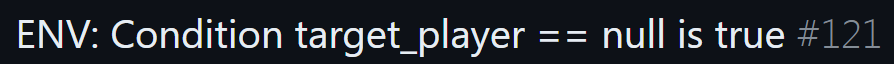


Рис. 5.1. Пример ошибки, отражающей неэффективный код

## [Измерение влияния и существенности проблемы на игровой мир и его системы](#_Toc30307)

Измерение существенности влияния ошибки (Рис. 5.2).

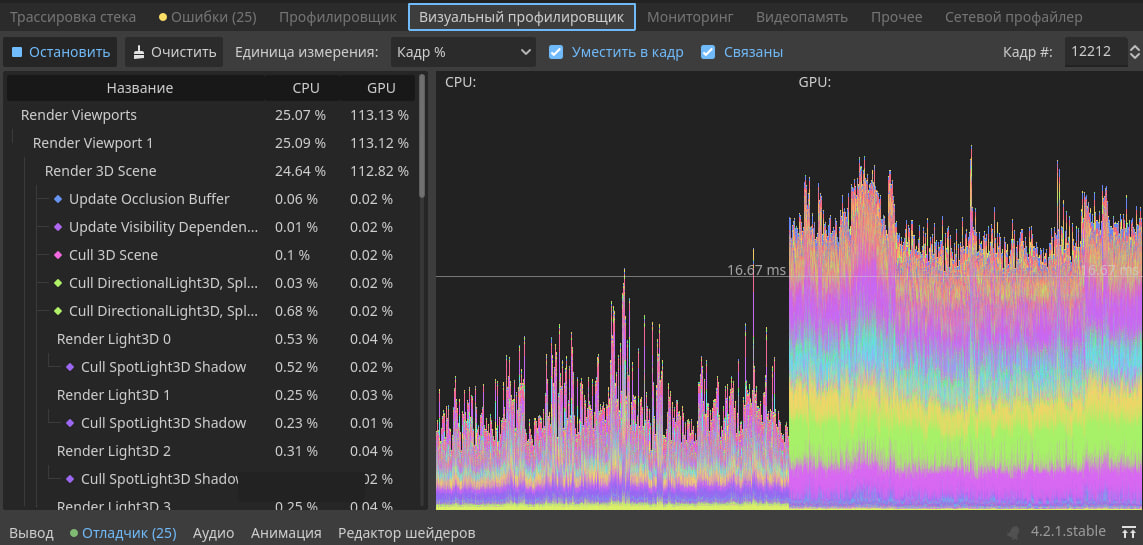


Рис. 5.2. Измерение существенности влияния ошибки

## [Оптимизация проблемной части кода](#_Toc20508)

Исправлением неэффективного кода стало избавление логики от постоянного обновления переменных и в целом удаления конечного автомата (Рис. 5.3).

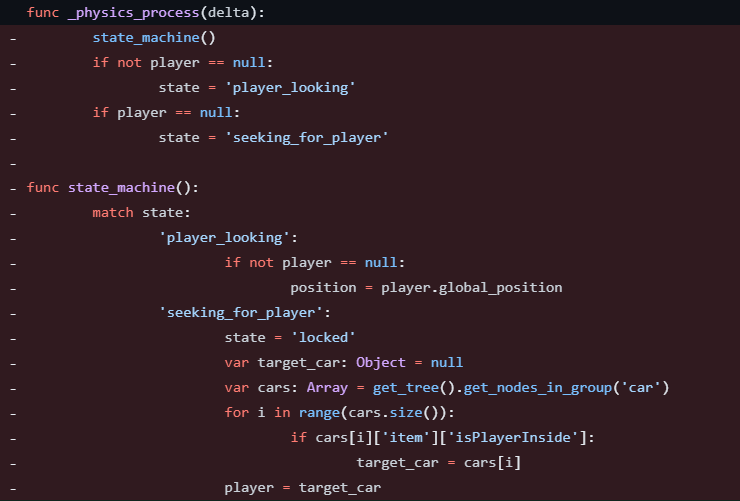


Рис. 5.3. Пример решения ошибки, исправления неэффективного кода

# Руководство программиста

При загрузке приложение появляется главное меню (Рис. 6.1), из которого пользователь может перейти в саму игру, ее настройки или выйти из игры.



Рис. 6.1. Меню приложения

Таким образом это окно выглядит в окне иерархии узлов. Существуют четыре кнопки, три из которых видны всегда, а одна дополнительная «Продолжить» - появляется только при наличии сохранения.



Рис. 6.2. Меню приложения - иерархия узлов

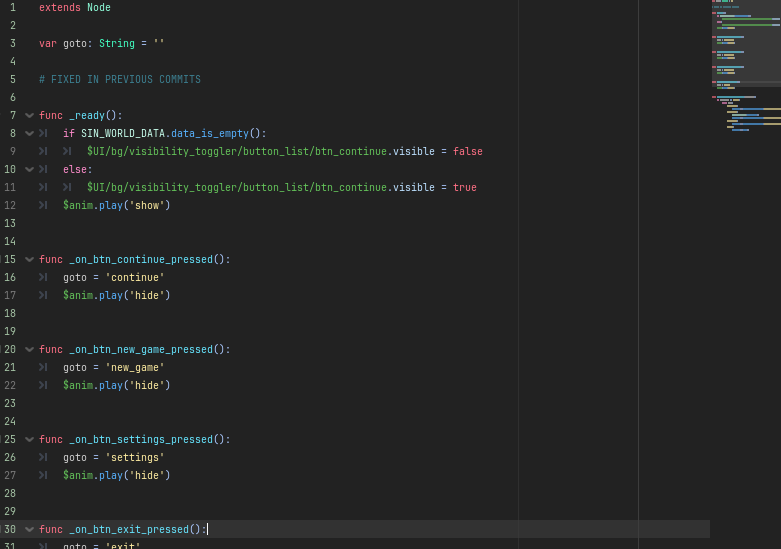


Рис. 6.3. Меню приложения - код

Код связывает сигналы кнопок с их логикой. Так, нажимая на каждую из них меняется переменная goto, по сути является id окна для перехода.

## Игровые настройки

При переходе в настойки у игрока появляется возможность детально настроить графику и звуки под свои потребности. Для возврата в меню необходимо нажать кнопку «Назад». В настройках есть 3 раздела: «Аудио», «Графика» и «Дополнительно». Для перемещения между разделами необходимо воспользоваться меню слева. Все настройки сохраняются и применяются автоматически при нажатии кнопки «Назад».

По умолчанию при переходе в настройки, открывается раздел «Аудио» (Рис. 6.4). Здесь пользователь может настроить общую громкость звуков и отдельно громкость музыки.

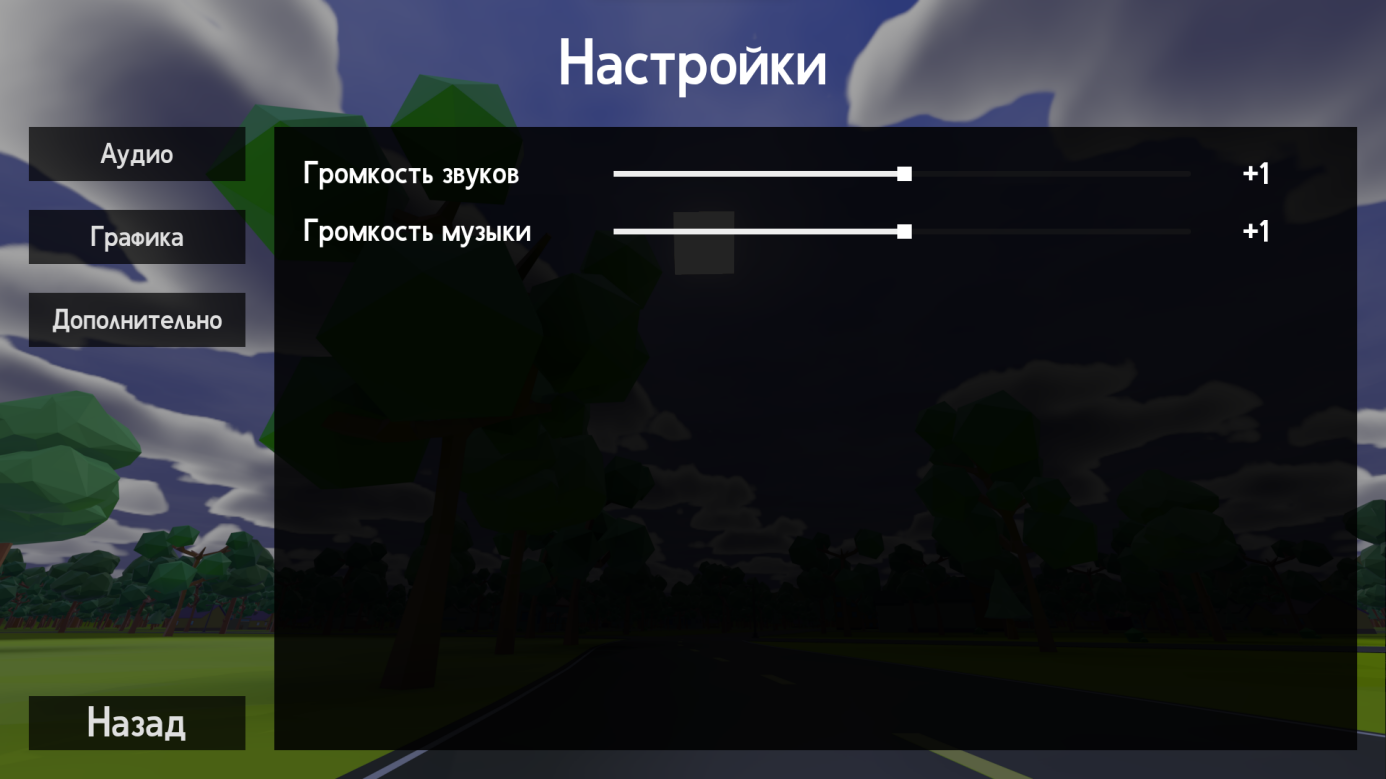


Рис. 6.4. Раздел «Аудио» в настройках приложения

Таким образом выглядит окно «Настройки» в иерархии узлов (Рис. 6.5).

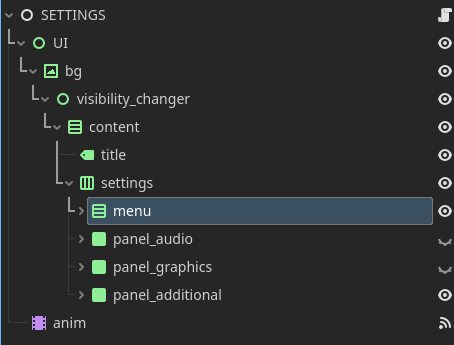


Рис. 6.5. Настройки - иерархия узлов

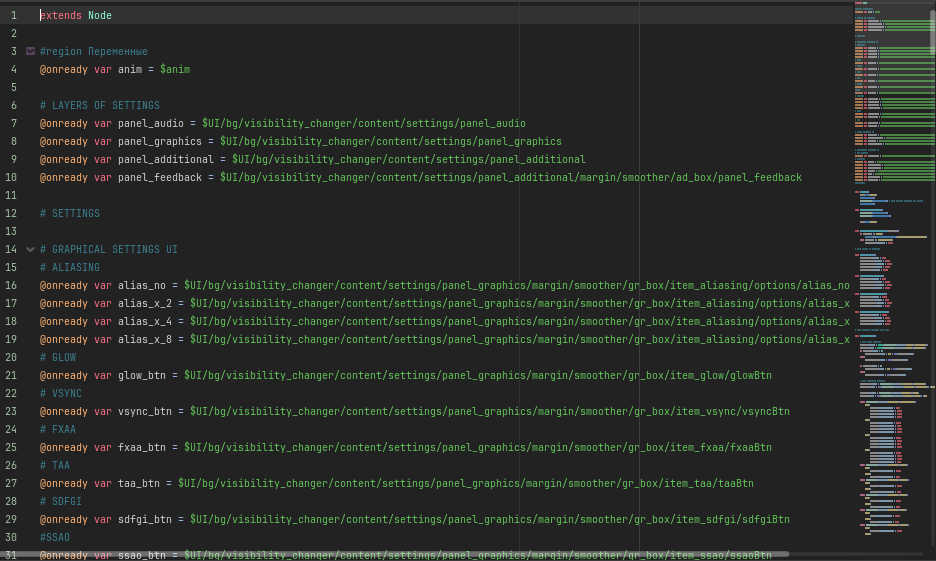


Рис. 6.6. Настройки Аудио - код

В разделе «Графика» (Рис. 6.7) у пользователя есть возможность детально настроить внутриигровую графику. Увеличение значений настроек может способствовать понижению производительности и FPS внутри игры.

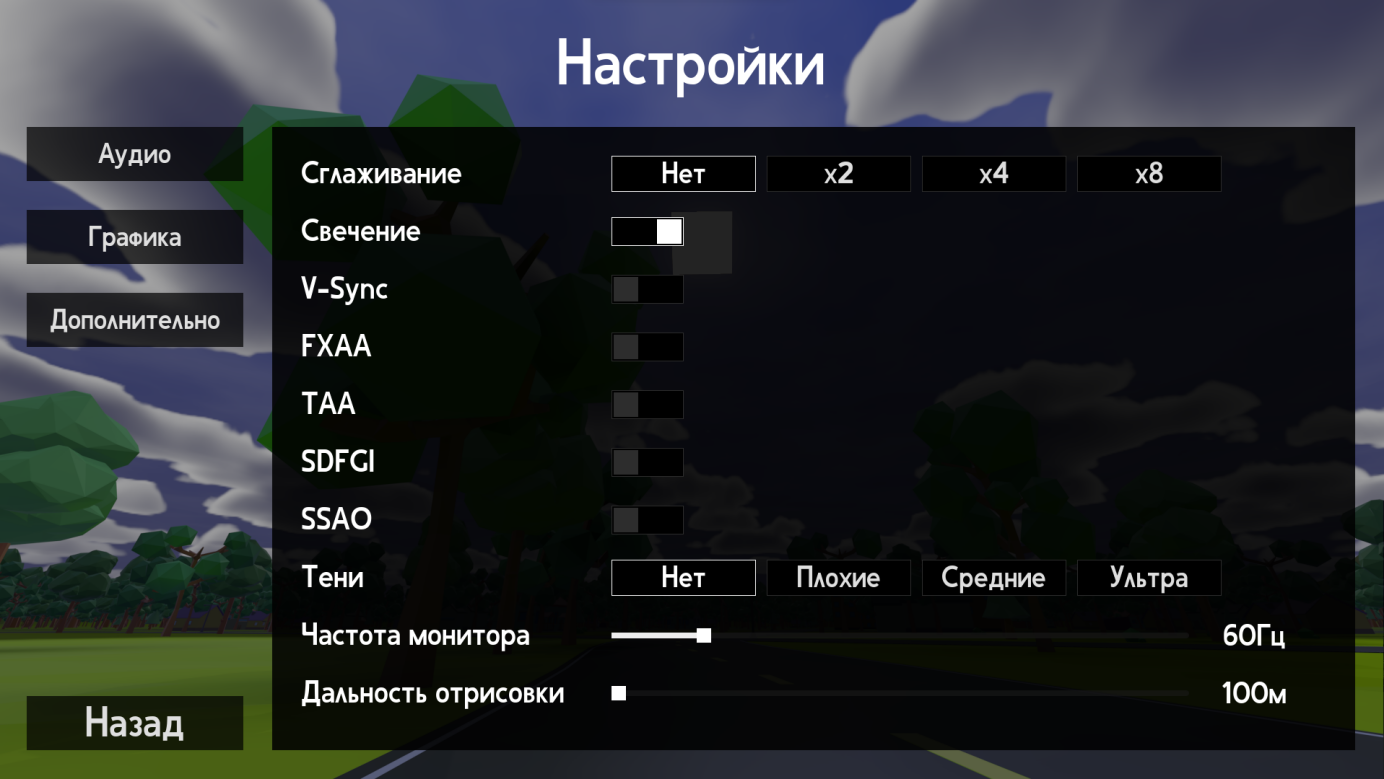


Рис. 6.7. Раздел «Графика» в настройках приложения

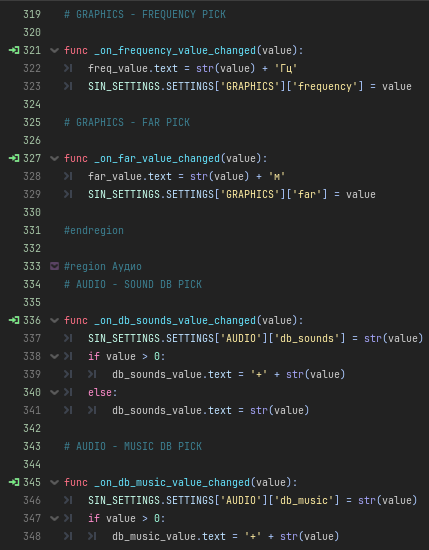


Рис. 6.8. Настройки Графики - код

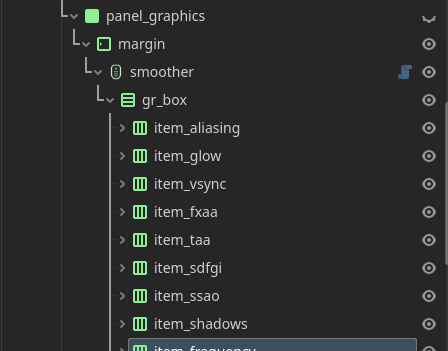


Рис. 6.9. Настройки Графики - иерархия узлов

В разделе «Дополнительно» (Рис. 6.10) есть возможность включить счетчик кадров (FPS) и отправить обращение разработчикам через специальную форму (Рис. 6.13). Для отправки обращения необходимо описать проблему или предложение в поле ввода и нажать кнопку «Отправить». Если пользователь хочет получить обратную связь - он может ввести данные для обратной связи в отдельном поле, до отправки обращения. Отправить пустое обращение нельзя.

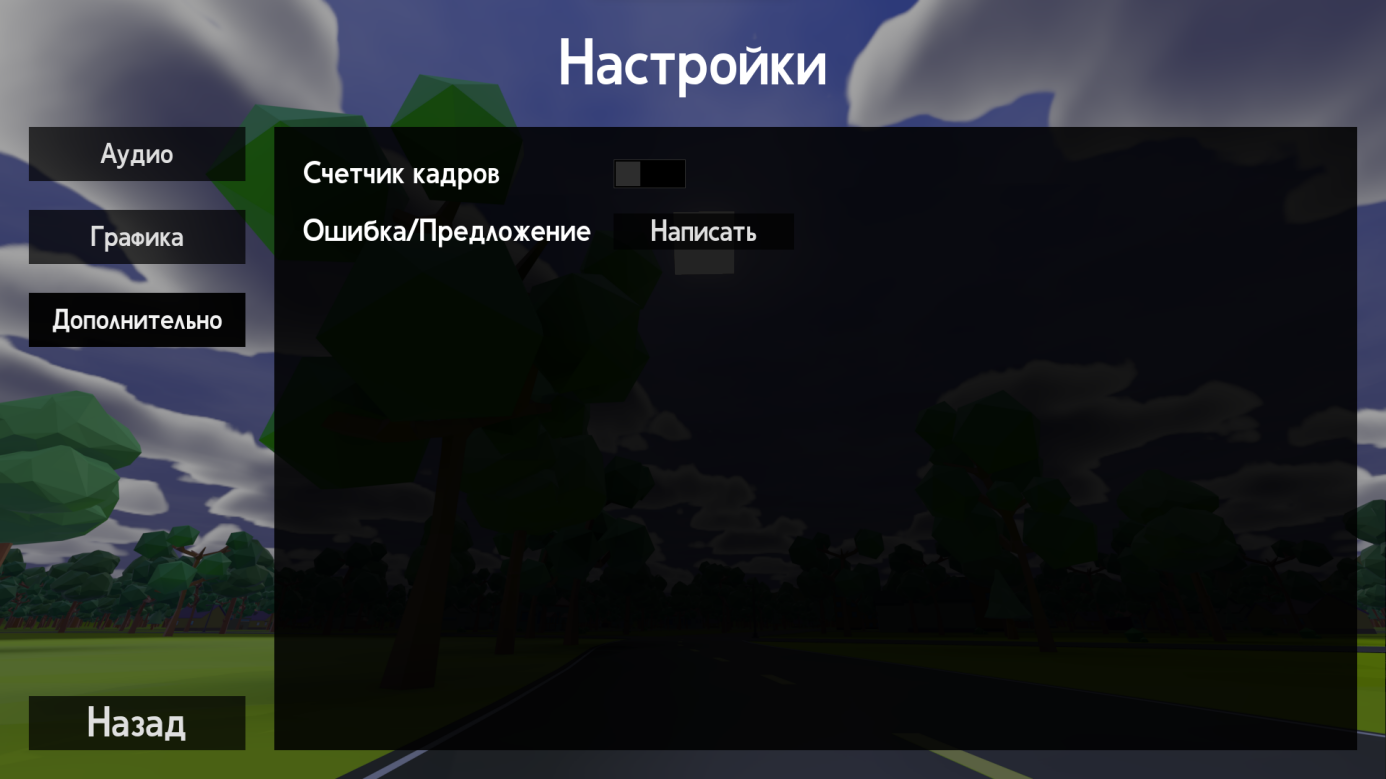


Рис. 6.10. Раздел «Дополнительно» в настройках приложения

Вот так выглядит окно настроек «Дополнительно» в окне иерархии узлов (Рис 3.5).

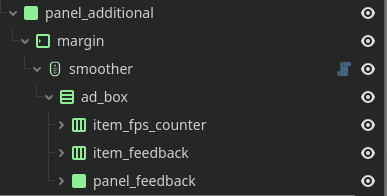


Рис. 6.11. Настройки Графики - иерархия узлов

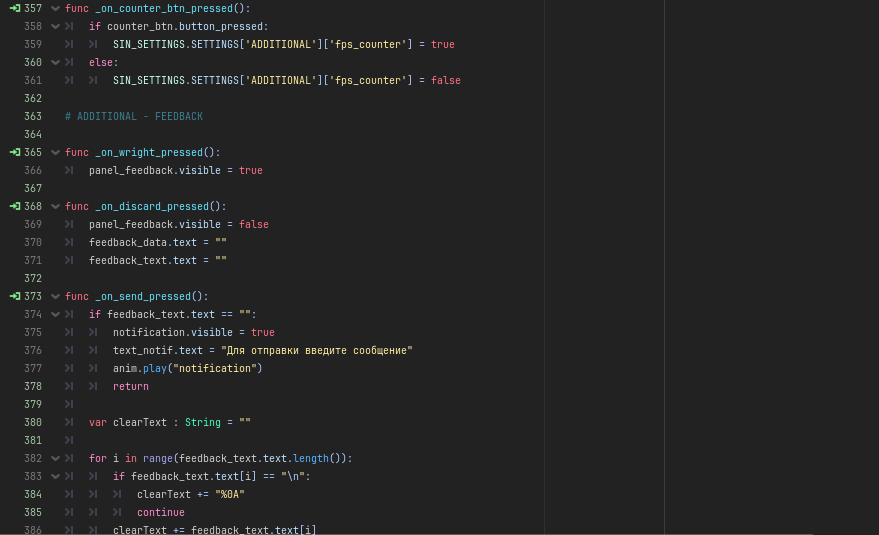


Рис. 6.12. Настройки «Дополнительно» - код

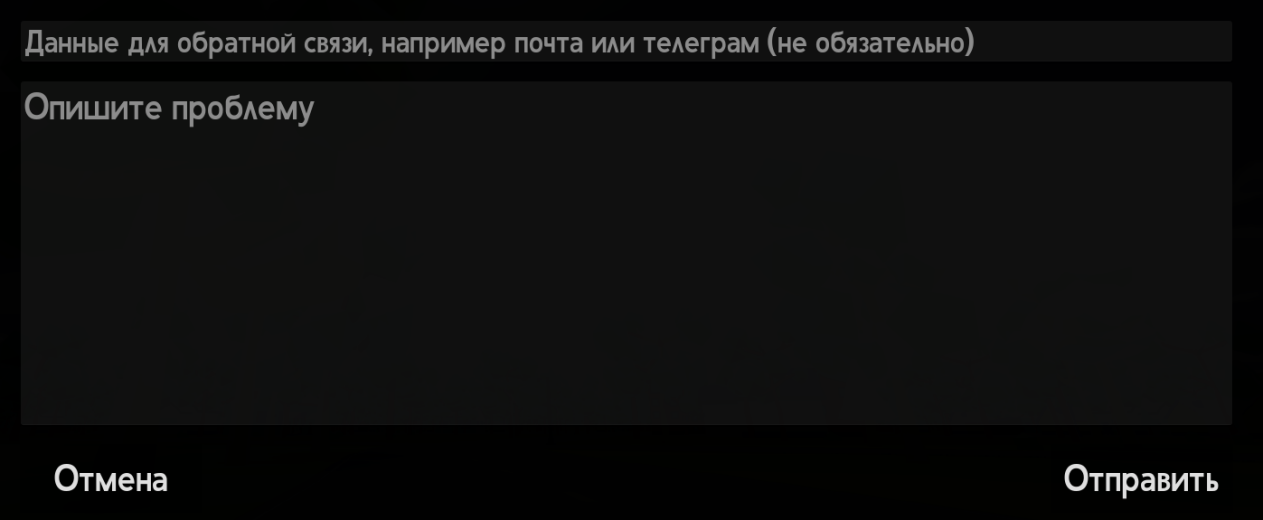


Рис. 6.13. Форма отправки обращения к разработчикам

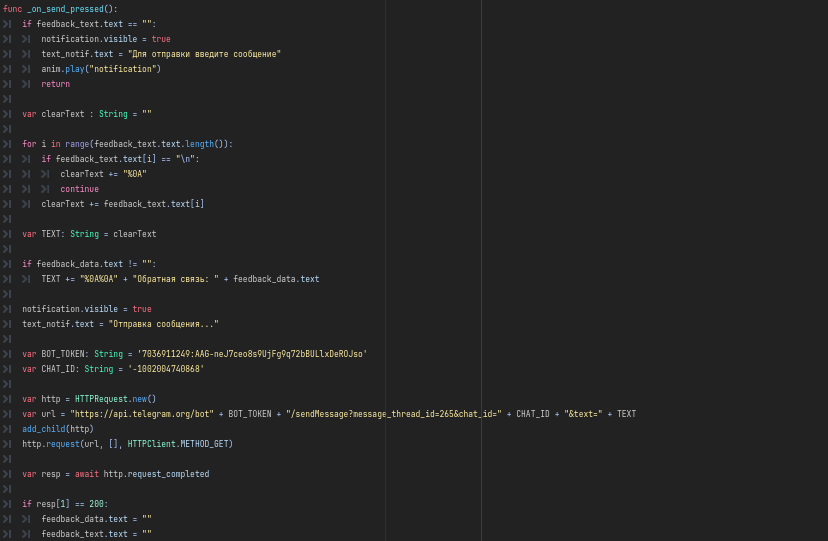


Рис. 6.14. Форма отправки обращения к разработчикам - код

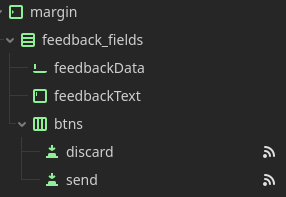


Рис. 6.15. Форма отправки обращения к разработчикам - иерархия узлов

Все настройки графики, аудио и дополнительные - сохраняются в отдельный файл- settings.dat (Рис. 6.16).



Рис. 6.16. Файл, содержащий настройки

В коде сохранение настроек выглядит следующим образом (Рис. 6.17).

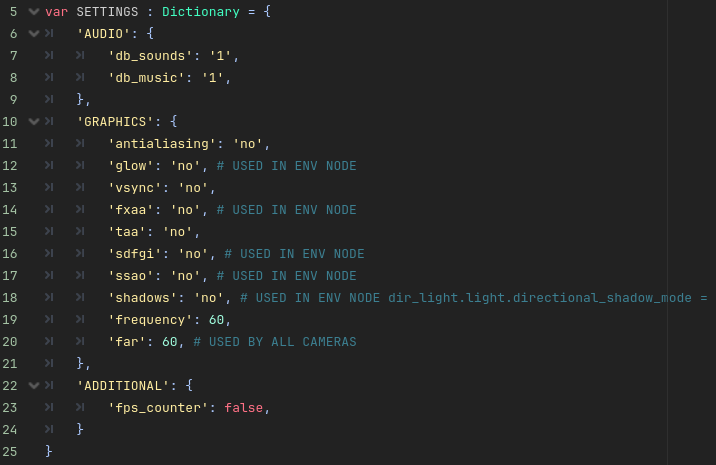


Рис. 6.17. Словарь значений файла «Настройки»

## Игра

Для перехода в саму игру пользователь, находясь в главном меню, должен нажать кнопку «Новая игра» или «Продолжить», если у него уже имеется сохранение. После нажатия на кнопку, пользователь переходит на экран загрузки (Рис. 6.18), который автоматически переключиться в игровой мир, после окончания настройки и загрузке всех ресурсов. Длительность загрузки зависит от производительности устройства.



Рис. 6.18. Экран загрузки игры

Функциональный код загрузочного экрана выглядит следующим образом (Рис. 6.19). В коде реализованы подсказки, в виде случайного элемента взятого из массива и динамическая подгрузка требуемого файла сцены, с помощью функции \_process().

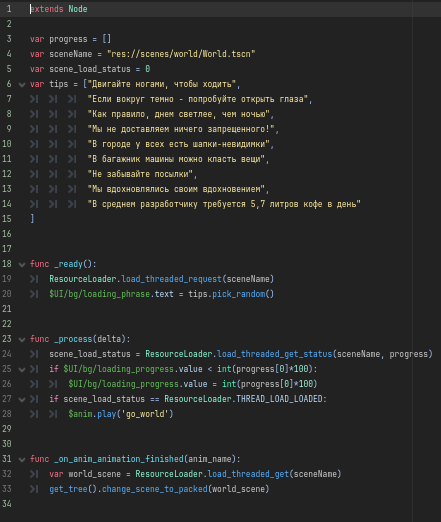


Рис. 6.19. Экран загрузки игры - Код

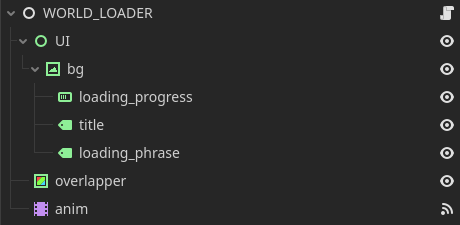


Рис. 6.20. Экран загрузки игры - Иерархия узлов

## Диаграммы UML

Диаграмма USE-CASE:

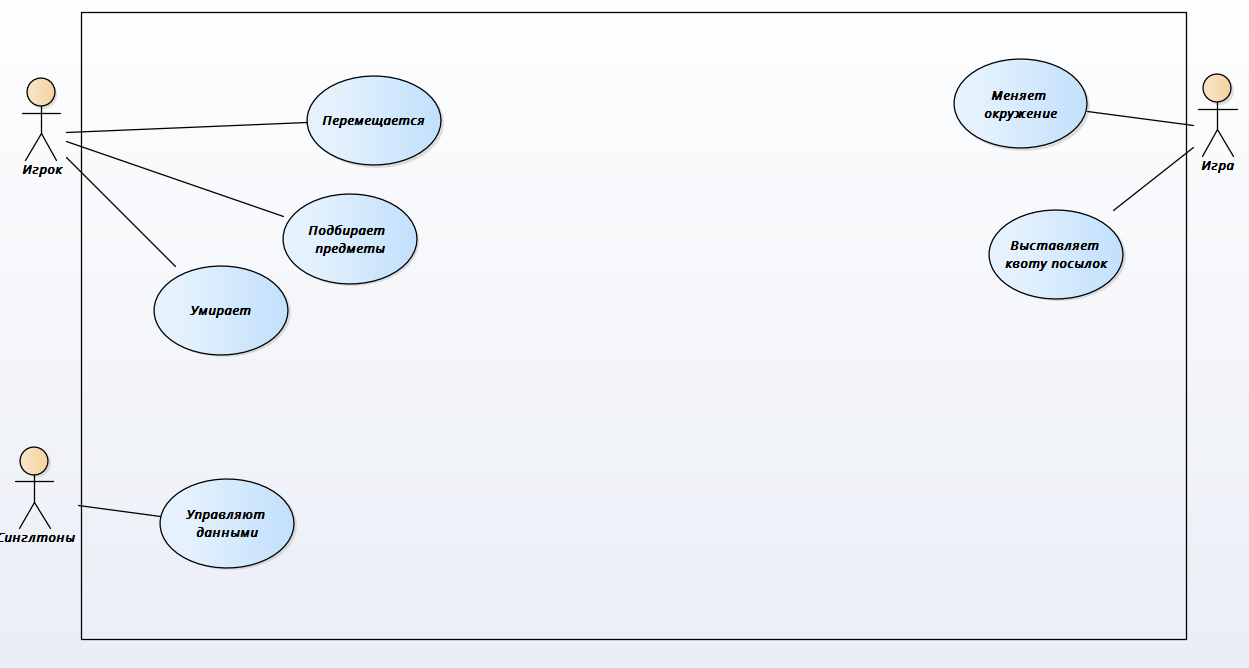


Рис. 6.21. Диаграмма вариантов использования

Диаграмма классов:

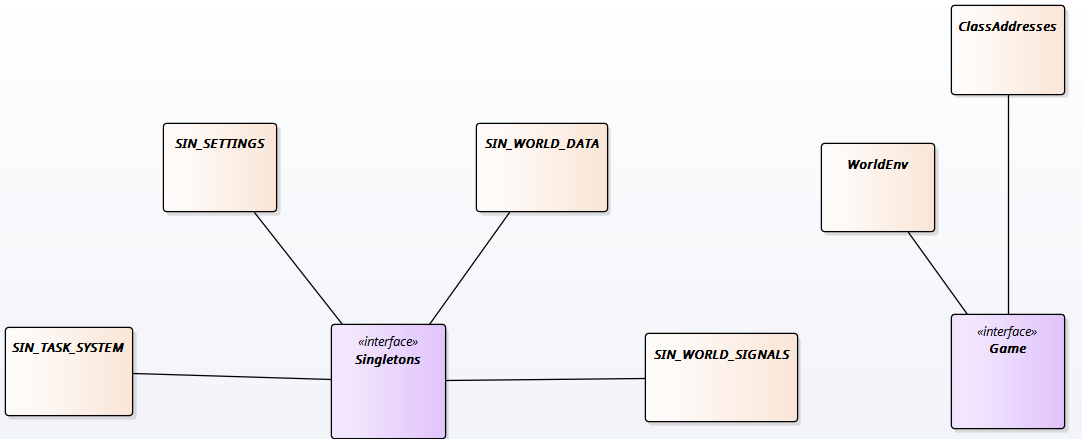


Рис. 6.22. Диаграмма классов и интерфейсов

Диаграмма компонентов:

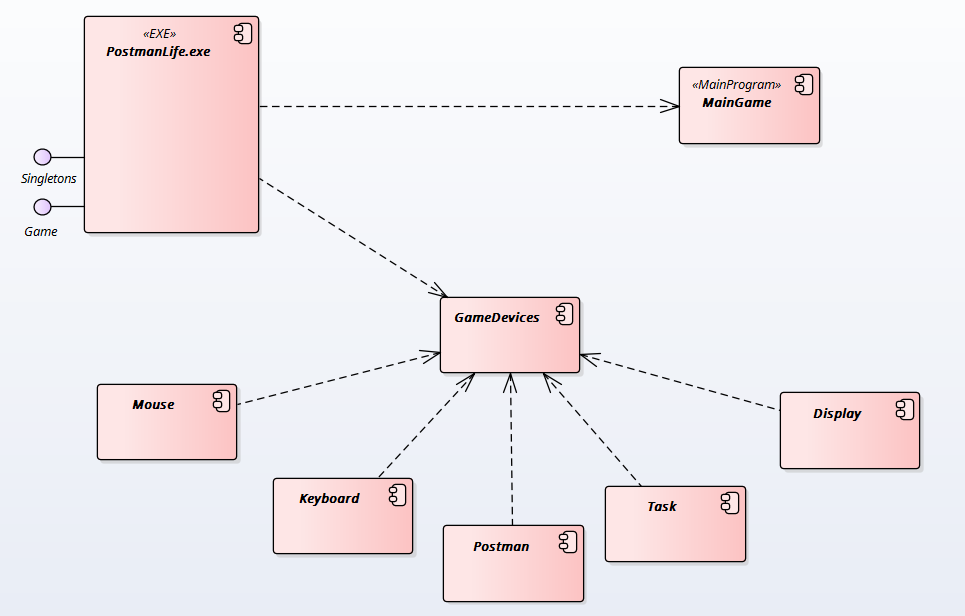


Рис. 6.23. Диаграмма компонентов

Диаграмма бизнес-логики доставки посылки:

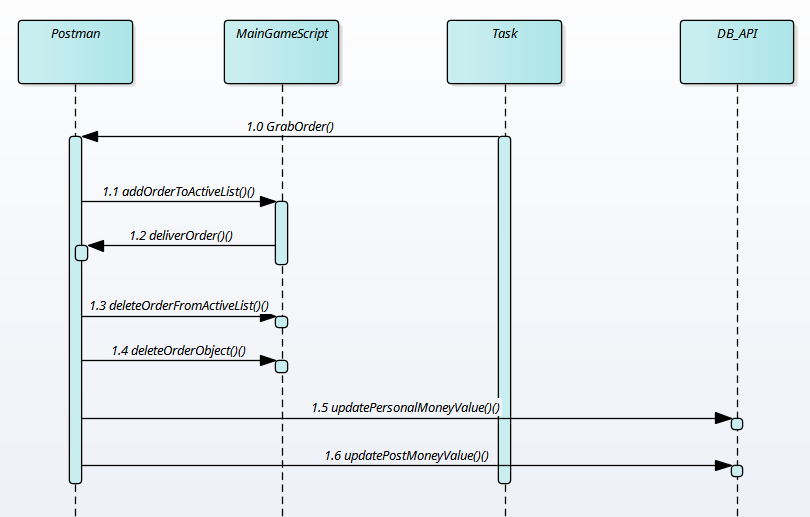


Рис. 6.24. Диаграмма бизнес-логики доставки посылки

Диаграмма бизнес-логики управления персонажем:

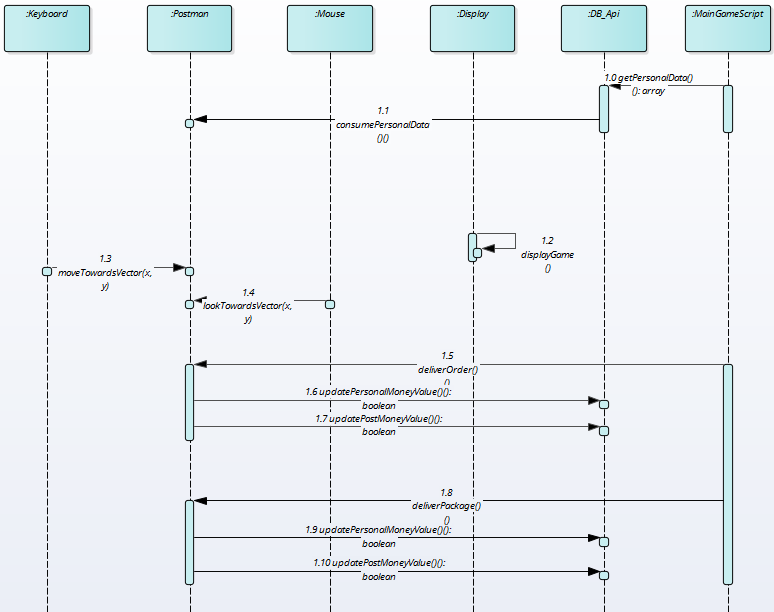


Рис. 6.25. Диаграмма бизнес-логики управления персонажем

# Заключение

В результате работы над расчетно-графической работой была создана игра жанра «Симулятор» - «Жизнь почтальона». Игра имеет дружественный интерфейс, широкие настройки графики и стабильную производительность.

Стоит отметить, что в силу сроков на данный момент в приложении реализован не весь задуманный функционал, а только базовый и наиболее важный. Тем не менее, этого хватает для полноценной игры.

В ходе работы на расчетно-графической работой были укреплены знания по работе с системой управления приложением и контроля версий - GitHub для организации и контроля процесса разработки.

Также были получены навыки работы со средой разработки Godot Engine и средой построения UML-диаграмм Enterprise Architect

# Приложение

**ClassAddresses.gd**

class\_name ClassAddresses

var addresses: Dictionary = {

# Delivery ID | User-friendly name

# Уникальный ID должен совпадать с ID ящика

'00001': 'Первая Железнодорожная, д1',

'00002': 'Первая Железнодорожная, д2',

'00003': 'Первая Железнодорожная, д3',

'00004': 'Первая Железнодорожная, д4',

'00005': 'Первая Железнодорожная, д5',

'00006': 'улица Круговая, д1',

'00007': 'улица Круговая, д2',

'00008': 'улица Круговая, д3',

'00009': 'улица Круговая, д4',

'00010': 'улица Круговая, д5',

'00011': 'Вторая Железнодорожная, д1',

'00012': 'Вторая Железнодорожная, д2',

'00013': 'Вторая Железнодорожная, д3',

'00014': 'Вторая Железнодорожная, д4',

'00015': 'Вторая Железнодорожная, д5',

'00016': 'Вторая Железнодорожная, д6',

'00017': 'Вторая Железнодорожная, д7',

'00018': 'Вторая Железнодорожная, д8',

'00019': 'Третья Железнодорожная, д1',

'00020': 'Третья Железнодорожная, д2',

'00021': 'Третья Железнодорожная, д3',

'00022': 'Третья Железнодорожная, д4',

'00023': 'Третья Железнодорожная, д5',

'00024': 'Третья Железнодорожная, д6',

'00025': 'Четвертая Железнодорожная, д1',

'00026': 'Четвертая Железнодорожная, д2',

'00027': 'Четвертая Железнодорожная, д3',

'00028': 'Четвертая Железнодорожная, д4',

'00029': 'Четвертая Железнодорожная, д5',

'00030': 'Четвертая Железнодорожная, д6',

'00031': 'Тихий переулок, д1',

'00032': 'Тихий переулок, д2',

'00033': 'Тихий переулок, д3',

'00034': 'Тихий переулок, д4',

'00035': 'Дудоровская улица, д1',

'00036': 'Дудоровская улица, д2',

'00037': 'Дудоровская улица, д3',

'00038': 'Дудоровская улица, д4',

'00039': 'Дудоровская улица, д5',

'00040': 'Дудоровская улица, д6',

'00041': 'Дудоровская улица, д7',

'00042': 'Дудоровская улица, д8',

'00043': 'Дудоровская улица, д9',

'00044': 'Дудоровская улица, д10',

'00045': 'Дудоровская улица, д11',

'00046': 'Дудоровская улица, д12',

'00047': 'Дудоровская улица, д13',

'00048': 'Дудоровская улица, д14',

'00049': 'Дудоровская улица, д15',

'00050': 'Дудоровская улица, д16',

'00051': 'Дудоровская улица, д17',

'00052': 'Дудоровская улица, д18',

'00053': 'Высотная улица, д1, парадная 1',

'00054': 'Высотная улица, д1, парадная 2',

'00055': 'Высотная улица, д1, парадная 3',

'00056': 'улица Победы, д2, парадная 1',

'00057': 'улица Победы, д2, парадная 2',

'00058': 'улица Победы, д2, парадная 3',

'00059': 'Высотная улица, д2, парадная 1',

'00060': 'Высотная улица, д2, парадная 2',

'00061': 'Высотная улица, д2, парадная 3',

'00062': 'улица Победы, д1', # БОЛЬНИЦА

'00063': 'улица Славы, д1',

'00064': 'улица Славы, д2',

'00065': 'улица Вторая Гражданская, д1',

'00066': 'улица Вторая Гражданская, д2',

'00067': 'улица Вторая Гражданская, д3',

'00068': 'улица Вторая Гражданская, д4',

'00069': 'улица Советская, д1',

'00070': 'улица Первого Мая, д1',

'00071': 'улица Первого Мая, д2',

'00072': 'улица Первого Мая, д3',

'00073': 'улица Школьная, д1', # ШКОЛА

'00074': 'улица Школьная, д2',

'00075': 'улица Цветочная, д1',

'00076': 'улица Цветочная, д2',

'00077': 'улица Торговая, д1',

'00078': 'улица Торговая, д2',

'00079': 'улица Торговая, д3',

'00080': 'улица Торговая, д4',

'00081': 'улица Торговая, д5',

'00082': 'улица Торговая, д6',

'00083': 'улица Торговая, д7',

'00084': 'улица Торговая, д8',

'00085': 'улица Торговая, д9',

'00086': 'улица Первая Гражданская, д1',

'00087': 'улица Первая Гражданская, д2',

'00088': 'улица Первая Гражданская, д3',

'00089': 'улица Первая Гражданская, д4',

'00090': 'улица Первая Гражданская, д5',

'00091': 'улица Первая Гражданская, д6',

'00092': 'улица Первая Гражданская, д7',

'00093': 'улица Первая Крестьянская, д1',

'00094': 'улица Первая Крестьянская, д2',

'00095': 'улица Первая Крестьянская, д3',

'00096': 'улица Первая Крестьянская, д4',

'00097': 'улица Молодежная, д1',

'00098': 'улица Молодежная, д2',

'00099': 'улица Молодежная, д3',

'00100': 'улица Молодежная, д4',

'00101': 'улица Вторая Крестьянская, д1',

'00102': 'улица Вторая Крестьянская, д2',

'00103': 'улица Вторая Крестьянская, д3',

'00104': 'улица Вторая Крестьянская, д4',

'00105': 'улица Вторая Крестьянская, д5',

'00106': 'переулок Торговый, д1',

'00107': 'переулок Торговый, д2',

'00108': 'переулок Торговый, д3',

'00109': 'переулок Торговый, д4',

'00110': 'улица Набережная, д1',

'00111': 'улица Набережная, д2',

'00112': 'улица Набережная, д3',

'00113': 'улица Набережная, д4',

'00114': 'улица Набережная, д5',

'00115': 'улица Набережная, д6',

'00116': 'улица Набережная, д7',

'00117': 'улица Набережная, д8',

'00118': 'улица Набережная, д9',

'00119': 'улица Неизвестная, д1',

'00120': 'улица Полевая, д1',

'00121': 'улица Полевая, д2',

'00122': 'улица Полевая, д3',

'00123': 'улица Полевая, д4',

'00124': 'улица Полевая, д5',

'00125': 'переулок Подгорный, д1',

'00126': 'переулок Подгорный, д2',

'00127': 'переулок Подгорный, д3',

'00128': 'переулок Подгорный, д4',

'00129': 'переулок Подгорный, д5',

'00130': 'улица Садовая, д1',

'00131': 'улица Садовая, д2',

'00132': 'улица Садовая, д3',

'00133': 'улица Садовая, д4',

'00134': 'улица Садовая, д5',

'00135': 'улица Садовая, д6',

}

func get\_random\_delivery\_id():

var a = addresses.keys()

a = a[randi() % a.size()]

return a

**ClassScenePacker.gd**

extends StaticBody3D

func \_on\_area\_3d\_body\_entered(body):

if body.is\_in\_group('player'):

body.isOnLadder = true

body.in\_air\_momentum = false

func \_on\_area\_3d\_body\_exited(body):

if body.is\_in\_group('player'):

body.isOnLadder = false

body.in\_air\_momentum = true

**CommTower.gd**

extends StaticBody3D

func \_on\_area\_3d\_body\_entered(body):

if body.is\_in\_group('player'):

body.isOnLadder = true

body.in\_air\_momentum = false

func \_on\_area\_3d\_body\_exited(body):

if body.is\_in\_group('player'):

body.isOnLadder = false

body.in\_air\_momentum = true

**GasStation.gd**

extends Node3D

@export var petrol\_canister: PackedScene

@onready var info\_station\_1 = $petrol\_station\_1/info

@onready var info\_station\_2 = $petrol\_station\_2/info

@onready var info\_station\_3 = $petrol\_station\_3/info

@onready var info\_station\_4 = $petrol\_station\_4/info

var petrol\_station\_1: int = 0

var petrol\_station\_2: int = 0

var petrol\_station\_3: int = 0

var petrol\_station\_4: int = 0

var petrol\_price\_station\_1: int = 0

var petrol\_price\_station\_2: int = 0

var petrol\_price\_station\_3: int = 0

var petrol\_price\_station\_4: int = 0

func \_ready():

update\_petrol()

func update\_petrol():

randomize()

petrol\_station\_1 = randi\_range(100, 1000)

petrol\_station\_2 = randf\_range(100, 1000)

petrol\_station\_3 = randi\_range(100, 1000)

petrol\_station\_4 = randi\_range(100, 1000)

petrol\_price\_station\_1 = randi\_range(1, 6)

petrol\_price\_station\_2 = randi\_range(1, 6)

petrol\_price\_station\_3 = randi\_range(1, 6)

petrol\_price\_station\_4 = randi\_range(1, 6)

update\_petrol\_info()

func update\_petrol\_info():

info\_station\_1.text = '''Станция №1

Запас: ''' + str(petrol\_station\_1) + '''л.

Цена: ''' + str(petrol\_price\_station\_1) + 'р за 1л'

info\_station\_2.text = '''Станция №2

Запас: ''' + str(petrol\_station\_2) + '''л.

Цена: ''' + str(petrol\_price\_station\_2) + 'р за 1л'

info\_station\_3.text = '''Станция №3

Запас: ''' + str(petrol\_station\_3) + '''л.

Цена: ''' + str(petrol\_price\_station\_3) + 'р за 1л'

info\_station\_4.text = '''Станция №4

Запас: ''' + str(petrol\_station\_4) + '''л.

Цена: ''' + str(petrol\_price\_station\_4) + 'р за 1л'

Terminal.gd

extends StaticBody3D

@onready var gas\_station = $".."

@onready var place\_to\_spawn = $place\_to\_spawn

# HOW MUCH CAN YOU BUY FOR A ONE CANISTER

var buy\_amount: int = 20

func buy\_petrol():

var isSuccess: bool = false

var amountBought: int = 0

var petrol\_left: int = gas\_station.petrol\_station\_1

if petrol\_left >= buy\_amount:

var petrol\_price: int = buy\_amount \* gas\_station.petrol\_price\_station\_1

if SIN\_WORLD\_DATA.WORLD\_DATA['money'] >= petrol\_price:

SIN\_WORLD\_DATA.WORLD\_DATA['money'] -= petrol\_price

SIN\_WORLD\_SIGNALS.emit\_signal('PLAYER\_UI\_CASH\_UPDATE', -petrol\_price)

isSuccess = true

amountBought = buy\_amount

gas\_station.petrol\_station\_1 -= buy\_amount

else:

if petrol\_left > 0:

var petrol\_price: int = gas\_station.petrol\_station\_1 \* gas\_station.petrol\_price\_station\_1

if SIN\_WORLD\_DATA.WORLD\_DATA['money'] >= petrol\_price:

isSuccess = true

amountBought = gas\_station.petrol\_station\_1

SIN\_WORLD\_DATA.WORLD\_DATA['money'] -= petrol\_price

SIN\_WORLD\_SIGNALS.emit\_signal('PLAYER\_UI\_CASH\_UPDATE', -petrol\_price)

gas\_station.petrol\_station\_1 = 0

if isSuccess:

var petrol\_canister = gas\_station.petrol\_canister.instantiate()

get\_tree().get\_root().add\_child(petrol\_canister)

petrol\_canister.item.litres = amountBought

petrol\_canister.item.weight = amountBought

petrol\_canister.global\_position = place\_to\_spawn.global\_position

petrol\_canister.global\_rotation = place\_to\_spawn.global\_rotation

gas\_station.update\_petrol\_info()

**Hospital.gd**

extends Node3D

@export var street\_name: String = ''

func \_ready():

$street\_name.text = street\_name

**Shop.gd**

extends StaticBody3D

@onready var animation\_player = $AnimationPlayer

func \_on\_area\_door\_detect\_body\_entered(body):

if body.is\_in\_group('player'):

if not animation\_player.is\_playing():

animation\_player.play("doors\_open")

else:

await animation\_player.animation\_finished

animation\_player.play("doors\_open")

func \_on\_area\_door\_detect\_body\_exited(body):

if body.is\_in\_group('player'):

if not animation\_player.is\_playing():

animation\_player.play("doors\_closed")

else:

await animation\_player.animation\_finished

animation\_player.play("doors\_closed")

**StreetLamp.gd**

extends Node3D

@onready var spot\_light\_3d = $SpotLight3D

@onready var anim\_light = $anim\_light

func \_ready():

SIN\_WORLD\_SIGNALS.LIGHTS\_ON.connect(\_lights\_on)

SIN\_WORLD\_SIGNALS.LIGHTS\_OFF.connect(\_lights\_off)

func \_lights\_on():

anim\_light.play("lights\_on")

func \_lights\_off():

anim\_light.play("lights\_off")

**Thunderbolt.gd**

extends Node3D

@onready var anim\_thunderbolt = $thunderbolt/anim\_thunderbolt

func \_ready():

$thunderbolt.global\_rotation.y = deg\_to\_rad(randi\_range(0,360))

anim\_thunderbolt.speed\_scale = randf\_range(0.3, 2.5)

anim\_thunderbolt.play("play")

func \_on\_thunder\_sound\_1\_finished():

queue\_free()

**ENV.gd**

extends Node3D

@export var thunderbolt: PackedScene

@onready var we = $WE

@onready var sun = $Sun

@onready var weather\_rainy = $WEATHER\_RAINY

@onready var weather\_snowy = $WEATHER\_SNOWY

@onready var weather\_windy = $WEATHER\_WINDY

# OBJECTS

@onready var ufo = $"../../MOBS/ufo"

var time\_hours: int = 0

var time\_minutes: int = 0

var player: Object = null

var weather\_id: String = 'clear'

var wind\_direction: int = 0

var wind\_intensivity: int = 5000

func \_ready():

# === WEATHER

# === SETTINGS

load\_graphic\_settings()

launch\_tod()

func load\_graphic\_settings():

# SET UP SHADOWS

match SIN\_SETTINGS.SETTINGS['GRAPHICS']['shadows']:

'no':

sun.shadow\_enabled = false

'bad':

sun.directional\_shadow\_mode = DirectionalLight3D.SHADOW\_ORTHOGONAL

sun.shadow\_enabled = true

'mid':

sun.directional\_shadow\_mode = DirectionalLight3D.SHADOW\_PARALLEL\_2\_SPLITS

sun.shadow\_enabled = true

'good':

sun.directional\_shadow\_mode = DirectionalLight3D.SHADOW\_PARALLEL\_4\_SPLITS

sun.shadow\_enabled = true

# SET UP BLOOM

match SIN\_SETTINGS.SETTINGS['GRAPHICS']['glow']:

'no':

we.environment.glow\_enabled = false

'yes':

we.environment.glow\_enabled = true

match SIN\_SETTINGS.SETTINGS['GRAPHICS']['sdfgi']:

'no':

we.environment.sdfgi\_enabled = false

'yes':

we.environment.sdfgi\_enabled = true

match SIN\_SETTINGS.SETTINGS['GRAPHICS']['ssao']:

'no':

we.environment.ssao\_enabled = false

'yes':

we.environment.ssao\_enabled = true

func launch\_tod():

time\_hours = int(SIN\_WORLD\_DATA.WORLD\_DATA['tod']/60)

time\_minutes = int(SIN\_WORLD\_DATA.WORLD\_DATA['tod']%60)

$timer\_daycycle.start()

$anim.play('tod')

$anim.seek(SIN\_WORLD\_DATA.WORLD\_DATA['tod'])

# ХОД ВРЕМЕНИ

func \_on\_timer\_daycycle\_timeout():

if SIN\_WORLD\_DATA.WORLD\_DATA['tod'] < 1439:

SIN\_WORLD\_DATA.WORLD\_DATA['tod'] += 1

else:

SIN\_WORLD\_DATA.value\_day\_passed()

SIN\_WORLD\_DATA.WORLD\_DATA['tod'] = 0

SIN\_WORLD\_SIGNALS.emit\_signal('TOD\_DAY\_ENDED')

time\_hours = int(SIN\_WORLD\_DATA.WORLD\_DATA['tod']/60)

time\_minutes = int(SIN\_WORLD\_DATA.WORLD\_DATA['tod']%60)

match\_tod()

# ВСЕ ДЕЙСТВИЯ С ВРЕМЕНЕМ

func match\_tod():

# УПРАВЛЕНИЕ ФОНАРЯМИ

match SIN\_WORLD\_DATA.WORLD\_DATA['tod']:

1300:

SIN\_WORLD\_SIGNALS.emit\_signal('LIGHTS\_ON')

120:

SIN\_WORLD\_SIGNALS.emit\_signal('LIGHTS\_OFF')

# НОВЫЙ РАБОЧИЙ ДЕНЬ НАЧАЛО В 9 УТРА

540:

SIN\_WORLD\_DATA.new\_quota()

SIN\_WORLD\_SIGNALS.emit\_signal('WORK\_DAY\_START')

# РАБОЧИЙ ДЕНЬ КОНЧИЛСЯ В 6 ВЕЧЕРА

1080:

SIN\_WORLD\_SIGNALS.emit\_signal('WORK\_DAY\_END')

# РАБОЧАЯ СМЕНА. ПРОВЕРКА. С 9-18.

func isItTimeOfWork():

return time\_hours >= 9 and time\_hours < 18

# =============================== WEATHER ======================================

func \_physics\_process(delta):

global\_position = SIN\_WORLD\_DATA.player\_position

func \_input(event):

if event.is\_action\_pressed("key\_e"):

var isPlayerInAnyCar: bool = false

var cars: Array = get\_tree().get\_nodes\_in\_group('car')

for i in range(cars.size()):

if cars[i]['item']['isPlayerInside']:

isPlayerInAnyCar = true

if not isPlayerInAnyCar:

var arr = get\_tree().get\_nodes\_in\_group('player')

player = arr[0]

# === WEATHERS

func \_set\_weather\_clear():

weather\_id = 'clear'

weather\_rainy.visible = false

weather\_rainy.emitting = false

weather\_snowy.visible = false

weather\_snowy.emitting = false

var tween = create\_tween()

tween.tween\_property(we, 'environment:volumetric\_fog\_density', 0, 15)

tween.play()

func \_set\_weather\_rainy():

weather\_id = 'rainy'

weather\_rainy.visible = true

weather\_rainy.emitting = true

weather\_snowy.visible = false

weather\_snowy.emitting = false

var tween = create\_tween()

var tween2 = create\_tween()

var tween3 = create\_tween()

tween.tween\_property(we, 'environment:volumetric\_fog\_density', 0.1, 15)

tween2.tween\_property(we, 'environment:volumetric\_fog\_albedo', Color('7f7f7f'), 15)

tween3.tween\_property(we, 'environment:volumetric\_fog\_emission', Color('7f7f7f'), 15)

func \_set\_weather\_snowy():

weather\_id = 'snowy'

weather\_snowy.visible = true

weather\_snowy.emitting = true

weather\_rainy.visible = false

weather\_rainy.emitting = false

var tween = create\_tween()

var tween2 = create\_tween()

var tween3 = create\_tween()

tween.tween\_property(we, 'environment:volumetric\_fog\_density', 0.1, 15)

tween2.tween\_property(we, 'environment:volumetric\_fog\_albedo', Color('ffffff'), 15)

tween3.tween\_property(we, 'environment:volumetric\_fog\_emission', Color('ffffff'), 15)

func \_set\_weather\_foggy():

weather\_id = 'foggy'

weather\_snowy.visible = false

weather\_snowy.emitting = false

weather\_rainy.visible = false

weather\_rainy.emitting = false

var tween = create\_tween()

var tween2 = create\_tween()

var tween3 = create\_tween()

tween.tween\_property(we, 'environment:volumetric\_fog\_density', 0.3, 15)

tween2.tween\_property(we, 'environment:volumetric\_fog\_albedo', Color('7f7f7f'), 15)

tween3.tween\_property(we, 'environment:volumetric\_fog\_emission', Color('7f7f7f'), 15)

func \_set\_weather\_thunder():

weather\_id = 'thunder'

weather\_snowy.visible = false

weather\_snowy.emitting = false

weather\_rainy.visible = true

weather\_rainy.emitting = true

var tween = create\_tween()

var tween2 = create\_tween()

var tween3 = create\_tween()

tween.tween\_property(we, 'environment:volumetric\_fog\_density', 0.3, 15)

tween2.tween\_property(we, 'environment:volumetric\_fog\_albedo', Color('323232'), 15)

tween3.tween\_property(we, 'environment:volumetric\_fog\_emission', Color('323232'), 15)

func \_change\_wind():

wind\_direction = randi\_range(0, 359)

wind\_intensivity = randi\_range(10000, 10000)

var tween = create\_tween()

tween.tween\_property(weather\_windy, 'global\_rotation', Vector3(deg\_to\_rad(0), deg\_to\_rad(wind\_direction), deg\_to\_rad(0)), 10)

tween.play()

var tween\_2 = create\_tween()

tween\_2.tween\_property(weather\_windy, "amount", wind\_intensivity, 20)

tween\_2.play()

func \_on\_timer\_switch\_weather\_timeout():

$timer\_switch\_weather.wait\_time = randi\_range(240, 1440)

var weather\_id: int = randi\_range(0,3)

match weather\_id:

0:

\_set\_weather\_clear()

1:

\_set\_weather\_foggy()

2:

\_set\_weather\_snowy()

3:

\_set\_weather\_rainy()

$timer\_switch\_weather.start()

func \_on\_timer\_spawn\_thunderbolts\_timeout():

if weather\_id == 'thunder':

var spawn\_radius: int = 50

$timer\_spawn\_thunderbolts.wait\_time = randi\_range(10, 15)

var bolt = thunderbolt.instantiate()

get\_tree().get\_root().add\_child(bolt)

bolt.global\_position = Vector3(SIN\_WORLD\_DATA.player\_position.x + randi\_range(-spawn\_radius,spawn\_radius), global\_position.y, SIN\_WORLD\_DATA.player\_position.z + randi\_range(-spawn\_radius,spawn\_radius))

$timer\_spawn\_thunderbolts.start()

func \_on\_timer\_change\_wind\_timeout():

\_change\_wind()

$timer\_change\_wind.wait\_time = randi\_range(120, 540)

$timer\_change\_wind.start()

**Clouds.gdshader**

shader\_type sky;

render\_mode use\_half\_res\_pass;

group\_uniforms clouds;

uniform sampler2D cloud\_shape\_sampler : filter\_linear\_mipmap\_anisotropic, repeat\_enable;

uniform sampler2D cloud\_noise\_sampler : filter\_linear\_mipmap\_anisotropic, repeat\_enable;

uniform sampler2D cloud\_curves;

uniform int clouds\_samples : hint\_range(8, 32, 8) = 16;

uniform int shadow\_sample : hint\_range(1, 4, 1) = 4;

uniform float clouds\_density : hint\_range(0.0, 1.0, 0.1) = 0.5;

uniform float clouds\_scale : hint\_range(0.5, 1.5, 0.1) = 1.0;

uniform float clouds\_smoothness : hint\_range(0.01, 0.1, 0.01) = 0.035;

uniform vec3 clouds\_light\_color : source\_color;

uniform float clouds\_shadow\_intensity : hint\_range(0.1, 10.0, 0.1) = 1.0;

group\_uniforms high\_clouds;

uniform sampler2D high\_clouds\_sampler;

uniform float high\_clouds\_density : hint\_range(0.0, 1.0, 0.05) = 0.0;

group\_uniforms sky;

uniform vec3 top\_color : source\_color = vec3(1.0);

uniform vec3 bottom\_color : source\_color = vec3(1.0);

uniform vec3 sun\_scatter : source\_color = vec3(1.0);

group\_uniforms astro;

uniform vec3 astro\_tint : source\_color;

uniform sampler2D astro\_sampler : repeat\_disable, filter\_linear\_mipmap;

uniform float astro\_scale : hint\_range(0.1, 10.0, 0.1) = 1.0;

uniform float astro\_intensity : hint\_range(1.0, 3.0, 0.1) = 1.0;

group\_uniforms stars;

uniform float stars\_intensity : hint\_range(0.0, 5.0, 0.1) = 0.0;

group\_uniforms shooting\_stars;

uniform float shooting\_stars\_intensity : hint\_range(0.0, 10.0, 0.1) = 0.0;

uniform sampler2D shooting\_star\_sampler : filter\_linear, repeat\_disable;

uniform vec3 shooting\_star\_tint : source\_color;

float rand(float n){return fract(sin(n) \* 43758.5453123);}

// Voronoi method credit:

// The MIT License

// Copyright © 2013 Inigo Quilez

// https://www.shadertoy.com/view/ldl3Dl

vec3 hash( vec3 x ){

x = vec3( dot(x,vec3(127.1,311.7, 74.7)),

dot(x,vec3(269.5,183.3,246.1)),

dot(x,vec3(113.5,271.9,124.6)));

return fract(sin(x)\*43758.5453123);

}

vec3 voronoi( in vec3 x ){

vec3 p = floor( x );

vec3 f = fract( x );

float id = 0.0;

vec2 res = vec2( 100.0 );

for( int k=-1; k<=1; k++ )

for( int j=-1; j<=1; j++ )

for( int i=-1; i<=1; i++ ) {

vec3 b = vec3( float(i), float(j), float(k) );

vec3 r = vec3( b ) - f + hash( p + b );

float d = dot( r, r );

if( d < res.x ) {

id = dot( p+b, vec3(1.0,57.0,113.0 ) );

res = vec2( d, res.x );

} else if( d < res.y ) {

res.y = d;

}

}

return vec3( sqrt( res ), abs(id) );

}

// https://stackoverflow.com/questions/18558910/direction-vector-to-rotation-matrix

mat3 direction\_to\_matrix(vec3 direction) {

vec3 x\_axis = normalize(cross(vec3(0.0, 1.0, 0.0), direction));

vec3 y\_axis = normalize(cross(direction, x\_axis));

return mat3(vec3(x\_axis.x, y\_axis.x, direction.x),

vec3(x\_axis.y, y\_axis.y, direction.y),

vec3(x\_axis.z, y\_axis.z, direction.z));

}

float cloud\_density(vec3 p, float progress){

float t\_o = TIME \* 0.001;

float t\_o\_small = TIME \* -0.005;

float noise = texture(cloud\_noise\_sampler, p.xy \* 4.0 + t\_o\_small).x \* 0.1 + 0.9;

float clouds\_shape = texture(cloud\_shape\_sampler, (p.xy + t\_o) \* clouds\_scale).x;

float height\_curve = texture(cloud\_curves, vec2(progress, 0.0)).x;

float base\_density = 1.0 - clouds\_density;

float density =

smoothstep(base\_density - clouds\_smoothness,

base\_density + clouds\_smoothness,

clouds\_shape \* noise \* height\_curve

);

return density;

}

vec2 cloud\_ray\_march(vec3 direction, vec3 sun\_direction){

float density = 0.0;

float light = 0.0;

float height = 0.03;

vec3 sample\_point = vec3(0.0, 0.0, 2.0);

int loop\_offset = clouds\_samples \* 3;

for(int i = loop\_offset; i < clouds\_samples + loop\_offset; i++) {

float progress = float(i) / float(clouds\_samples);

sample\_point = direction \* height \* progress;

float point\_density = cloud\_density(sample\_point, progress);

density += point\_density;

float point\_light = 0.0;

for(int f = 0; f < shadow\_sample; f++){

float shadow\_progress = float(f) / float(shadow\_sample);

vec3 shadow\_offset = sun\_direction \* height \* shadow\_progress;

point\_light += cloud\_density(sample\_point + shadow\_offset, progress);

}

light += point\_light;

}

return vec2(density, light / float(shadow\_sample \* clouds\_samples));

}

vec3 random\_direction(float seed){

float phi = rand(seed) \* PI;

float costheta = rand(seed + 100.0) \* 2.0 - 1.0;

float theta = acos(costheta);

return vec3( sin(theta) \* cos(phi), (theta) \* sin(phi), cos(theta) );

}

float get\_shooting\_star(vec3 eyedir){

float shooting\_star = 0.0;

for(int i = 0; i < 4; i++){

float base\_rand = rand(float(i));

float time = TIME + base\_rand \* 2.0;

float duration = 0.5 + base\_rand;

float seed = floor(time / duration) \* duration + base\_rand;

float progress = mod(time, duration) / duration;

float rand\_value = rand(seed + 100.0);

float rand\_scale = base\_rand \* 10.0;

float a = rand\_value \* 0.8;

mat3 angle = mat3(vec3(cos(a), -sin(a), 0.0), vec3(sin(a), cos(a), 0.0), vec3(0.0, 0.0, 1.0));

vec3 shooting\_dir = direction\_to\_matrix(random\_direction(seed)) \* angle \* eyedir;

vec2 shooting\_uv = ((shooting\_dir.xy + vec2(0.0, progress \* 0.4)) \* (8.0 + rand\_scale)) + vec2(0.5);

float shooting\_mask = ceil(

clamp(shooting\_uv.x \* (1.0 - shooting\_uv.x), 0.0, 1.0) \*

clamp(shooting\_uv.y \* (1.0 - shooting\_uv.y), 0.0, 1.0)

) \* ceil(shooting\_dir.z);

shooting\_star = clamp(

shooting\_star + texture(shooting\_star\_sampler, shooting\_uv).x

\* sin(progress \* PI)

\* shooting\_mask \* rand\_value,

0.0, 1.0);

}

return clamp(shooting\_star, 0.0, 1.0);

}

void sky() {

float horizon\_mask = abs(EYEDIR.y);

float bottom\_mask = smoothstep(0.5, 0.45, SKY\_COORDS.y);

vec3 dir = direction\_to\_matrix(LIGHT0\_DIRECTION) \* EYEDIR;

vec2 astro\_uv = (-(dir.xy / dir.z) \* astro\_scale) + vec2(0.5);

float astro\_mask = ceil(

clamp(astro\_uv.x \* (1.0 - astro\_uv.x), 0.0, 1.0) \*

clamp(astro\_uv.y \* (1.0 - astro\_uv.y), 0.0, 1.0)

) \* ceil(dir.z);

vec4 astro\_color = texture(astro\_sampler, astro\_uv);

// Sky color

vec3 sky\_gradient = mix(bottom\_color.rgb, top\_color.rgb, clamp(EYEDIR.y, 0.0, 1.0));

vec3 sunset\_color = sun\_scatter \* (1.0 - horizon\_mask);

vec3 sky\_color = clamp(sky\_gradient + sunset\_color, 0.0, 1.0);

// Stars

if(stars\_intensity > 0.0){

vec2 stars = voronoi(EYEDIR \* 25.0).xz;

sky\_color += smoothstep(0.025 + ((1.0 + sin(TIME + stars.y)) / 2.0) \* 0.05, 0.0, stars.x) \* stars\_intensity;

}

// Add shooting stars

if(shooting\_stars\_intensity > 0.0){

sky\_color += get\_shooting\_star(EYEDIR) \* shooting\_stars\_intensity \* shooting\_star\_tint;

}

// Add astro

sky\_color = mix(sky\_color, astro\_color.rgb \* astro\_intensity \* astro\_tint, astro\_color.a \* astro\_mask \* bottom\_mask);

// Add high clouds

if(high\_clouds\_density > 0.0){

vec2 high\_clouds\_uv = (EYEDIR.xz / clamp(EYEDIR.y, 0.0, 1.0)) \* 0.25 + TIME \* 0.001;

float high\_clouds\_mask = texture(high\_clouds\_sampler, high\_clouds\_uv).x;

sky\_color = mix(sky\_color, clouds\_light\_color, smoothstep(0.0, 1.0, high\_clouds\_mask) \* horizon\_mask \* bottom\_mask \* high\_clouds\_density);

}

// clouds

if (AT\_HALF\_RES\_PASS) {

vec3 clouds\_direction = vec3(EYEDIR.xz / clamp(EYEDIR.y, 0.0, 1.0), 1.0);

vec2 clouds = EYEDIR.y > 0.0 ? cloud\_ray\_march(clouds\_direction, LIGHT0\_DIRECTION) : vec2(0.0);

COLOR = mix(bottom\_color, clouds\_light\_color, exp(-clouds.y \* clouds\_shadow\_intensity));

ALPHA = (1.0 - exp(-clouds.x \* horizon\_mask \* bottom\_mask \* 10.0));

}else{

COLOR.rgb = mix(sky\_color, HALF\_RES\_COLOR.rgb, HALF\_RES\_COLOR.a);

}

}

**Grass.gdshader**

shader\_type spatial;

render\_mode depth\_draw\_opaque, cull\_disabled;

// Texture settings

uniform sampler2D texture\_albedo : hint\_default\_white, repeat\_disable;

uniform sampler2D texture\_gradient : hint\_default\_white, repeat\_disable;

uniform sampler2D texture\_noise : hint\_default\_white;

uniform float alpha\_scissor\_threshold : hint\_range(0.0, 1.0);

uniform vec4 transmission : source\_color;

uniform vec4 secondary\_color : source\_color;

uniform float secondary\_attenuation = 0.2;

uniform float grass\_height = 1.0;

// Wind settings

uniform vec2 wind\_direction = vec2(1, -0.5);

uniform float wind\_speed = 1.0;

uniform float wind\_strength = 2.0;

uniform float noise\_scale = 20.0;

instance uniform float camera\_bend\_strength : hint\_range(0.0, 3.0) = 0.2;

varying float color;

varying float height;

void vertex() {

height = VERTEX.y;

float influence = smoothstep(0, 1, height / 2.0);

vec4 world\_pos = MODEL\_MATRIX \* vec4(VERTEX, 1.0);

vec2 uv = world\_pos.xz / (noise\_scale + 1e-2);

vec2 panning\_uv = uv + fract(TIME \* wind\_direction \* wind\_speed);

float wind = texture(texture\_noise, panning\_uv).r \* 2.0 - 0.4;

color = texture(texture\_noise, uv).r;

vec2 wind\_offset = -wind\_direction \* wind\_strength \* influence \* wind;

world\_pos.xz += wind\_offset;

world\_pos.y -= wind \* influence \* smoothstep(0.0, height, wind\_strength);

//Push the top vertex away from the camera to bend the grass clump

float ndotv = 1.0 - dot(vec3(0.0, 1.0, 0.0), normalize(INV\_VIEW\_MATRIX[1].xyz));

world\_pos.xz += INV\_VIEW\_MATRIX[1].xz \* camera\_bend\_strength \* height \* ndotv;

vec4 local\_pos = inverse(MODEL\_MATRIX) \* world\_pos;

local\_pos.x += wind\_strength \* influence \* cos(TIME \* 1.0) / 8.0;

local\_pos.z += wind\_strength \* influence \* sin(TIME \* 1.5) / 8.0;

VERTEX = local\_pos.xyz;

//NORMAL = vec3(0.0, 1.0, 0.0);

}

void fragment() {

vec4 tex = texture(texture\_albedo, UV);

if (tex.a < alpha\_scissor\_threshold) {

discard;

}

BACKLIGHT = transmission.rgb;

vec4 gradient = texture(texture\_gradient, vec2(height / grass\_height, 0.0));

float secondary\_weight = smoothstep(0.0, 1.0, color - secondary\_attenuation);

ALBEDO = tex.rbg \* gradient.rgb;

//ALBEDO = mix(ALBEDO, secondary\_color.rgb, secondary\_weight);

}

**LakeWater.gdshader**

shader\_type spatial;

render\_mode cull\_disabled;

uniform vec3 albedo : source\_color;

uniform vec3 albedo2 : source\_color;

uniform float metallic : hint\_range(0.0, 1.0) = 0;

uniform float roughness : hint\_range(0.0, 1.0) = 0.02;

uniform sampler2D wave;

uniform sampler2D texture\_normal;

uniform sampler2D texture\_normal2;

uniform vec2 wave\_direction = vec2(2.0,0.0); // Direction of wave 1

uniform vec2 wave\_direction2 = vec2(0.0,1.0); // Direction of wave 2

uniform float time\_scale : hint\_range(0.0, 0.2, 0.005) = 0.025; // Rate of movement multiplied by TIME

uniform float noise\_scale = 10.0;

uniform float height\_scale = 0.15;

uniform vec4 color\_deep : source\_color; // Deep depth color

uniform vec4 color\_shallow : source\_color; // Shallow depth color

uniform float beers\_law = 2.0; // Beer's law application

uniform float depth\_offset = -0.75; // Offset

uniform float edge\_scale = 0.1;

uniform float near = 1.0;

uniform float far = 100.0;

uniform vec3 edge\_color : source\_color;

uniform sampler2D DEPTH\_TEXTURE : hint\_depth\_texture, filter\_linear\_mipmap;

uniform sampler2D SCREEN\_TEXTURE : hint\_screen\_texture, filter\_linear\_mipmap;

// Varying variables

varying float height;

varying vec3 world\_pos;

float fresnel(float amount, vec3 normal, vec3 view)

{

return pow((1.0 - clamp(dot(normalize(normal), normalize(view)), 0.0, 1.0 )), amount);

}

float edge(float depth){

depth = 2.0 \* depth - 1.0;

return near \* far / (far + depth \* (near - far));

}

void vertex() {

world\_pos = (MODEL\_MATRIX \* vec4(VERTEX, 1.0)).xyz;

height = texture(wave, world\_pos.xz / noise\_scale + TIME \* time\_scale).r;

VERTEX.y += height \* height\_scale;

}

void fragment() {

// Depth variables and calc

float depth\_texture = texture(DEPTH\_TEXTURE, SCREEN\_UV).r \* 2.0 - 1.0;

float depth = PROJECTION\_MATRIX[3][2] / (depth\_texture + PROJECTION\_MATRIX[2][2]);

float depth\_blend = exp((depth+VERTEX.z + depth\_offset) \* -beers\_law);

depth\_blend = clamp(1.0 - depth\_blend, 0.0, 1.0);

float depth\_blend\_power = clamp(pow(depth\_blend, 2.5), 0.0, 1.0);

// Retrieving depth color and applying the deep and shallow colors

vec3 screen\_color = textureLod(SCREEN\_TEXTURE, SCREEN\_UV, depth\_blend\_power \* 2.5).rgb;

vec3 depth\_color = mix(color\_shallow.rgb, color\_deep.rgb, depth\_blend\_power);

vec3 color = mix(screen\_color \* depth\_color, depth\_color \* 0.25, depth\_blend\_power \* 0.5);

// Getting edge depth calc

float z\_depth = edge(texture(DEPTH\_TEXTURE, SCREEN\_UV).x);

float z\_pos = edge(FRAGCOORD.z);

float z\_dif = z\_depth - z\_pos;

// Time calculations for wave (normal map) movement

vec2 time = (TIME \* wave\_direction) \* time\_scale; // Movement rate of first wave

vec2 time2 = (TIME \* wave\_direction2) \* time\_scale; // Movement rate of second wave

// Blend normal maps into one

vec3 normal\_blend = mix(texture(texture\_normal,world\_pos.xz + time).rgb, texture(texture\_normal2,world\_pos.xz + time2).rgb, 0.5);

// Calculate Fresnel

float fresnel = fresnel(5.0, NORMAL, VIEW);

vec3 surface\_color = mix(albedo, albedo2, fresnel); // Interpolate albedo values by frensel

vec3 depth\_color\_adj = mix(edge\_color, color, step(edge\_scale, z\_dif));

ALBEDO = clamp(surface\_color + depth\_color\_adj,vec3(0.0),vec3(1.0));

METALLIC = metallic;

ROUGHNESS = roughness;

NORMAL\_MAP = normal\_blend;

}

**Leaves.gdshader**

shader\_type spatial;

render\_mode depth\_draw\_opaque, cull\_disabled;

// Texture settings

uniform sampler2D texture\_albedo : hint\_default\_white, repeat\_disable;

uniform sampler2D texture\_gradient : hint\_default\_white;

uniform sampler2D texture\_noise : hint\_default\_white;

uniform float alpha\_scissor\_threshold : hint\_range(0.0, 1.0);

uniform vec4 transmission : source\_color;

uniform float total\_height = 1.0;

// Wind settings

uniform vec2 wind\_direction = vec2(1, -0.5);

uniform float wind\_speed = 1.0;

uniform float wind\_strength = 2.0;

uniform float noise\_scale = 20.0;

varying float color;

varying float height;

void vertex() {

height = VERTEX.y;

vec4 world\_pos = MODEL\_MATRIX \* vec4(VERTEX, 1.0);

vec2 uv = (world\_pos.xz + VERTEX.yy) / (noise\_scale + 1e-2) ;

vec2 panning\_uv = uv + fract(TIME \* wind\_direction \* wind\_speed);

float wind = texture(texture\_noise, panning\_uv).r \* 2.0 - 0.4;

color = texture(texture\_noise, uv).r;

float wind\_influence = smoothstep(0, 1, 1.0 - UV.y);

vec2 wind\_offset = -wind\_direction \* wind\_strength \* wind\_influence \* wind;

world\_pos.xz += wind\_offset;

world\_pos.y -= wind \* wind\_influence \* wind\_strength \* 0.45;

vec4 local\_pos = inverse(MODEL\_MATRIX) \* world\_pos;

VERTEX = local\_pos.xyz;

//NORMAL = vec3(0.0, 1.0, 0.0);

}

void fragment() {

vec4 tex = texture(texture\_albedo, UV);

if (tex.a < alpha\_scissor\_threshold) {

discard;

}

BACKLIGHT = transmission.rgb;

vec4 gradient = texture(texture\_gradient, vec2(height / total\_height, 0.0));

ALBEDO = tex.rbg \* gradient.rgb;

}

**Player.gdshader**

shader\_type canvas\_item;

uniform sampler2D noise: repeat\_enable;

uniform vec4 line\_color: source\_color;

uniform float line\_count: hint\_range(0.0, 2.0, 0.05) = 2;

uniform float line\_density: hint\_range(0.0, 1.0) = 0.56;

uniform float line\_faloff: hint\_range(0.0, 1.0) = 0;

uniform float mask\_size: hint\_range(0.0, 1.0) = 0.333;

uniform float mask\_edge: hint\_range(0.0, 1.0) = 0.372;

uniform float animation\_speed: hint\_range(1.0, 20.0) = 20;

float inv\_lerp(float from, float to, float value){

return (value - from) / (to - from);

}

vec2 polar\_coordinates(vec2 uv, vec2 center, float zoom, float repeat)

{

vec2 dir = uv - center;

float radius = length(dir) \* 2.0;

float angle = atan(dir.y, dir.x) \* 1.0/(PI \* 2.0);

return mod(vec2(radius \* zoom, angle \* repeat), 1.0);

}

vec2 rotate\_uv(vec2 uv, vec2 pivot, float rotation) {

float cosa = cos(rotation);

float sina = sin(rotation);

uv -= pivot;

return vec2(

cosa \* uv.x - sina \* uv.y,

cosa \* uv.y + sina \* uv.x

) + pivot;

}

void fragment(){

vec2 polar\_uv = polar\_coordinates(rotate\_uv(UV, vec2(0.5), floor(fract(TIME) \* animation\_speed) ) , vec2(0.5), 0.01, line\_count);

vec3 lines = texture(noise, polar\_uv).rgb;

float mask\_value = length(UV - vec2(0.5));

float mask = inv\_lerp(mask\_size, mask\_edge, mask\_value);

float result = 1.0 - (mask \* line\_density);

result = smoothstep(result, result + line\_faloff, lines.r);

COLOR.rgb = vec3(line\_color.rgb);

COLOR.a = min(line\_color.a, result);

}

**Underwater.gdshader**

shader\_type spatial;

render\_mode blend\_mix, unshaded;

uniform sampler2D SCREEN\_TEXTURE : hint\_screen\_texture, filter\_linear\_mipmap;

uniform vec4 tint : source\_color; // tint effect

uniform float wave\_speed = 3.0; // wave loop speed

uniform float wave\_freq = 10.0; // wave vertical freq

uniform float wave\_width = 1; // wave width

uniform float blur = 2.0; // Defines the blur strength. Increase for a larger blur radius.

void fragment(){

vec2 wave\_uv\_offset;

wave\_uv\_offset.y = 0.0;

wave\_uv\_offset.x = cos((TIME\*wave\_speed)+UV.x+UV.y\*wave\_freq\*2.0)\*wave\_width\*0.01;

float blur\_strength = blur / 100.0;

vec3 blurred\_color = vec3(0.0);

int sample\_count = 0;

for (float x = -blur\_strength; x <= blur\_strength; x += blur\_strength / 10.0) {

for (float y = -blur\_strength; y <= blur\_strength; y += blur\_strength / 10.0) {

blurred\_color += texture(SCREEN\_TEXTURE, SCREEN\_UV + wave\_uv\_offset + vec2(x, y)).rgb;

sample\_count++;

}

}

blurred\_color /= float(sample\_count);

ALBEDO = blurred\_color \* tint.rgb;

ALPHA = tint.a;

}

**SIN\_SETTINGS.gd**

extends Node

var work\_path: String = 'user://settings.dat'

var SETTINGS : Dictionary = {

'AUDIO': {

'db\_sounds': '1',

'db\_music': '1',

},

'GRAPHICS': {

'antialiasing': 'no',

'glow': 'no', # USED IN ENV NODE

'vsync': 'no',

'fxaa': 'no', # USED IN ENV NODE

'taa': 'no',

'sdfgi': 'no', # USED IN ENV NODE

'ssao': 'no', # USED IN ENV NODE

'shadows': 'no', # USED IN ENV NODE dir\_light.light.directional\_shadow\_mode =

'frequency': 60,

'far': 60, # USED BY ALL CAMERAS

},

'ADDITIONAL': {

'fps\_counter': false,

}

}

func \_ready():

if settings\_exists():

settings\_load()

settings\_consume()

else:

settings\_save()

func settings\_exists():

return FileAccess.file\_exists(work\_path)

func settings\_save():

var filer = FileAccess.open(work\_path,FileAccess.WRITE)

filer.store\_var(SETTINGS)

filer.close()

func settings\_load():

var filer = FileAccess.open(work\_path,FileAccess.READ)

SETTINGS = filer.get\_var()

filer.close()

func settings\_consume():

match SETTINGS['GRAPHICS']['antialiasing']:

'no':

get\_viewport().msaa\_3d = Viewport.MSAA\_DISABLED

'x2':

get\_viewport().msaa\_3d = Viewport.MSAA\_2X

'x4':

get\_viewport().msaa\_3d = Viewport.MSAA\_4X

'x8':

get\_viewport().msaa\_3d = Viewport.MSAA\_8X

match SETTINGS['GRAPHICS']['vsync']:

'no':

DisplayServer.window\_set\_vsync\_mode(DisplayServer.VSYNC\_DISABLED)

'yes':

DisplayServer.window\_set\_vsync\_mode(DisplayServer.VSYNC\_ENABLED)

match SETTINGS['GRAPHICS']['taa']:

'no':

get\_viewport().use\_taa = false

'yes':

get\_viewport().use\_taa = true

# SET PHYSICAL FPS

ProjectSettings.set\_setting('physics/common/physics\_ticks\_per\_second', SETTINGS['GRAPHICS']['frequency'])

**SIN\_TASK\_SYSTEM.gd**

extends Node

var characters = 'abcdefghijklmnopqrstuvwxyz'

var ACTIVE\_TASKS: Array = []

func \_ready():

SIN\_WORLD\_SIGNALS.PACKAGE\_DELIVERED.connect(\_package\_delivered)

SIN\_WORLD\_SIGNALS.PACKAGE\_FAILED.connect(\_package\_failed)

# FUNCS

# Deliver ID должен выбираться из списка доступных Deliver ID.

# Deliver ID должен совпадать с Deliver ID для конкретного дома (Зоны приема посылки)

func add\_random\_task() -> void:

var package\_weight: int = 0

var package\_fill\_type: int = randi\_range(0,10)

var package\_fill: String = 'void'

if package\_fill\_type >= 0 and package\_fill\_type < 6:

package\_fill = 'based'

elif package\_fill\_type >= 6 and package\_fill\_type < 9:

package\_fill = 'food' # ECO

elif package\_fill\_type == 9:

package\_fill = 'docs' # AVOID WATER

elif package\_fill\_type == 10:

package\_fill = 'glass' # FRAGILE

var package\_type: int = randi\_range(0,1)

var package\_type\_id: String = 'void'

match package\_type:

0:

package\_type\_id = 'box'

package\_weight = randi\_range(2, 20)

1:

package\_type\_id = 'letter'

package\_weight = randi\_range(1, 3)

var address\_lib = ClassAddresses.new()

var deliver\_id = address\_lib.get\_random\_delivery\_id()

var target\_address = address\_lib.addresses[str(deliver\_id)]

var task\_data: Dictionary = {

'task\_id': generate\_word(10) + str(Time.get\_unix\_time\_from\_system()) + str(randi\_range(10000,99999)),

'id': package\_type\_id,

'name': 'Доставка на адрес ' + target\_address,

'deliver\_id': deliver\_id,

'target\_address': target\_address,

'fill\_type': package\_fill,

'weight': package\_weight

}

ACTIVE\_TASKS.append(task\_data)

SIN\_WORLD\_SIGNALS.emit\_signal('PACKAGE\_CREATED', task\_data)

func add\_fixed\_task(deliver\_id) -> void:

var address\_lib = ClassAddresses.new()

var target\_address = address\_lib.addresses[str(deliver\_id)]

var package\_fill\_type: int = randi\_range(0,10)

var package\_fill: String = 'void'

if package\_fill\_type >= 0 and package\_fill\_type < 6:

package\_fill = 'based'

elif package\_fill\_type >= 6 and package\_fill\_type < 9:

package\_fill = 'food' # ECO

elif package\_fill\_type == 9:

package\_fill = 'docs' # AVOID WATER

elif package\_fill\_type == 10:

package\_fill = 'glass' # FRAGILE

var package\_type: int = randi\_range(0,1)

var package\_type\_id: String = 'void'

match package\_type:

0:

package\_type\_id = 'box'

1:

package\_type\_id = 'letter'

var task\_data: Dictionary = {

'task\_id': generate\_word(10) + str(Time.get\_unix\_time\_from\_system()) + str(randi\_range(10000,99999)),

'id': package\_type\_id,

'name': 'Доставка на адрес ' + target\_address,

'deliver\_id': deliver\_id,

'target\_address': target\_address,

'fill\_type': package\_fill,

'weight': randi\_range(2, 20)

}

ACTIVE\_TASKS.append(task\_data)

SIN\_WORLD\_SIGNALS.emit\_signal('PACKAGE\_CREATED', task\_data)

func remove\_task(task\_id: String) -> bool:

for i in range(ACTIVE\_TASKS.size()):

if ACTIVE\_TASKS[i]['task\_id'] == task\_id:

ACTIVE\_TASKS.remove\_at(i)

return true

return false

# UTILS

func generate\_word(length):

var word: String

var n\_char = len(characters)

for i in range(length):

word += characters[randi()% n\_char]

return word

# SIGNAL HANDLE

func \_package\_delivered(task\_id: String):

randomize()

# ЗАКРЫТЬ ЗАДАНИЕ НА ДОСТАВКУ

SIN\_TASK\_SYSTEM.remove\_task(task\_id)

# ДОБАВИТЬ ДЕНЕГ ЗА ДОСТАВКУ

var value\_reward: int = randi\_range(10,50)

SIN\_WORLD\_DATA.value\_change\_money(value\_reward)

# ДОБАВИТЬ В СЧЕТЧИК КВОТЫ ОДНУ ДОСТАВЛЕННУЮ ПОСЫЛКУ

if not SIN\_WORLD\_DATA.isQuotaReached():

SIN\_WORLD\_DATA.add\_delivered\_to\_quota\_counter()

func \_package\_failed(task\_id: String):

randomize()

# ЗАКРЫТЬ ЗАДАНИЕ НА ДОСТАВКУ

SIN\_TASK\_SYSTEM.remove\_task(task\_id)

# ОТНЯТЬ ДЕНЕГ ЗА ХУЁВУЮ ДОСТАВКУ

var value\_to\_pay: int = randi\_range(10,50)

SIN\_WORLD\_DATA.value\_change\_money(-value\_to\_pay)

**SIN\_WORLD\_DATA.gd**

extends Node

var work\_path: String = 'user://save.dat'

var WORLD\_DATA: Dictionary = {

'money': 0,

'player\_inv': {

'slot\_1': {'id': 'void'},

'slot\_2': {'id': 'void'},

'slot\_3': {'id': 'void'},

'slot\_4': {'id': 'void'}

},

'tod': 540,

'world': [],

'day\_num': 1,

'daily\_quota': 1,

'daily\_quota\_delivered': 0,

'firing\_warns': 0,

'player\_insane': false,

}

var last\_death\_reason: String = ''

var player\_position: Vector3 = Vector3()

func \_ready():

if data\_exists():

data\_load()

else:

data\_reset()

# MAIN FUNCS

func data\_reset():

var WORLD\_DATA\_BASE: Dictionary = {

'money': 0, # MONEY OF PLAYER

'player\_inv': {

'slot\_1': {'id': 'void'},

'slot\_2': {'id': 'void'},

'slot\_3': {'id': 'void'},

'slot\_4': {'id': 'void'}

}, # INVENTORY

'tod': 540, # TIME OF DAY IN SECONDS

'world': [],

'day\_num': 1, # NUM OF DAYS PASSED

'daily\_quota': 1, # AMOUNT OF PACKAGES TO DELIVER

'daily\_quota\_delivered': 0,

'firing\_warns': 0,

# BUFFS & DEBUFFS

'player\_insane': false,

}

WORLD\_DATA = WORLD\_DATA\_BASE

var filer = FileAccess.open(work\_path, FileAccess.WRITE)

filer.store\_var(WORLD\_DATA)

filer.close()

data\_save()

func data\_load() -> bool:

if data\_exists():

var filer = FileAccess.open(work\_path, FileAccess.READ)

WORLD\_DATA = filer.get\_var()

filer.close()

return true

return false

func data\_save():

var filer = FileAccess.open(work\_path, FileAccess.WRITE)

filer.store\_var(WORLD\_DATA)

filer.close()

func data\_is\_empty() -> bool:

var WORLD\_DATA\_BASE: Dictionary = {

'money': 0, # MONEY OF PLAYER

'player\_inv': {

'slot\_1': {'id': 'void'},

'slot\_2': {'id': 'void'},

'slot\_3': {'id': 'void'},

'slot\_4': {'id': 'void'}

}, # INVENTORY

'tod': 540, # TIME OF DAY IN SECONDS

'world': [],

'day\_num': 1, # NUM OF DAYS PASSED

'daily\_quota': 1, # AMOUNT OF PACKAGES TO DELIVER

'daily\_quota\_delivered': 0,

'firing\_warns': 0,

# BUFFS & DEBUFFS

'player\_insane': false,

}

if data\_exists():

if data\_load():

if WORLD\_DATA == WORLD\_DATA\_BASE:

return true

else:

return false

else:

return true

return true

func data\_exists() -> bool:

return FileAccess.file\_exists(work\_path)

# VALUE CHANGERS

func value\_change\_money(money\_to\_add: int) -> void:

WORLD\_DATA['money'] += money\_to\_add

SIN\_WORLD\_SIGNALS.emit\_signal('PLAYER\_UI\_CASH\_UPDATE', money\_to\_add)

func value\_day\_passed():

WORLD\_DATA['day\_num'] += 1

func new\_quota():

randomize()

WORLD\_DATA['daily\_quota'] += randi\_range(1, 3)

WORLD\_DATA['daily\_quota\_delivered'] = 0

func add\_delivered\_to\_quota\_counter():

if SIN\_WORLD\_DATA.WORLD\_DATA['tod'] >= 540 and SIN\_WORLD\_DATA.WORLD\_DATA['tod'] <= 1080:

SIN\_WORLD\_DATA.WORLD\_DATA['daily\_quota\_delivered'] += 1

func isQuotaReached():

return WORLD\_DATA['daily\_quota'] == WORLD\_DATA['daily\_quota\_delivered']

**SIN\_WORLD\_SIGNALS.gd**

extends Node

signal PACKAGE\_DELIVERED(task\_id)

signal PACKAGE\_FAILED(task\_id)

signal PACKAGE\_CREATED(task\_data)

signal LIGHTS\_ON

signal LIGHTS\_OFF

signal WORK\_DAY\_START

signal WORK\_DAY\_END

signal TOD\_DAY\_ENDED

signal GHOST\_GIRL\_ANGRY

signal PLAYER\_UI\_CASH\_UPDATE(value\_change)

signal WORLD\_SAVE

signal WORLD\_SAVED

signal OFFICE\_LIGHTS\_OFF

signal OFFICE\_LIGHTS\_ON

signal NEW\_FIRING\_WARN

**Bird.gd**

extends CharacterBody3D

var agro\_chance: int = 100

var bird\_speed: float = 10

var state: String = 'idle'

var current\_target: Object = null

var current\_slot\_item = {'id': 'void'}

var home\_point: Vector3 = Vector3()

var home\_point\_rot: Vector3 = Vector3()

var trigger\_cooldown: int = 0

# idle - bird is seating somewhere

# going\_to\_steal - bird is going to steal package

# follow\_target - bird is following package

# stealed - bird stole package

# go\_home - bird is flying

# lock - valve variable

func \_ready():

home\_point = global\_position

home\_point\_rot = global\_rotation

$hand/letter.visible = false

$hand/box.visible = false

$kar\_interval.wait\_time = randf\_range(5, 20)

$kar\_interval.start()

func \_on\_trigger\_body\_entered(body):

if state == 'idle':

if current\_target == null:

if trigger\_cooldown == 0:

if body is RigidBody3D:

if body.has\_method('deliver'):

var willAgro: int = randi\_range(0, 100)

if willAgro < agro\_chance:

# BIRD TARGETS PACKAGE

state = 'follow\_target'

current\_target = body

look\_at(current\_target.global\_transform.origin, Vector3.UP)

func \_physics\_process(delta):

if state == 'follow\_target':

go\_steal()

func \_on\_grab\_zone\_body\_entered(body):

# ONLY PACKAGES

if 'item' in body and body.has\_method('deliver') and body is RigidBody3D:

if trigger\_cooldown == 0:

var item = body.item

# IF BIRD'S SLOT IS FREE

if current\_slot\_item == {'id': 'void'}:

if not item.id == null:

match item.id:

'box':

state = 'go\_home'

look\_at(home\_point, Vector3.UP)

current\_slot\_item = body.item

$hand/letter.visible = false

$hand/box.visible = true

body.queue\_free()

go\_home()

'letter':

state = 'go\_home'

current\_slot\_item = body.item

$hand/letter.visible = true

$hand/box.visible = false

body.queue\_free()

go\_home()

\_:

pass

func try\_drop\_item():

var drop\_success: bool = false

if not current\_slot\_item == {'id': 'void'}:

match current\_slot\_item.id:

'box':

var obj = load('res://objects/PROPS/package\_box/package\_box.tscn').instantiate()

get\_tree().get\_root().add\_child(obj)

trigger\_cooldown = 20

obj.global\_position = $hand.global\_position

obj.global\_rotation = $hand.global\_rotation

obj.item = current\_slot\_item

current\_slot\_item = {'id': 'void'}

drop\_success = true

'letter':

var obj = load('res://objects/PROPS/package\_letter/package\_letter.tscn').instantiate()

get\_tree().get\_root().add\_child(obj)

trigger\_cooldown = 20

obj.global\_position = $hand.global\_position

obj.global\_rotation = $hand.global\_rotation

obj.item = current\_slot\_item

current\_slot\_item = {'id': 'void'}

drop\_success = true

if drop\_success:

current\_target = null

$hand/letter.visible = false

$hand/box.visible = false

func go\_steal():

if not current\_target == null:

look\_at(Vector3(current\_target.global\_position.x, current\_target.global\_position.y, current\_target.global\_position.z), Vector3.UP)

global\_position = lerp(global\_position, current\_target.global\_position, 0.01)

func go\_home():

look\_at(Vector3(home\_point.x, global\_position.y, home\_point.z), Vector3.UP)

var tween = create\_tween()

tween.tween\_property(self, 'global\_position', home\_point, home\_point.distance\_to(global\_position)/bird\_speed)

tween.play()

await tween.finished

state = 'idle'

func \_on\_timer\_timeout():

if trigger\_cooldown > 0:

trigger\_cooldown -= 1

func rotate\_towards\_target():

var tween = create\_tween()

tween.tween\_property(self, 'global\_rotation', Vector3(), 1)

tween.play()

func kar():

var kar\_type = int(randi\_range(0,1))

match kar\_type:

0:

$karkalka.stream = load("res://assets/sounds/MOBS/bird/bird\_1.mp3")

$karkalka.pitch\_scale = randf\_range(0.8, 1.2)

$karkalka.play()

1:

$karkalka.stream = load("res://assets/sounds/MOBS/bird/bird\_2.mp3")

$karkalka.pitch\_scale = randf\_range(0.8, 1.2)

$karkalka.play()

2:

$karkalka.stream = load("res://assets/sounds/MOBS/bird/bird\_3.mp3")

$karkalka.pitch\_scale = randf\_range(0.8, 1.2)

$karkalka.play()

$kar\_interval.wait\_time = randf\_range(5, 20)

$kar\_interval.start()

func \_on\_kar\_interval\_timeout():

kar()

func hurt():

try\_drop\_item()

go\_home()

state = 'go\_home'

$hurt.play()

**Cow.gd**

extends StaticBody3D

func \_ready():

reset\_moo()

func reset\_moo():

$interval\_between\_moo.wait\_time = randi\_range(20, 50)

$interval\_between\_moo.start()

func \_on\_interval\_between\_moo\_timeout():

var moo\_id: int = randi\_range(0, 2)

match moo\_id:

0:

$cow\_moo\_1.pitch\_scale = randf\_range(0.8, 1.2)

$cow\_moo\_1.play()

1:

$cow\_moo\_2.pitch\_scale = randf\_range(0.8, 1.2)

$cow\_moo\_2.play()

2:

$cow\_moo\_3.pitch\_scale = randf\_range(0.8, 1.2)

$cow\_moo\_3.play()

reset\_moo()

**GhostGirl.gd**

extends CharacterBody3D

@onready var mesh = $mesh

var isTargettingPlayer: bool = false

var obj\_targetting: Object = null

var agro\_meter: int = 0

var isAgro: bool = false

func \_ready():

SIN\_WORLD\_SIGNALS.WORK\_DAY\_START.connect(\_reset\_girl)

func \_physics\_process(delta):

if isTargettingPlayer:

if not obj\_targetting == null:

look\_at(Vector3(obj\_targetting.global\_position.x, global\_position.y, obj\_targetting.global\_position.z), Vector3.UP)

if isAgro:

if not obj\_targetting == null:

var tween = create\_tween()

tween.tween\_property(self, 'global\_position', obj\_targetting.global\_position, global\_position.distance\_to(obj\_targetting.global\_position)/2)

tween.play()

func \_on\_appearance\_trigger\_body\_entered(body):

if SIN\_WORLD\_DATA.WORLD\_DATA['tod'] > 1300 or SIN\_WORLD\_DATA.WORLD\_DATA['tod'] < 120:

if body.is\_in\_group('player'):

isTargettingPlayer = true

mesh.visible = true

obj\_targetting = body

agro\_meter += 1

$AnimationPlayer.play('show')

if not isAgro:

play\_random\_screamer()

if agro\_meter == 2:

isAgro = true

SIN\_WORLD\_SIGNALS.emit\_signal('GHOST\_GIRL\_ANGRY')

func \_on\_appearance\_trigger\_body\_exited(body):

if body.is\_in\_group('player'):

if not isAgro:

# if it is player

isTargettingPlayer = false

mesh.visible = false

obj\_targetting = null

$AnimationPlayer.play('hide')

func play\_random\_screamer():

randomize()

var scream\_id: int = randi\_range(1,5)

match scream\_id:

1:

$screamer\_1.play()

2:

$screamer\_2.play()

3:

$screamer\_3.play()

4:

$screamer\_4.play()

5:

$screamer\_5.play()

6:

$screamer\_6.play()

func \_on\_death\_area\_body\_entered(body):

if SIN\_WORLD\_DATA.WORLD\_DATA['tod'] > 1300 or SIN\_WORLD\_DATA.WORLD\_DATA['tod'] < 120:

if body.is\_in\_group('player'):

body.queue\_free()

SIN\_WORLD\_DATA.last\_death\_reason = 'ghost'

get\_tree().change\_scene\_to\_file.bind("res://scenes/death/death.tscn").call\_deferred()

func \_reset\_girl():

isAgro = false

agro\_meter = 0

obj\_targetting = null

isTargettingPlayer = false

global\_position = Vector3(50,0,1177)

$AnimationPlayer.play('hide')

**Rooster.gd**

extends Node3D

@onready var between\_picks = $between\_picks

@onready var scream\_delay = $scream\_delay

func \_ready():

SIN\_WORLD\_SIGNALS.WORK\_DAY\_START.connect(scream)

reset\_pick\_timer()

func reset\_pick\_timer():

between\_picks.wait\_time = randi\_range(5, 10)

between\_picks.start()

func \_on\_between\_picks\_timeout():

$anim.play('pick')

func \_on\_anim\_animation\_finished(anim\_name):

reset\_pick\_timer()

func scream():

scream\_delay.wait\_time = randi\_range(5, 20)

scream\_delay.start()

func \_on\_scream\_delay\_timeout():

var scream\_id: int = randi\_range(0, 3)

match scream\_id:

0:

$scream\_1.pitch\_scale = randf\_range(0.9, 1.1)

$scream\_1.play()

1:

$scream\_2.pitch\_scale = randf\_range(0.9, 1.1)

$scream\_2.play()

2:

$scream\_3.pitch\_scale = randf\_range(0.9, 1.1)

$scream\_3.play()

3:

$scream\_4.pitch\_scale = randf\_range(0.9, 1.1)

$scream\_4.play()

**Ufo.gd**

extends CharacterBody3D

var object\_pool: Array = []

var rayIsOn: bool = false

@export var patrol\_region: Array = [Vector3(-500, 200, -500), Vector3(500, 500, 500)]

var target\_place: Vector3 = Vector3()

func \_ready():

$anim.play('rotate')

life\_cycle()

func \_physics\_process(delta):

if rayIsOn:

if object\_pool.size() > 0:

for i in range(object\_pool.size()):

if not object\_pool[i] == null:

# LIFT PACKAGE

object\_pool[i].global\_position.y = object\_pool[i].global\_position.y + 1

# LOCK PACKAGE IN BEAM

object\_pool[i].global\_position.z = lerp(object\_pool[i].global\_position.z, $mesh.global\_transform.origin.z, 0.2)

object\_pool[i].global\_position.x = lerp(object\_pool[i].global\_position.x, $mesh.global\_transform.origin.x, 0.2)

func \_on\_trigger\_body\_entered(body):

if rayIsOn:

if body is RigidBody3D:

object\_pool.append(body)

func \_on\_trigger\_body\_exited(body):

if rayIsOn:

if object\_pool.has(body):

object\_pool.erase(body)

func \_on\_destructor\_body\_entered(body):

if not body == self:

if body is RigidBody3D:

if body.has\_method('destroy\_task\_failed'):

body.destroy\_task\_failed()

func ray\_on():

rayIsOn = true

$ray/ray\_anim.play('ray\_on')

func ray\_off():

$ray/ray\_anim.play('ray\_off')

await $ray/ray\_anim.animation\_finished

rayIsOn = false

func life\_cycle():

# UFO VISIBILITY

var isVisible: bool = false

if SIN\_WORLD\_DATA.WORLD\_DATA['tod'] > 1300 or SIN\_WORLD\_DATA.WORLD\_DATA['tod'] < 120:

visible = true

elif SIN\_WORLD\_DATA.WORLD\_DATA['tod'] >= 120:

visible = false

if visible:

ray\_on()

$lifecycle\_part\_2.wait\_time = randf\_range(5.0, 15.0)

$lifecycle\_part\_2.start()

func \_on\_lifecycle\_part\_2\_timeout():

if visible:

await ray\_off()

target\_place = Vector3(randi\_range(patrol\_region[0].x, patrol\_region[1].x), randi\_range(patrol\_region[0].y, patrol\_region[1].y), randi\_range(patrol\_region[0].z, patrol\_region[1].z))

var tween = create\_tween()

tween.tween\_property(self, 'global\_position', target\_place, Vector3(global\_transform.origin).distance\_to(target\_place)/150)

tween.play()

await tween.finished

life\_cycle()

**UfoGiant.gd**

extends CharacterBody3D

func \_ready():

SIN\_WORLD\_SIGNALS.TOD\_DAY\_ENDED.connect(ufo\_show)

func ufo\_show():

if SIN\_WORLD\_DATA.WORLD\_DATA['day\_num'] % 10 == 0:

$anim\_appearance.play('show')

$anim.play('rotate')

await $anim\_appearance.animation\_finished

$giant\_ufo\_alert.play()

$giant\_ufo\_ambience.play()

await $giant\_ufo\_alert.finished

$anim\_appearance.play('hide')

await $anim\_appearance.animation\_finished

$giant\_ufo\_ambience.stop()

$giant\_ufo\_alert.stop()

$anim.stop()

**Player.gd**

extends CharacterBody3D

var isOnLadder: bool = false

# HOW MUCH KILOGRAMMS

var payload: int = 0

var payload\_multiplier: float = 10.0

# MUST BE HERE IN ORDER TO SAVE PLAYER

var item: Dictionary = {

'id': 'player'

}

# Money UI

@onready var money = $UI/UI/money

# FPS Counter

@onready var fps\_counter = $UI/UI/fps\_counter

# ESC MENU

var isEscMenuOpened: bool = false

@onready var esc\_menu = $UI/esc\_menu

@onready var anim\_esc\_menu = $UI/esc\_menu/anim\_esc\_menu

# INVENTORY

@onready var inventory\_loader = $INVENTORY\_LOADER

@export var throw\_item\_power: float = 10.0

var current\_slot\_selected: int = 1

@onready var slot\_1 = $UI/UI/inventory/slot\_1

@onready var slot\_2 = $UI/UI/inventory/slot\_2

@onready var slot\_3 = $UI/UI/inventory/slot\_3

@onready var slot\_4 = $UI/UI/inventory/slot\_4

@export\_category("Character")

@export var base\_speed : float = 3.0

@export var sprint\_speed : float = 6.0

@export var crouch\_speed : float = 1.0

@export var acceleration : float = 10.0

@export var jump\_velocity : float = 4.5

@export var mouse\_sensitivity : float = 0.1

@export var initial\_facing\_direction : Vector3 = Vector3.ZERO

@export\_group("Nodes")

@export var HEAD : Node3D

@export var CAMERA : Camera3D

@export var HEADBOB\_ANIMATION : AnimationPlayer

@export var JUMP\_ANIMATION : AnimationPlayer

@export var CROUCH\_ANIMATION : AnimationPlayer

@export var COLLISION\_MESH : CollisionShape3D

@export\_group("Controls")

# We are using UI controls because they are built into Godot Engine so they can be used right away

@export var JUMP : String = "move\_jump"

@export var LEFT : String = "move\_left"

@export var RIGHT : String = "move\_right"

@export var FORWARD : String = "move\_foward"

@export var BACKWARD : String = "move\_backward"

@export var PAUSE : String = "ui\_cancel"

@export var CROUCH : String

@export var SPRINT : String

# Uncomment if you want full controller support

#@export var LOOK\_LEFT : String

#@export var LOOK\_RIGHT : String

#@export var LOOK\_UP : String

#@export var LOOK\_DOWN : String

@export\_group("Feature Settings")

@export var immobile : bool = false

@export var jumping\_enabled : bool = true

@export var in\_air\_momentum : bool = true

@export var motion\_smoothing : bool = true

@export var sprint\_enabled : bool = true

@export var crouch\_enabled : bool = true

@export\_enum("Hold to Crouch", "Toggle Crouch") var crouch\_mode : int = 0

@export\_enum("Hold to Sprint", "Toggle Sprint") var sprint\_mode : int = 0

@export var dynamic\_fov : bool = true

@export var continuous\_jumping : bool = true

@export var view\_bobbing : bool = true

@export var jump\_animation : bool = true

# Member variables

var speed : float = base\_speed

var current\_speed : float = 0.0

var stamina: float = 100.0

# States: normal, crouching, sprinting

var state : String = "normal"

var low\_ceiling : bool = false # This is for when the cieling is too low and the player needs to crouch.

var was\_on\_floor : bool = true

# Get the gravity from the project settings to be synced with RigidBody nodes

var gravity : float = ProjectSettings.get\_setting("physics/3d/default\_gravity") # Don't set this as a const, see the gravity section in \_physics\_process

func \_ready():

# CONNECT ALL SIGNALS NEEDED

SIN\_WORLD\_SIGNALS.GHOST\_GIRL\_ANGRY.connect(\_go\_insane)

SIN\_WORLD\_SIGNALS.PLAYER\_UI\_CASH\_UPDATE.connect(\_update\_money\_ui)

# SET CAMERA FAR FROM SETTINGS

$Head/Camera.far = SIN\_SETTINGS.SETTINGS['GRAPHICS']['far']

# SET VISIBLE FPS COUNTER

fps\_counter.visible = SIN\_SETTINGS.SETTINGS['ADDITIONAL']['fps\_counter']

# EVERYTHING ELSE

set\_money\_value\_ui()

set\_slot\_selected(1)

calculate\_speed()

inventory\_loader.load\_hand\_visual(1)

inventory\_loader.load\_inventory\_visual()

Input.mouse\_mode = Input.MOUSE\_MODE\_CAPTURED

# Set the camera rotation to whatever initial\_facing\_direction is

if initial\_facing\_direction:

HEAD.set\_rotation\_degrees(initial\_facing\_direction) # I don't want to be calling this function if the vector is zero

# Reset the camera position

HEADBOB\_ANIMATION.play("RESET")

JUMP\_ANIMATION.play("RESET")

func \_physics\_process(delta):

SIN\_WORLD\_DATA.player\_position = global\_position

#region СТАМИНА

if stamina < 100 and state == 'normal':

stamina += 0.1

if Input.is\_action\_pressed(SPRINT):

var a = stamina - 0.1

if a > 0:

stamina -= 0.2

#endregion

current\_speed = Vector3.ZERO.distance\_to(get\_real\_velocity())

var cv : Vector3 = get\_real\_velocity()

var vd : Array[float] = [

snappedf(cv.x, 0.001),

snappedf(cv.y, 0.001),

snappedf(cv.z, 0.001)

]

# Gravity

if not is\_on\_floor():

velocity.y -= gravity \* delta

handle\_jumping()

var input\_dir = Vector2.ZERO

if !immobile:

input\_dir = Input.get\_vector(LEFT, RIGHT, FORWARD, BACKWARD)

handle\_movement(delta, input\_dir)

low\_ceiling = $CrouchCeilingDetection.is\_colliding()

handle\_state(input\_dir)

if dynamic\_fov:

update\_camera\_fov()

if view\_bobbing:

headbob\_animation(input\_dir)

if jump\_animation:

if !was\_on\_floor and is\_on\_floor(): # Just landed

JUMP\_ANIMATION.play("land")

was\_on\_floor = is\_on\_floor() # This must always be at the end of physics\_process

func handle\_jumping():

if jumping\_enabled:

if continuous\_jumping:

if isOnLadder:

if Input.is\_action\_pressed(JUMP):

velocity.y = 2

if Input.is\_action\_pressed(JUMP) and is\_on\_floor() and !low\_ceiling:

if jump\_animation:

JUMP\_ANIMATION.play("jump")

velocity.y += jump\_velocity

else:

if Input.is\_action\_just\_pressed(JUMP) and is\_on\_floor() and !low\_ceiling:

if jump\_animation:

JUMP\_ANIMATION.play("jump")

velocity.y += jump\_velocity

func handle\_movement(delta, input\_dir):

var direction = input\_dir.rotated(-HEAD.rotation.y)

direction = Vector3(direction.x, 0, direction.y)

move\_and\_slide()

if in\_air\_momentum:

if is\_on\_floor():

if motion\_smoothing:

velocity.x = lerp(velocity.x, direction.x \* speed, acceleration \* delta)

velocity.z = lerp(velocity.z, direction.z \* speed, acceleration \* delta)

else:

velocity.x = direction.x \* speed

velocity.z = direction.z \* speed

else:

if motion\_smoothing:

velocity.x = lerp(velocity.x, direction.x \* speed, acceleration \* delta)

velocity.z = lerp(velocity.z, direction.z \* speed, acceleration \* delta)

else:

velocity.x = direction.x \* speed

velocity.z = direction.z \* speed

func handle\_state(moving):

if sprint\_enabled:

if sprint\_mode == 0:

if Input.is\_action\_pressed(SPRINT) and state != "crouching" and int(stamina) > 0:

if moving:

if state != "sprinting":

enter\_sprint\_state()

else:

if state == "sprinting":

enter\_normal\_state()

elif state == "sprinting":

enter\_normal\_state()

elif sprint\_mode == 1:

if moving:

# If the player is holding sprint before moving, handle that cenerio

if Input.is\_action\_pressed(SPRINT) and state == "normal":

enter\_sprint\_state()

if Input.is\_action\_just\_pressed(SPRINT):

match state:

"normal":

enter\_sprint\_state()

"sprinting":

enter\_normal\_state()

elif state == "sprinting":

enter\_normal\_state()

if crouch\_enabled:

if crouch\_mode == 0:

if Input.is\_action\_pressed(CROUCH) and state != "sprinting":

if state != "crouching":

enter\_crouch\_state()

elif state == "crouching" and !$CrouchCeilingDetection.is\_colliding():

enter\_normal\_state()

elif crouch\_mode == 1:

if Input.is\_action\_just\_pressed(CROUCH):

match state:

"normal":

enter\_crouch\_state()

"crouching":

if !$CrouchCeilingDetection.is\_colliding():

enter\_normal\_state()

func enter\_normal\_state():

var prev\_state = state

if prev\_state == "crouching":

CROUCH\_ANIMATION.play\_backwards("crouch")

state = "normal"

if base\_speed - (payload/payload\_multiplier) > 0:

speed = base\_speed - (payload/payload\_multiplier)

else:

speed = 0

func enter\_crouch\_state():

var prev\_state = state

state = "crouching"

CROUCH\_ANIMATION.play("crouch")

if crouch\_speed - (payload/payload\_multiplier) > 0:

speed = crouch\_speed - (payload/payload\_multiplier)

else:

speed = 0

func enter\_sprint\_state():

var prev\_state = state

if prev\_state == "crouching":

CROUCH\_ANIMATION.play\_backwards("crouch")

state = "sprinting"

if sprint\_speed - (payload/payload\_multiplier) > 0:

speed = sprint\_speed - (payload/payload\_multiplier)

else:

speed = 0

func update\_camera\_fov():

if state == "sprinting":

CAMERA.fov = lerp(CAMERA.fov, 85.0, 0.3)

else:

CAMERA.fov = lerp(CAMERA.fov, 75.0, 0.3)

func headbob\_animation(moving):

if moving and is\_on\_floor():

var was\_playing : bool = false

if HEADBOB\_ANIMATION.current\_animation == "headbob":

was\_playing = true

HEADBOB\_ANIMATION.play("headbob", 0.25)

HEADBOB\_ANIMATION.speed\_scale = (current\_speed / base\_speed) \* 1.75

if !was\_playing:

HEADBOB\_ANIMATION.seek(float(randi() % 2)) # Randomize the initial headbob direction

# Let me explain that piece of code because it looks like it does the opposite of what it actually does.

# The headbob animation has two starting positions. One is at 0 and the other is at 1.

# randi() % 2 returns either 0 or 1, and so the animation randomly starts at one of the starting positions.

else:

HEADBOB\_ANIMATION.play("RESET", 0.25)

HEADBOB\_ANIMATION.speed\_scale = 1

func \_process(delta):

\_update\_time\_ui()

fps\_counter.text = 'FPS: ' + str(Performance.get\_monitor(Performance.TIME\_FPS))

var status : String = state

if !is\_on\_floor():

status += " in the air"

if Input.is\_action\_just\_pressed(PAUSE):

if Input.mouse\_mode == Input.MOUSE\_MODE\_CAPTURED:

Input.mouse\_mode = Input.MOUSE\_MODE\_VISIBLE

elif Input.mouse\_mode == Input.MOUSE\_MODE\_VISIBLE:

Input.mouse\_mode = Input.MOUSE\_MODE\_CAPTURED

HEAD.rotation.x = clamp(HEAD.rotation.x, deg\_to\_rad(-90), deg\_to\_rad(90))

# Uncomment if you want full controller support

#var controller\_view\_rotation = Input.get\_vector(LOOK\_LEFT, LOOK\_RIGHT, LOOK\_UP, LOOK\_DOWN)

#HEAD.rotation\_degrees.y -= controller\_view\_rotation.x \* 1.5

#HEAD.rotation\_degrees.x -= controller\_view\_rotation.y \* 1.5

func add\_item\_to\_inv(item\_data: Dictionary) -> bool:

var hasFreeSlot: bool = false

var target\_slot: int = -1

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_4']['id'] == 'void':

hasFreeSlot = true

target\_slot = 4

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_3']['id'] == 'void':

hasFreeSlot = true

target\_slot = 3

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_2']['id'] == 'void':

hasFreeSlot = true

target\_slot = 2

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_1']['id'] == 'void':

hasFreeSlot = true

target\_slot = 1

if hasFreeSlot:

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(target\_slot)] = item\_data

inventory\_loader.load\_inventory\_visual()

inventory\_loader.load\_hand\_visual(target\_slot)

set\_slot\_selected(target\_slot)

calculate\_speed()

return true

return false

func drop\_item\_slot(slot\_id: int):

var item\_full\_data = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

match item\_full\_data.id:

'void':

pass

'box':

var obj = load('res://objects/PROPS/package\_box/package\_box.tscn').instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.set\_labels()

obj.global\_position = $Head/Camera/item\_display/box.global\_position

obj.global\_rotation = $Head/Camera/item\_display/box.global\_rotation

obj.apply\_central\_impulse($Head/Camera/item\_display/box.global\_transform.basis.z \* -throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'letter':

var obj = load('res://objects/PROPS/package\_letter/package\_letter.tscn').instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/letter.global\_position

obj.global\_rotation = $Head/Camera/item\_display/letter.global\_rotation

obj.apply\_central\_impulse($Head/Camera/item\_display/letter.global\_transform.basis.z \* -throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'ball':

var obj = load('res://objects/PROPS/ball/ball.tscn').instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/ball.global\_position

obj.global\_rotation = $Head/Camera/item\_display/ball.global\_rotation

obj.apply\_central\_impulse($Head/Camera/item\_display/ball.global\_transform.basis.z \* -100)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'fuel\_tank':

var obj = load('res://objects/PROPS/fuel\_tank/fuel\_tank.tscn').instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/fuel\_tank.global\_position

obj.global\_rotation = $Head/Camera/item\_display/fuel\_tank.global\_rotation

obj.apply\_central\_impulse($Head/Camera/item\_display/fuel\_tank.global\_transform.basis.x \* throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'shotgun':

var obj = load("res://objects/PROPS/shotgun/shotgun.tscn").instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/fuel\_tank.global\_position

obj.global\_rotation = $Head/Camera/item\_display/fuel\_tank.global\_rotation

obj.global\_rotation.y += deg\_to\_rad(90)

obj.apply\_central\_impulse($Head/Camera/item\_display/fuel\_tank.global\_transform.basis.x \* throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'shell':

var obj = load("res://objects/PROPS/shell/shell.tscn").instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/fuel\_tank.global\_position

obj.global\_rotation = $Head/Camera/item\_display/fuel\_tank.global\_rotation

obj.global\_rotation.y += deg\_to\_rad(180)

obj.apply\_central\_impulse($Head/Camera/item\_display/fuel\_tank.global\_transform.basis.x \* throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'milk':

var obj = load("res://objects/PROPS/FOOD/milk/milk.tscn").instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/fuel\_tank.global\_position

obj.global\_rotation = $Head/Camera/item\_display/fuel\_tank.global\_rotation

obj.global\_rotation.y += deg\_to\_rad(180)

obj.apply\_central\_impulse($Head/Camera/item\_display/fuel\_tank.global\_transform.basis.x \* throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'spray\_orange':

var obj = load("res://objects/PROPS/spray\_orange/spray\_orange.tscn").instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/spray\_orange.global\_position

obj.global\_rotation = $Head/Camera/item\_display/spray\_orange.global\_rotation

obj.apply\_central\_impulse($Head/Camera/item\_display/spray\_orange.global\_transform.basis.x \* throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

'coffee\_can':

var obj = load("res://objects/PROPS/FOOD/coffee\_can/coffee\_can.tscn").instantiate()

get\_tree().get\_root().add\_child(obj)

obj.item = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)]

obj.global\_position = $Head/Camera/item\_display/coffee\_can.global\_position

obj.global\_rotation = $Head/Camera/item\_display/coffee\_can.global\_rotation

obj.apply\_central\_impulse($Head/Camera/item\_display/coffee\_can.global\_transform.basis.z \* -throw\_item\_power)

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(slot\_id)] = {'id': 'void'}

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

calculate\_speed()

inventory\_loader.load\_inventory\_visual()

func set\_slot\_selected(slot\_id: int):

match slot\_id:

1:

current\_slot\_selected = 1

slot\_1.modulate.a = 1

slot\_2.modulate.a = 0.4

slot\_3.modulate.a = 0.4

slot\_4.modulate.a = 0.4

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

2:

current\_slot\_selected = 2

slot\_1.modulate.a = 0.4

slot\_2.modulate.a = 1

slot\_3.modulate.a = 0.4

slot\_4.modulate.a = 0.4

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

3:

current\_slot\_selected = 3

slot\_1.modulate.a = 0.4

slot\_2.modulate.a = 0.4

slot\_3.modulate.a = 1

slot\_4.modulate.a = 0.4

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

4:

current\_slot\_selected = 4

slot\_1.modulate.a = 0.4

slot\_2.modulate.a = 0.4

slot\_3.modulate.a = 0.4

slot\_4.modulate.a = 1

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

func \_unhandled\_input(event):

if event is InputEventMouseMotion and Input.mouse\_mode == Input.MOUSE\_MODE\_CAPTURED:

HEAD.rotation\_degrees.y -= event.relative.x \* mouse\_sensitivity

HEAD.rotation\_degrees.x -= event.relative.y \* mouse\_sensitivity

# INVENTORY MOUSE WHEEL

if event.is\_action\_pressed('wheel\_up'):

if current\_slot\_selected < 4:

current\_slot\_selected += 1

else:

current\_slot\_selected = 1

if event.is\_action\_pressed('wheel\_down'):

if current\_slot\_selected > 1:

current\_slot\_selected -= 1

else:

current\_slot\_selected = 4

set\_slot\_selected(current\_slot\_selected)

# INVENTORY BY KEYS

if event.is\_action\_pressed("key\_1"):

set\_slot\_selected(1)

if event.is\_action\_pressed("key\_2"):

set\_slot\_selected(2)

if event.is\_action\_pressed("key\_3"):

set\_slot\_selected(3)

if event.is\_action\_pressed("key\_4"):

set\_slot\_selected(4)

# DROP ITEMS

if event.is\_action\_pressed('key\_g'):

drop\_item\_slot(current\_slot\_selected)

# PICKUP/INTERACT WITH ITEMS/PROPS

if event.is\_action\_pressed("key\_e"):

if not SIN\_WORLD\_DATA.WORLD\_DATA['player\_insane']:

if $Head/Camera/raycast\_hand.is\_colliding():

if not $Head/Camera/raycast\_hand.get\_collider() == null:

# BUY PETROL FROM GAS STATION

if $Head/Camera/raycast\_hand.get\_collider().has\_method('buy\_petrol'):

$Head/Camera/raycast\_hand.get\_collider().buy\_petrol()

# GRAB PACKAGES

if $Head/Camera/raycast\_hand.get\_collider().has\_method('grab'):

if add\_item\_to\_inv($Head/Camera/raycast\_hand.get\_collider().grab()):

$Head/Camera/raycast\_hand.get\_collider().queue\_free()

# PUT PACKAGE IN A BOX

if $Head/Camera/raycast\_hand.get\_collider().has\_method('try\_put\_package'):

if $Head/Camera/raycast\_hand.get\_collider().try\_put\_package(SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(current\_slot\_selected)]):

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(current\_slot\_selected)] = {'id': 'void'}

inventory\_loader.load\_inventory\_visual()

inventory\_loader.load\_hand\_visual(current\_slot\_selected)

# DRIVE A CAR

if $Head/Camera/raycast\_hand.get\_collider().has\_method('enter'):

$Head/Camera/raycast\_hand.get\_collider().enter(self)

# INTERRACT WITH ITEM

if $Head/Camera/raycast\_hand.get\_collider().has\_method('interract'):

$Head/Camera/raycast\_hand.get\_collider().interract()

func \_go\_insane():

SIN\_WORLD\_DATA.WORLD\_DATA['player\_insane'] = true

func \_update\_money\_ui(value\_changed):

var notif\_money\_change = load("res://objects/UI/money\_change\_notification/money\_change\_notification.tscn").instantiate()

$UI/UI/money\_change\_notifications.add\_child(notif\_money\_change)

notif\_money\_change.set\_content(value\_changed)

set\_money\_value\_ui()

func set\_money\_value\_ui():

if SIN\_WORLD\_DATA.WORLD\_DATA['money'] >= 0:

$UI/UI/money.add\_theme\_color\_override('font\_color', Color('00ff00'))

$UI/UI/money.text = str(SIN\_WORLD\_DATA.WORLD\_DATA['money']) + '₽'

else:

$UI/UI/money.add\_theme\_color\_override('font\_color', Color('ff0000'))

$UI/UI/money.text = '0(' + str(SIN\_WORLD\_DATA.WORLD\_DATA['money']) + ')₽'

func \_update\_time\_ui():

var time\_hours = int(SIN\_WORLD\_DATA.WORLD\_DATA['tod']/60)

var time\_minutes = int(SIN\_WORLD\_DATA.WORLD\_DATA['tod']%60)

var time\_h\_str: String = ''

var time\_m\_str: String = ''

if time\_hours < 10:

time\_h\_str = '0' + str(time\_hours)

else:

time\_h\_str = str(time\_hours)

if time\_minutes < 10:

time\_m\_str = '0' + str(time\_minutes)

else:

time\_m\_str = str(time\_minutes)

$UI/UI/time.text = str(time\_h\_str) + ':' + str(time\_m\_str)

func calculate\_speed():

payload = 0

for i in range(SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv'].size()):

if not SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(i+1)] == {}:

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(i+1)].has('weight'):

payload += SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_' + str(i+1)]['weight']

$UI/UI/weight.text = str(payload) + 'кг'

set\_speed\_by\_payload()

func set\_speed\_by\_payload():

match state:

'crouching':

enter\_crouch\_state()

'normal':

enter\_normal\_state()

'sprinting':

enter\_sprint\_state()

func \_on\_check\_die\_from\_height\_body\_entered(body):

if -velocity.y > 20:

SIN\_WORLD\_DATA.last\_death\_reason = 'fall'

queue\_free()

get\_tree().change\_scene\_to\_file.bind("res://scenes/death/death.tscn").call\_deferred()

**ITEM\_USAGE.gd**

extends Node

@onready var inventory\_loader = $"../INVENTORY\_LOADER"

@onready var player = $".."

# RAYCAST

@onready var raycast\_hand = $"../Head/Camera/raycast\_hand"

# ITEMS

# ORANGE SPRAY CAN

@onready var raycast\_spray = $"../Head/Camera/raycast\_spray"

@onready var spray\_orange\_spraying\_sound = $"../Head/Camera/item\_display/spray\_orange/spraying\_sound"

@onready var spray\_orange = $"../Head/Camera/item\_display/spray\_orange"

@onready var spray\_orange\_spraying\_effect = $"../Head/Camera/item\_display/spray\_orange/spraying\_effect"

@onready var spray\_orange\_sound\_empty = $"../Head/Camera/item\_display/spray\_orange/spray\_empty"

# SHOTGUN

@onready var shotgun = $"../Head/Camera/item\_display/shotgun"

@onready var shotgun\_anim\_reload = $"../Head/Camera/item\_display/shotgun/anim\_reload"

@onready var shotgun\_anim\_shot = $"../Head/Camera/item\_display/shotgun/shot"

@onready var raycast\_shotgun\_1 = $"../Head/Camera/raycast\_shotgun\_1"

@onready var raycast\_shotgun\_2 = $"../Head/Camera/raycast\_shotgun\_2"

@onready var raycast\_shotgun\_3 = $"../Head/Camera/raycast\_shotgun\_3"

@onready var raycast\_shotgun\_4 = $"../Head/Camera/raycast\_shotgun\_4"

@onready var raycast\_shotgun\_5 = $"../Head/Camera/raycast\_shotgun\_5"

@onready var no\_ammo = $"../Head/Camera/item\_display/shotgun/no\_ammo"

# MILK

@onready var milk = $"../Head/Camera/item\_display/milk"

@onready var milk\_anim\_consume = $"../Head/Camera/item\_display/milk/anim\_consume"

# COFFEE CAN

@onready var coffee\_can = $"../Head/Camera/item\_display/coffee\_can"

@onready var coffee\_can\_anim\_consume = $"../Head/Camera/item\_display/coffee\_can/anim\_drink\_coffee\_can"

func \_input(event):

if event.is\_action\_pressed('key\_r'):

if shotgun.visible:

shotgun\_try\_reload()

if event.is\_action\_pressed('lmb'):

if shotgun.visible:

shotgun\_shoot()

if milk.visible:

drink\_milk()

if coffee\_can.visible:

drink\_coffee()

if spray\_orange.visible:

if not SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['paint\_amount'] > 0:

spray\_orange\_sound\_empty.play()

func \_physics\_process(delta):

# ACTIVATE ITEMS IN HAND (CONTINUOUS)

# PAINT ORANGE

if Input.is\_action\_pressed('lmb'):

if spray\_orange.visible:

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['paint\_amount'] > 0:

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['paint\_amount'] -= 0.005

spray\_orange\_spraying\_effect.emitting = true

if not spray\_orange\_spraying\_sound.is\_playing():

spray\_orange\_spraying\_sound.play()

if raycast\_spray.is\_colliding() and not raycast\_spray.get\_collider() == null:

var paint = load("res://objects/EFFECTS/spray\_paint\_orange/spray\_paint\_orange.tscn").instantiate()

raycast\_spray.get\_collider().add\_child(paint)

paint.global\_position = raycast\_spray.get\_collision\_point()

else:

spray\_orange\_spraying\_effect.emitting = false

else:

spray\_orange\_spraying\_effect.emitting = false

# ============================ SERVICE FUNCS ===================================

# ===== SHOTGUN

func shotgun\_try\_reload():

if not shotgun\_anim\_reload.is\_playing():

var isSuccessReload: bool = false

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['ammo\_inside'] < 5:

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_1']['id'] == 'shell':

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_1'] = {'id': 'void'}

isSuccessReload = true

elif SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_2']['id'] == 'shell':

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_2'] = {'id': 'void'}

isSuccessReload = true

elif SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_3']['id'] == 'shell':

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_3'] = {'id': 'void'}

isSuccessReload = true

elif SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_4']['id'] == 'shell':

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_4'] = {'id': 'void'}

isSuccessReload = true

if isSuccessReload:

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['ammo\_inside'] += 1

inventory\_loader.load\_inventory\_visual()

shotgun\_anim\_reload.play('reload')

func shotgun\_shoot():

if shotgun.visible:

if not shotgun\_anim\_shot.is\_playing():

if SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['ammo\_inside'] > 0:

var spread: int = 10

raycast\_shotgun\_1.rotation.x = deg\_to\_rad(0)

raycast\_shotgun\_1.rotation.y = deg\_to\_rad(0)

raycast\_shotgun\_2.rotation.x = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_2.rotation.y = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_3.rotation.x = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_3.rotation.y = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_4.rotation.x = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_4.rotation.y = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_5.rotation.x = deg\_to\_rad(randi\_range(-spread,spread))

raycast\_shotgun\_5.rotation.y = deg\_to\_rad(randi\_range(-spread,spread))

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)]['ammo\_inside'] -= 1

shotgun\_anim\_shot.play('shot')

var sound\_id: int = randi\_range(1,5)

get\_node("../Head/Camera/item\_display/shotgun/fire\_" + str(sound\_id)).play()

for i in range(5):

var raycast = get\_node("../Head/Camera/raycast\_shotgun\_"+str(i+1))

if raycast.is\_colliding():

if not raycast.get\_collider() == null:

if raycast.get\_collider().has\_method('hurt'):

raycast.get\_collider().hurt()

else:

no\_ammo.play()

# ===== MILK

func drink\_milk():

player.stamina = 100.0

milk\_anim\_consume.play('consume')

await milk\_anim\_consume.animation\_finished

milk\_anim\_consume.play('RESET')

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)] = {'id': 'void'}

inventory\_loader.load\_inventory\_visual()

inventory\_loader.load\_hand\_visual(player.current\_slot\_selected)

func drink\_coffee():

player.stamina = 100.0

coffee\_can\_anim\_consume.play('consume')

await coffee\_can\_anim\_consume.animation\_finished

coffee\_can\_anim\_consume.play('RESET')

SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']['slot\_'+str(player.current\_slot\_selected)] = {'id': 'void'}

inventory\_loader.load\_inventory\_visual()

inventory\_loader.load\_hand\_visual(player.current\_slot\_selected)

**INVENTORY\_LOADER.gd**

extends Node

@onready var item\_1\_icon = $"../UI/UI/inventory/slot\_1/item\_icon"

@onready var item\_2\_icon = $"../UI/UI/inventory/slot\_2/item\_icon"

@onready var item\_3\_icon = $"../UI/UI/inventory/slot\_3/item\_icon"

@onready var item\_4\_icon = $"../UI/UI/inventory/slot\_4/item\_icon"

func load\_inventory\_visual():

var inv: Dictionary = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']

for i in range(4):

match inv['slot\_' + str(i+1)].id:

# space is free

'void':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = null

'box':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_box.png')

'letter':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_letter.png')

'ball':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_ball.png')

'fuel\_tank':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_fuel\_tank.png')

'shotgun':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_shotgun.png')

'shell':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_shell.png')

'milk':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_milk.png')

'spray\_orange':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_spray.png')

'coffee\_can':

get\_node("../UI/UI/inventory/slot\_" + str(i+1) + "/item\_icon").texture = load('res://assets/images/item\_coffee\_can.png')

func load\_hand\_visual(slot\_id):

$"../UI/UI/item\_name".text = ''

var inv: Dictionary = SIN\_WORLD\_DATA.WORLD\_DATA['player\_inv']

var item = inv['slot\_' + str(slot\_id)]

$"../Head/Camera/item\_display/letter".visible = false

$"../Head/Camera/item\_display/box".visible = false

$"../Head/Camera/item\_display/ball".visible = false

$"../Head/Camera/item\_display/fuel\_tank".visible = false

$"../Head/Camera/item\_display/shotgun".visible = false

$"../Head/Camera/item\_display/shell".visible = false

$"../Head/Camera/item\_display/milk".visible = false

$"../Head/Camera/item\_display/spray\_orange".visible = false

$"../Head/Camera/item\_display/coffee\_can".visible = false

match item.id:

'void':

$"../UI/UI/item\_name".text = ''

'letter':

$"../Head/Camera/item\_display/letter".visible = true

$"../UI/UI/item\_name".text = item['name']

'box':

$"../Head/Camera/item\_display/box".visible = true

$"../UI/UI/item\_name".text = item['name']

'ball':

$"../Head/Camera/item\_display/ball".visible = true

$"../UI/UI/item\_name".text = item['name']

'fuel\_tank':

$"../Head/Camera/item\_display/fuel\_tank".visible = true

$"../UI/UI/item\_name".text = item['name']

'shotgun':

$"../Head/Camera/item\_display/shotgun".visible = true

$"../UI/UI/item\_name".text = item['name']

'shell':

$"../Head/Camera/item\_display/shell".visible = true

$"../UI/UI/item\_name".text = item['name']

'milk':

$"../Head/Camera/item\_display/milk".visible = true

$"../UI/UI/item\_name".text = item['name']

'spray\_orange':

$"../Head/Camera/item\_display/spray\_orange".visible = true

$"../UI/UI/item\_name".text = item['name']

'coffee\_can':

$"../Head/Camera/item\_display/coffee\_can".visible = true

$"../UI/UI/item\_name".text = item['name']