Министерство образования Калининградской области

государственное автономное учреждение

Калининградской области

профессиональная образовательная организация

«Колледж предпринимательства»

**ДИПЛОМНАЯ РАБОТА**

**Тема: Разработка компьютерной игры “RTS стратегия”**

|  |  |
| --- | --- |
| Выпускная квалификационная  (дипломная) работа  допущена к защите  Заместитель директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_ Бурыкина Ю.И.  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. | Выполнил:  обучающийся группы ИСП19-21  специальность 09.02.02 Компьютерные сети  \_\_\_\_\_\_\_\_\_\_\_ Родин А.А.  Руководитель:  \_\_\_\_\_\_\_\_\_\_\_ Мамаев П.В. |

Калининград

2023**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc137499160)

[1.Теоретическая часть 4](#_Toc137499161)

[1.1 объектно-ориентированный язык программирования общего назначения C# 6](#_Toc137499162)

[1.2 Navigation Mesh 7](#_Toc137499163)

[1.3 Жанр Стратегии 8](#_Toc137499164)

[1.4 Классификация стратегий 8](#_Toc137499165)

[2. Реализация дипломной работы 16](#_Toc137499166)

[2.1 Техническое задание 16](#_Toc137499167)

[2.2 Работа кода 19](#_Toc137499168)

[2.3 Создание моделей 49](#_Toc137499169)

[ЗАКЛЮЧЕНИЕ 55](#_Toc137499170)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 56](#_Toc137499171)

[ПРИЛОЖЕНИЯ 57](#_Toc137499172)

ВВЕДЕНИЕ

В современном мире большую популярность набирает индустрия видео игр, при это самыми популярными направлениями являются многопользовательские, условно бесплатные (с возможностью внутри игровых транзакций) и с элементами коллекционирования виртуальных объектов проекты. Из-за чего сегмент однопользовательских проектов, ориентированных на долгое погружение в историю, изучение механик и бросающих вызов пользователю вымирает. Одними из таких частей стали стратегии – упрощение, попытка ввести внутри игровые транзакции, и концентрация на визуальной части привели к падению продаж, потери целевой аудитории у крупных издателей, и как результат серьёзное падение популярности жанра. Так же проекты данного жанра считаются экономически не выгодными, ввиду больших затрат для производства и малого спроса, так как целевая аудитория не доверяет крупным издателям и внимательно следит за публикуемой информацией о проектах. При этом спрос на стратегии не упал, что послужило хорошей почвой для маленьких команд разработчиков, что выпускают одну игру по шаблонам старых игр или же меняют привычные механики на совсем новые и необычные.

Целью данной работы является создание видео игры в жанре Стратегии, в под жанре RTS, по шаблону старых RTS. Были поставлены следующие цели:

* создание базовых механик передвижения, управления и стрельбы у игровых пешек;
* создание игрового интерфейса;
* создание механик строительства, ремонта и захвата строений;
* реализация созданных механик и интерфейса в проект;

1.Теоретическая часть

Для осуществления, запланированного необходимо выбрать “игровой движок” – базовое [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [компьютерной игры](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), которое пригодно для повторного использования и расширения, и тем самым может быть рассмотрено как основание для разработки множества различных игр без существенных изменений.

Среди вариантов были:

**Unity** (*unity* в переводе с [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) — «единство», произносится как «ю́нити») — [кроссплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), разработанная американской компанией [Unity Technologies](https://ru.wikipedia.org/wiki/Unity_Technologies). Написан на С++, С#. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих [персональные компьютеры](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), [игровые консоли](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0), [мобильные устройства](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE), [интернет-приложения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и другие. Выпуск Unity состоялся в [2005](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4_%D0%B2_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D1%85_%D0%B8%D0%B3%D1%80%D0%B0%D1%85) году и с того времени идёт постоянное развитие.

Основными преимуществами Unity являются наличие [визуальной среды разработки](https://ru.wikipedia.org/wiki/RAD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [межплатформенной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних [библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).

На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и [независимыми студиями](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B8-%D0%B8%D0%B3%D1%80%D0%B0). В итоге простой интерфейс и удобные функции, но подходит в основном для небольших игр.

**Unreal Engine** — базовое программное обеспечение компьютерной игры, разработанное и поддерживаемое компанией [Epic Games](https://ru.wikipedia.org/wiki/Epic_Games). Написано на языке программирования [C++](https://ru.wikipedia.org/wiki/C%2B%2B), программное обеспечение позволяет создавать игры для большинства [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и платформ, таки как: [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows), [Linux](https://ru.wikipedia.org/wiki/Linux), [Mac OS](https://ru.wikipedia.org/wiki/Mac_OS) и [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X); [консолей](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0) [Xbox](https://ru.wikipedia.org/wiki/Xbox), [Xbox 360](https://ru.wikipedia.org/wiki/Xbox_360), [Xbox One](https://ru.wikipedia.org/wiki/Xbox_One), [PlayStation 2](https://ru.wikipedia.org/wiki/PlayStation_2), [PlayStation 3](https://ru.wikipedia.org/wiki/PlayStation_3), [PlayStation 4](https://ru.wikipedia.org/wiki/PlayStation_4), [PlayStation 5](https://ru.wikipedia.org/wiki/PlayStation_5), [PSP](https://ru.wikipedia.org/wiki/PlayStation_Portable), [PS Vita](https://ru.wikipedia.org/wiki/PS_Vita), [Wii](https://ru.wikipedia.org/wiki/Wii), [Dreamcast](https://ru.wikipedia.org/wiki/Dreamcast), [GameCube](https://ru.wikipedia.org/wiki/Nintendo_GameCube) и др., а также на различных портативных устройствах, например, устройствах [Apple](https://ru.wikipedia.org/wiki/Apple) ([iPad](https://ru.wikipedia.org/wiki/IPad), [iPhone](https://ru.wikipedia.org/wiki/IPhone)), управляемых системой [iOS](https://ru.wikipedia.org/wiki/IOS) и прочих. (Впервые работа с [iOS](https://ru.wikipedia.org/wiki/IOS) была представлена в 2009 году, в 2010 году продемонстрирована работа движка на устройстве с системой [webOS](https://ru.wikipedia.org/wiki/Palm_webOS)).

Для упрощения адаптирования программного обеспечения используется модульная система зависимых компонентов; поддерживает различные системы получения изображения ([Direct3D](https://ru.wikipedia.org/wiki/Direct3D), [OpenGL](https://ru.wikipedia.org/wiki/OpenGL), [Pixomatic](https://ru.wikipedia.org/w/index.php?title=Pixomatic&action=edit&redlink=1); в ранних версиях: [Glide](https://ru.wikipedia.org/w/index.php?title=Glide_API&action=edit&redlink=1), [S3](https://ru.wikipedia.org/w/index.php?title=S3_Metal&action=edit&redlink=1), [PowerVR](https://ru.wikipedia.org/w/index.php?title=PowerVR_SGL&action=edit&redlink=1)), воспроизведения звука ([EAX](https://ru.wikipedia.org/wiki/Environmental_Audio_Extensions), [OpenAL](https://ru.wikipedia.org/wiki/OpenAL), [DirectSound3D](https://ru.wikipedia.org/wiki/DirectSound3D); ранее: [A3D](https://ru.wikipedia.org/w/index.php?title=A3D&action=edit&redlink=1)), средства голосового воспроизведения текста, распознавание речи, модули для работы с сетью и поддержки различных устройств ввода.

Для игры по сети поддерживаются технологии [Windows Live](https://ru.wikipedia.org/wiki/Windows_Live), [Xbox Live](https://ru.wikipedia.org/wiki/Xbox_Live), [GameSpy](https://ru.wikipedia.org/wiki/GameSpy) и прочие, включая до 64 игроков (клиентов) одновременно. Таким образом, движок адаптировали и для применения в играх жанра [MMORPG](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD-%D0%B8%D0%B3%D1%80%D0%B0). В итоге возможность реализации качественной графики, хорошая масштабируемость.

**Godot Engine** ([фр.](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B0%D0%BD%D1%86%D1%83%D0%B7%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *gó-doh* ) — [открыт](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)ое [кроссплатформенн](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C)ое 2D и 3D базовое программное обеспечение компьютерной игры под [лицензией MIT](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_MIT), который разрабатывается сообществом Godot Engine Community.

Задача Godot — быть максимально интегрированной и самодостаточной средой для [разработки игр](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B2%D0%B8%D0%B4%D0%B5%D0%BE%D0%B8%D0%B3%D1%80). Среда позволяет разработчикам создавать игры с нуля, не пользуясь более никакими инструментами, за исключением тех, которые необходимы для создания игрового контента (элементы графики, музыкальные треки и т. д.). Процесс программирования также не требует внешних инструментов (хотя при необходимости использовать внешний редактор, это можно сделать относительно легко).

Общая архитектура движка построена вокруг концепции [дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) из наследуемых «сцен». Каждый элемент сцены ([нода](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D1%88%D0%B8%D0%BD%D0%B0_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2))), в любой момент сам может стать полноценной сценой. Поэтому при разработке можно легко изменять полностью всю архитектуру проекта, расширять её элементы в любую сторону и работать с комплексными сценами на уровне простых абстракций.

Все игровые ресурсы, от скриптов до графических ассетов и игровых сцен, хранятся в папке проекта как обычные файлы, и не являются частью сложной базы данных проекта. Ресурсы, которые не представляют собой комплексных данных, хранятся в простых текстовых форматах (например, скрипты и сцены, в отличие от моделей и текстур). Эти решения позволяют значительно упростить разным командам разработчиков работу с [системами управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8). В итоге самодостаточен, бесплатен, но плохо подходит для создания 3д игр.

Исходя из поставленной задачи был выбран UNITY, ввиду простого интерфейса и удобных функций для разработчика. Исходя из этого игра будет написана на языке программирования C#.

1.1 [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения C#

C# (произносится *си шарп*) — [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения. Разработан в [1998](https://ru.wikipedia.org/wiki/1998)—[2001 годах](https://ru.wikipedia.org/wiki/2001_%D0%B3%D0%BE%D0%B4) группой инженеров компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) под руководством [Андерса Хейлсберга](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%B9%D0%BB%D1%81%D0%B1%D0%B5%D1%80%D0%B3,_%D0%90%D0%BD%D0%B4%D0%B5%D1%80%D1%81) и Скотта Вильтаумота как язык разработки приложений для платформы [Microsoft .NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) и [.NET Core](https://ru.wikipedia.org/wiki/.NET_Core). Впоследствии был стандартизирован как [ECMA](https://ru.wikipedia.org/wiki/ECMA)-334 и [ISO](https://ru.wikipedia.org/wiki/ISO)/[IEC](https://ru.wikipedia.org/wiki/IEC) 23270.

C# относится к семье языков с [C-подобным синтаксисом](https://ru.wikipedia.org/wiki/C-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D1%8B%D0%B9_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81), из них его синтаксис наиболее близок к [C++](https://ru.wikipedia.org/wiki/C%2B%2B) и [Java](https://ru.wikipedia.org/wiki/Java). Язык имеет [статическую типизацию](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), поддерживает [полиморфизм](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [перегрузку операторов](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2) (в том числе операторов явного и неявного приведения типа), [делегаты](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%B3%D0%B0%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), атрибуты, [события](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [переменные](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [свойства](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [обобщённые](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) типы и методы, [итераторы](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80), [анонимные функции](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%BE%D0%BD%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) с поддержкой [замыканий](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [LINQ](https://ru.wikipedia.org/wiki/Language_Integrated_Query), [исключения](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), [комментарии](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%B8%D0%B8_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) в формате [XML](https://ru.wikipedia.org/wiki/XML).

Переняв многое от своих предшественников — языков [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Модула](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0-2), [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk) и, в особенности, [Java](https://ru.wikipedia.org/wiki/Java) — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает [множественное наследование](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) классов (между тем допускается [множественная реализация интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)#%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B8_%D1%80%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%8)).

1.2 Navigation Mesh

Навигационная сетка — это набор двумерных [многоугольников](https://en.wikipedia.org/wiki/Convex_polygon) ( [полигональная сетка](https://en.wikipedia.org/wiki/Polygon_mesh) ), которые определяют, какие области среды могут проходить агенты. Другими словами, персонаж в игре может свободно ходить по этим областям, не встречая препятствий из-за деревьев, лавы или других барьеров, являющихся частью окружающей среды. Соседние полигоны соединяются друг с другом в [графе](https://en.wikipedia.org/wiki/Graph_data_structure) .

Поиск пути в пределах одного из этих многоугольников может быть тривиально выполнен по прямой линии, потому что многоугольник выпуклый и проходимый. Поиск пути между полигонами в сетке можно выполнить с помощью одного из большого количества алгоритмов [поиска по графу](https://en.wikipedia.org/wiki/Graph_search). Таким образом, агенты в навигационной сетке могут избежать дорогостоящих проверок [обнаружения столкновений](https://en.wikipedia.org/wiki/Collision_detection) с препятствиями, которые являются частью среды.

Представление проходимых областей в двумерной форме упрощает расчеты, которые в противном случае пришлось бы выполнять в «настоящей» трехмерной среде, но, в отличие от двумерной сетки, позволяет проходимым областям перекрываться сверху и снизу на разной высоте. Многоугольники различных размеров и форм в навигационных сетках могут отображать произвольные среды с большей точностью, чем обычные сетки.

1.3 Жанр Стратегии

Стратегическая компьютерная игра ([разг.](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D1%80%D0%BD%D1%8B%D0%B9_%D1%81%D1%82%D0%B8%D0%BB%D1%8C) «Стратегия») — один из основных жанров [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), в котором игроку для победы необходимо применять [стратегическое мышление](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F). В популярных играх такого жанра игроку часто предлагается играть не за конкретного персонажа, а за их условные массы, к примеру, руководить строительством города или командовать целыми армиями в военных кампаниях.

Хотя многие другие типы видеоигр могут содержать стратегические элементы, в собственно стратегических играх основное внимание уделяется планированию высшего уровня, логистике и управлению ресурсами. Стратегии обычно делятся на две основные подкатегории: [пошаговые](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D0%B0%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F) и [в реальном времени](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8), но есть также ряд кросс-/поджанров стратегий, которые содержат дополнительные элементы, такие как тактика, дипломатия, экономика и исследование.

К стратегическим играм относят также абстрактные ([*Ataxx*](https://en.wikipedia.org/wiki/Ataxx) (англ.)[рус.](https://ru.wikipedia.org/w/index.php?title=Ataxx&action=edit&redlink=1)), настольные ([шахматы](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D1%85%D0%BC%D0%B0%D1%82%D1%8B), [шашки](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D1%88%D0%BA%D0%B8), «[Монополия](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%BE%D0%BF%D0%BE%D0%BB%D0%B8%D1%8F_(%D0%B8%D0%B3%D1%80%D0%B0))»), с симуляцией менеджмента (“[*M.U.L.E.*](https://ru.wikipedia.org/wiki/M.U.L.E.)*”*, “[*Spaceward Ho!*](https://en.wikipedia.org/wiki/Spaceward_Ho!)*”* (англ.)[рус.](https://ru.wikipedia.org/w/index.php?title=Spaceward_Ho!&action=edit&redlink=1)) и другие; особенности некоторых из них были переняты разработчиками сюжетных компьютерных стратегий в реалистичном, историческом или фантастическом [сеттингах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D1%82%D0%B8%D0%BD%D0%B3).

1.4 Классификация стратегий

Все компьютерные стратегии относятся к двум основным типам:

* стратегии в реальном времени (RTS, от англ. Real Time Strategy)
* пошаговые стратегии (TBS, от англ. Turn Based Strategy).

Классификация по времени действия относится лишь к принципу временной организации в игре и не имеет никакого отношения к присутствию или отсутствию таких элементов геймплея, как строительство базы, добыча ресурсов и подобных.

Пошаговые стратегии/TBS

TBS от англ. Turn-Based Strategy. Пошаговые стратегии — игры, в которых игроки производят свои действия по очереди. Пошаговые стратегии появились раньше RTS и отличаются значительно большим разнообразием. Разделение игрового процесса на ходы отрывает его от реальной жизни и лишает игру динамизма, в результате чего эти игры не так популярны, как стратегии в реальном времени. С другой стороны, в TBS у игрока гораздо больше времени на размышление, во время совершения хода его ничто не торопит, что позволяет уделять больше времени планированию.

Особый подвид пошаговых стратегий представляют игра “Scorched Earth” и произошедшая от неё серия “Worms”. В них много общего с жанром аркады, логическое мышление и выбор правильных решений почти не требуется, игрок управляет очень небольшим числом персонажей (в “Worms” — командой из нескольких червяков) или вообще одним (“Scorched Earth”). Тем не менее, эти игры относят к пошаговым стратегиям.

Стратегии в реальном времени/RTS

RTS от англ. Real Time Strategy. В этих стратегиях игроки производят свои действия одновременно. Они появились несколько позже пошаговых, а первой получившей известность игрой этого жанра была “[Dune II](https://ru.wikipedia.org/wiki/Dune_II)” ([1992](https://ru.wikipedia.org/wiki/1992_%D0%B3%D0%BE%D0%B4_%D0%B2_%D0%B8%D0%B3%D1%80%D0%B0%D1%85)), сюжет которой основан на [одноимённом произведении](https://ru.wikipedia.org/wiki/%D0%94%D1%8E%D0%BD%D0%B0_(%D1%80%D0%BE%D0%BC%D0%B0%D0%BD)) [Фрэнка Герберта](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D1%80%D0%B1%D0%B5%D1%80%D1%82,_%D0%A4%D1%80%D1%8D%D0%BD%D0%BA). В настоящее время жанр стратегий в реальном времени стал весьма широк и включает игры совершенно разного типажа – от тактических варгеймов до глобальных стратегий, поэтому классические стратегии в реальном времени вроде “Dune” традиционно называют "классическими RTS". Особый подвид представляют собой стратегии в реальном времени с активной паузой, где игроку предоставляется возможность поставить игру на паузу и раздать необходимые приказы (например, “[Europa Universalis IV](https://ru.wikipedia.org/wiki/Europa_Universalis_IV)”).

Во времена выхода “[Dune II](https://ru.wikipedia.org/wiki/Dune_II)” сформировались общие принципы классических стратегий в реальном времени (часть из них общие с другими стратегиями):

* Экономика в игре носит вспомогательный характер и нацелена на сбор ресурсов;
* В основе экономики лежат здания, которые можно строить и разрушать. Именно они строят юниты (см. ниже) и проводят исследования. Иногда строительство производят особые юниты, иногда (как в “Dune”) — другие здания. Некоторые строения могут атаковать противника сразу после окончания строительства, некоторые после специальных усовершенствований;
* Сбор ресурсов осуществляется особыми юнитами в специально предусмотренных для этого местах (в “Dune” это поля [спайса](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D1%8F%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D0%94%D1%8E%D0%BD%D0%B0))), после чего они переносятся к специальному зданию на базе и только после этого поступают в распоряжение игрока;
* Ресурсы могут быть разных видов (например, золото, древесина, деньги, металл, уголь) и расходуются на строительство юнитов и зданий (также на это уходит время);
* Юнит (unit) — любая боевая единица (пехотинец, танк, самолёт, корабль), которая обычно может атаковать другие юниты и здания и уничтожать их. Юниты имеют параметры, главными среди которых являются «[очки здоровья](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%BA%D0%B8_%D0%B7%D0%B4%D0%BE%D1%80%D0%BE%D0%B2%D1%8C%D1%8F)», наносимые повреждения, броня (уменьшающая наносимый юниту урон), скорость. Юнит может включать в себя и несколько солдат (танков, артиллерийских орудий и т.д.), но в этом случае командовать ими по отдельности нельзя. Некоторые юниты не могут атаковать и предназначены для сбора ресурсов, строительства зданий, транспортировки других юнитов;
* Все здания и юниты имеют радиус обзора, дальше которого «видеть» они не могут;
* Игрок может видеть, что происходит только на тех территориях, что попадают в радиус обзора его строений и юнитов. Те территории, на которых он ещё не был, закрашены чёрным. Те, на которых его войска уже были, но в данный момент не могут их видеть, покрыты так называемым «[туманом войны](https://ru.wikipedia.org/wiki/%D0%A2%D1%83%D0%BC%D0%B0%D0%BD_%D0%B2%D0%BE%D0%B9%D0%BD%D1%8B)» (“Fog of War”), который обычно позволяет видеть ландшафт этой местности и находящиеся там здания (такими, какими они были в последний момент, когда их видели юниты игрока), но не позволяет видеть передвижения противника и связанные с этим возможные изменения ландшафта.

**Тактические** стратегии

Тактические стратегии представляют собой симуляцию тактики как в реальном времени, так и в пошаговом режиме, тем самым подразделяясь на жанры [RTT](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8) (Real-Time Tactics) и [TBT](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D0%B0%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D1%82%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0) (Turn-Based Tactics). В отличие от классических RTS, в данных стратегиях игроку не приходится заниматься менеджментом базы или добычей ресурсов, тем самым данные игры сконцентрированы в большей степени на тактике ведения боевых действий в условиях заранее заданного количества войск и/или ресурсов. Возможность получения подкреплений и дополнительных ресурсов в таких играх либо отсутствует вовсе, либо на ней не делается основной акцент. Примеры: “[World in Conflict](https://ru.wikipedia.org/wiki/World_in_Conflict)”, “Achtung! Cthulhu Tactics”, серия “[Jagged Alliance](https://ru.wikipedia.org/wiki/Jagged_Alliance_(%D1%81%D0%B5%D1%80%D0%B8%D1%8F_%D0%B8%D0%B3%D1%80))”.

Браузерные онлайн-стратегии

Стратегии, сочетающие основные элементы классической [RTS](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8) с возможностью коллективной игры с большим количеством игроков по сети [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). Примерами браузерных онлайн-стратегий могут служить игры “[Травиан](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%B2%D0%B8%D0%B0%D0%BD)”, “[My Lands](https://ru.wikipedia.org/wiki/My_Lands)”, “[OGame](https://ru.wikipedia.org/wiki/OGame)”, “[Сфера Судьбы](https://ru.wikipedia.org/wiki/%D0%A1%D1%84%D0%B5%D1%80%D0%B0_%D0%A1%D1%83%D0%B4%D1%8C%D0%B1%D1%8B)”, “[Война племён](https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D0%B9%D0%BD%D0%B0_%D0%BF%D0%BB%D0%B5%D0%BC%D1%91%D0%BD)”.

#### Симуляторы строительства и управления

Симуляторы строительства и управления – жанр компьютерных игр, стоящий на стыке симуляторов и стратегий. Они могут быть посвящены строительству города, парка развлечений и других объектов, при этом предлагая экономические испытания в противовес военным и тактическим. Условно их можно разделить на несколько поджанров.

Градостроительные стратегии

Градостроительные стратегии (англ. city-building games) посвящены строительству городов и поселений. В них игрок занимается строительством собственного города и решает связанные с этим задачи и испытания – например, строит жильё и борется с автомобильными пробками.

Экономические стратегии

Экономические стратегии (англ. business simulation games; tycoon) — стратегии, в которых игрок занимается созданием и управлением собственным бизнесом. В таких стратегиях обычно более проработана экономическая составляющая, на которой и делается основной акцент. Иногда в таких стратегиях присутствуют соперники (конкуренты), но конкуренция с ними, в основном, экономическая.

Экономические стратегии бывают однопользовательские, в которых рыночная среда полностью моделируется только запрограммированными алгоритмами, и многопользовательские, в которых конкурентное окружение и рыночная среда формируется не только алгоритмами, но и в результате взаимодействия множества игроков, что делает виртуальную экономику и её поведение более реалистичным и непредсказуемым по сравнению с однопользовательскими играми. Одним из наиболее известных классических однопользовательских экономических симуляторов является игра “Capitalism” Тревора Чена. Среди многопользовательских экономических симуляторов можно выделить серию экономических браузерных игр от издателя [Виртономика](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D0%BA%D0%B0), в которой пользователи развивают свой виртуальный бизнес в конкуренции и кооперации со множеством игроков, строят города, создают политические партии и управляют виртуальными государствами.

Наиболее распространены симуляторы бизнеса («тайкуны», от часто встречающегося в их названиях слова tycoon — «магнат»). Первой игрой подобного рода была игра [Сида Мейера](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B9%D0%B5%D1%80,_%D0%A1%D0%B8%D0%B4) “Railroad Tycoon”. В симуляторах бизнеса играющий управляет как огромными промышленными компаниями (“[Capitalism II](https://ru.wikipedia.org/wiki/Capitalism_II)”, “[Industry Giant II](https://ru.wikipedia.org/wiki/Industry_Giant_II)”), так и более мелкими предприятиями, такими, как закусочная или зоопарк (такие симуляторы носят менее серьёзный характер и часто предназначены не для того, чтобы игрок долго просчитывал ситуации и принимал выгодные решения, а просто для его развлечения; проиграть в таких играх нелегко, так как противников в явном виде нет). Очень распространены транспортные экономические симуляторы, такие, как “[Transport Tycoon](https://ru.wikipedia.org/wiki/Transport_Tycoon)”.

Среди других видов экономических стратегий можно отметить такие поджанры, как [градостроительные симуляторы](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%81%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80) (наиболее известна породившая этот жанр игра “[SimCity](https://ru.wikipedia.org/wiki/SimCity)”, а также серия градостроительных игр, действие которых происходит в древнем мире, — “[Caesar III](https://ru.wikipedia.org/wiki/Caesar_III)”, “[Zeus: Master of Olympus](https://ru.wikipedia.org/wiki/Zeus:_Master_of_Olympus)”, и прочие; в последней серии присутствуют военные действия), спортивные менеджеры (в которых игрок руководит спортивным клубом, обычно футбольным, но не участвует в матчах непосредственно, а выполняет роль тренера и менеджера команды), симуляторы жизни (очень специфический жанр, происходящий от игры “[The Sims](https://ru.wikipedia.org/wiki/The_Sims)”).

В связи со всеобщим глобальным развитием сети Интернет набирают силу экономические онлайн игры, то есть игры представляющие собой некий мир со своими правилами и законами, в котором игроку предоставляется возможность заниматься активной коммерческой деятельностью: строить торговые и производственные предприятия и сооружения, заниматься разработкой природных ресурсов, вести активную финансовую и банковскую деятельность в рамках игрового пространства.

Развитие технологий дистанционного онлайн обучения дали заметный толчок к использованию экономических онлайн игр и симуляторов в образовательных процессах. Корпорации и бизнес-школы начали активно использовать обучающие [экономические игры](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B) такие как “CapSim” или “[Simformer](https://ru.wikipedia.org/w/index.php?title=Simformer&action=edit&redlink=1)” в обучении студентов и менеджеров.

Варгеймы

Стратегии, в которых отсутствует экономическая составляющая. Обычно это игры, в которых играющий управляет отрядом или армией во время боя. Примером из популярных игр могут служить некоторые [моды](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B)) для “[Sudden Strike](https://ru.wikipedia.org/wiki/Sudden_Strike)” и “[Блицкрига](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%B8%D1%86%D0%BA%D1%80%D0%B8%D0%B3_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0))”.

«Варгейм» — не жанр, а характеристика конкретной игры, которая говорит — вам не придётся считать в ней деньги, большую часть игрового времени вы будете воевать всеми доступными средствами. А вот какие средства окажутся вам доступны, и что надо для этого сделать, зависит от фантазии разработчиков.

Примером такой игры является “[Z](https://ru.wikipedia.org/wiki/Z_(%D0%B8%D0%B3%D1%80%D0%B0))” [1996 года](https://ru.wikipedia.org/wiki/1996_%D0%B3%D0%BE%D0%B4_%D0%B2_%D0%B8%D0%B3%D1%80%D0%B0%D1%85) от английской компании “[The Bitmap Brothers](https://ru.wikipedia.org/wiki/The_Bitmap_Brothers)”. Несмотря на полное отсутствие экономики, игроки без проблем повышают качество и количество своих отрядов, умело распоряжаясь двумя важнейшими игровыми ресурсами — территорией и временем. Для стратегического планирования товарно-денежные отношения вовсе не обязательны, хотя, традиционны. Сам жанр остаётся всё той же стратегией или тактикой.

Глобальные стратегии

В глобальных стратегиях игрок принимает участие в целом периоде мировой истории, управляя государством; при этом игра учитывает различные факторы развития государств — экономический, военный, культурный, социальный и политический. Выделение глобальных стратегий в единый жанр характерно для русскоязычных специалистов; на [Западе](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B8%D1%80) этот жанр обычно разделяют на два — [4X](https://ru.wikipedia.org/wiki/4X) и [grand strategy](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D0%B1%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F#%D0%A2%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F). Глобальные стратегии могут быть как в реальном времени, так и пошаговые.

**Стратегии модели 4X**

Геймплей 4Х стратегий включает 4 полноценно реализованных элемента: исследование (новых земель), расширение, эксплуатация, уничтожение. Название происходит от буквы X в соответствующих данным понятиям английских словах eXplore, eXpand, eXploit, eXterminate. Классическим примером 4X стратегий является серия “[Civilization](https://ru.wikipedia.org/wiki/Civilization)”. В 4X стратегиях, также, как и в глобальных, игрок зачастую управляет целым государством. Однако это не является обязательным критерием. 4X стратегии могут быть как в реальном времени, так и пошаговые.

Отличие 4X стратегий от глобальных стратегий

Отличие 4X стратегий от глобальных стратегий состоит в том, что в глобальных стратегиях может отсутствовать или быть слабо выраженным один из X-элементов. Например, глобальная стратегия “Hearts of Iron IV” не содержит в себе элемента eXplore (исследование новых земель). Это означает, что стратегия может быть одновременно 4X и глобальной, но не каждая глобальная стратегия является 4X.

MOBA

Игры жанра MOBA ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Multiplayer Online Battle Arena) сочетают в себе элементы [стратегий в реальном времени](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8) и [компьютерных ролевых игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0). Этот жанр возник как ответвление (через [модификации](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B))) стратегий в реальном времени, и многие игры жанра MOBA используют схожий с ними интерфейс, но отличаются особыми правилами сражения.

В играх жанра MOBA две команды игроков сражаются друг с другом на карте особого вида. Каждый игрок управляет одним персонажем из определённого списка доступных героев, отличающихся способностями. В течение матча персонажи могут становиться сильнее, получать новые способности и снаряжение, подобно [компьютерным ролевым играм](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0). Конечной целью в ходе матча является уничтожение главного здания вражеской команды с помощью как героев, управляемых игроками, так и юнитов, управляемых компьютером.

#### Компьютерная адаптация настольных карточные стратегий

Игры, схожие с [настольными](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D1%82%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0) [коллекционными карточными играми](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0). Пример: “[Magic The Gathering](https://ru.wikipedia.org/wiki/Magic_The_Gathering).”

2. Реализация дипломной работы

2.1 Техническое задание

Техническое задание

* Игрок должен иметь возможность управлять игровой камерой с помощью клавиатуры.
* Игрок должен иметь возможность устанавливать на игровую сцену свои здания. Каждое здание должно иметь свой функцию, производство игровую технику на стороне игрока, производство внутри игровой валюты, защита территории.
* Игрок должен иметь искусственного оппонента, который умеет управлять своей техникой и зданиями.
* Игрок должен иметь возможность управлять своей техникой.
* Игрок должен иметь возможность выйти из игры в любой момент через специальное меню.

Техническое задание по созданию модель управления игровой камерой:

* Перемещение по осям x и z, изменение угла наклона по осям x и y.

Техническое задание по самостоятельному определению врагов:

* Создание хранилища – файла, в которой будет хранится название команды к которой принадлежит игровая техника или здание, количества здоровья и цена.
* Создание у боевой игровой техники и зданий прозрачной сферы – поля видимости.
* Создание у боевой игровой техники и зданий алгоритма, проверяющего хранилище игровых объектов, попадающих в поле видимости и сверяющего хранилищем у боевой игровой техники или зданий, к которым алгоритм прикреплён. Если параметр хранящий название команды, к которой игровая техника или здание принадлежат разные то, передача проверяемого игрового объекта в алгоритм отвечающего за стрельбу. Алгоритм должен исключать из проверки объекты без хранилища.

Техническое задание по стрельбе игровой техники или зданий по противнику:

* Орудие должно наводится на противника, в зависимости от особенностей техники или здания. Если орудие не принадлежит к артиллерийской технике или защитой турели, то орудие наводится прямо на цель.
* Артиллерийская техника наводит орудие на сорок пять градусов по оси x и высчитывает скорость необходимую для достижения снарядом цели.
* Защитная турель наводит орудие на десять градусов по оси x и высчитывает скорость необходимую для достижения снарядом цели.

Техническое задание по производству игровой техники фабриками:

* Создание пользовательского интерфейса для выбора игровой техники доступной для производства.
* Создание алгоритма кнопок пользовательского интерфейса, передающего в фабрику информацию о выбранной технике.
* Создание алгоритма для фабрики, создающего на игровую технику и задающего в её хранилище информацию о команде, которую берёт из хранилища фабрики. При создании техники алгоритм должен проверить количество внутри игровой валюты у игрока и сравнить со стоимостью техники, если цена меньше или равна – создать технику и вычесть из средств игрока, если цена больше – сообщить игроку о нехватке средств.
* Создание алгоритма, включающего и выключающего интерфейс фабрики, если игрок нажал левой кнопкой мыши на фабрику.
* Создание второго алгоритма создания техники без проверки на средства для искусственного противника.

Техническое задание по созданию искусственного противника:

* Разместить на игровом поле точки маршрутов для техники противника.
* Разместить на игровом поле строения противника.
* Создать алгоритм, собирающий все фабрики противника в список и запрашивающий у них создание техники.
* Создать алгоритм, собирающий всю технику искусственного противника в списки и отправляющий её по точкам маршрута, которые надо ввести в виде списка в алгоритм.

Техническое задание по выбору игровой техники или зданий:

* Необходимо реализовать несколько способов выбора игровой техники или зданий.
* Игрок должен навести курсор мыши на технику или здание и нажать левую кнопку мыши, после чего параметр “выбран” главного алгоритма этой техники или здания, отвечающий за состояние “выбран/не выбран” становится правда, при этом у другой игровой техники или зданий параметр “выбран” становится ложь. Если игрок при выборе игровой техники или зданий зажал кнопку “Shift”, то параметр “выбран” у другой техники не изменяется.
* Если игрок зажал левую кнопку мыши и повел курсор мыши, то создаётся четырёх угольный элемент интерфейса “рамка”, одна точка которого лежит в месте где был курсор мыши, когда игрок зажал левую кнопку мыши, противоположная ей точка находится на курсоре мышки и следует за ним. При нажатии правой кнопки мыши у всей техники или зданий, находящиеся внутри “рамки” параметр “выбран” главного алгоритма становится правда, у остальной – ложь, сама “рамка” удаляется.

Техническое задание по управлению игровой техникой:

* Техника должна проверять параметр “выбран”, если его значение правда и игрок нажал правой кнопкой мыши на игровое поле. Алгоритм проводит луч из камеры через курсор и определяет точку пересечения луча с игровым полем. Техника движется к полученной точке.

Техническое задание по системе строительства:

* Создание файла для игровых зданий в котором должны записываться размеры строения.
* Создать алгоритм строительства, который хранит размеры области, в которой можно строить. Так же проверять выбрал ли игрок какое-либо здание, если выбрал – провести луч из камеры через курсор мыши и определить точку пересечения луча с игровым полем, после округлить полученные координаты точки до 100 условных единиц измерения Unity и установить в полученные координаты выбранное игроком здание.
* Проверить выходит ли выбранное строение за границы зоны строительства, если выходит, то окрасить выбранное здание в красный и передаёт в переменную “возможно” ложь.
* Проверить занято ли место другим зданием, проверяя каждую координату, занимаемую зданием с записанными координатами в листе занятых координат. Если совпадений нет, то считается, что место не занято и здание перекрашивается в зелёный и передаёт в переменную “возможно” правду, если совпадения есть, то здание перекрашивается в красный передаёт в переменную “возможно” ложь.
* Если игрок нажал кнопку Е, то повернуть строение на девяносто градусов против часовой стрелки.
* Если игрок нажал правую клавишу мышки, переменная “возможно” является правдой и количество внутри игровой валюты у игрока больше чем цена здания, то у игрока из количества внутри игровой валюты вычитается цена здания, все занимаемые зданием координаты записываются в лист занятых координат, здание красится в исходный цвет и становится осязаемым.

2.2 Работа кода

Код в файле “Control Unit” (смотрите приложение) отвечает за управление игровой техникой. Первым делом код проверяет нажатие левой кнопки мыши и клавиши “Shift".

После чего проводится луч из камеры через курсор мышки и проверяется объект, с которым столкнулся луч. Если объект имеет третий слой, тэг “Unit” и переменная его родительского элемента в файле “storega” равна значению “player”, то в переменную “selected” его родительского элемента в файле “ControlUnit” передаётся значение правда, иначе переменная “selected” во всех файлах “ControlUnit” получает значение ложь.

Дальше идёт проверка на нажатие только левой кнопки мыши. Если условие верно, то проводится луч из камеры через курсор мыши и проверяется объект, с которым столкнулся луч. Если объект имеет третий слой, тэг “Unit” и переменная его родительского элемента в файле “storega” равна значению “player”, то переменная “selected” во всех файлах “ControlUnit” получает значение ложь, далее переменная “selected” в его файле “ControlUnit” получает значение правда. Если объект имеет третий слой и имеет тег “earth”, то иначе переменная “selected” во всех файлах “ControlUnit” получает значение ложь. Если ни одно из условий не выполнено, то переменная “selected” во всех файлах “ControlUnit” получает значение ложь.

Дальше код проверяет нажатие на правую клавишу мыши и переменную “selected”. Если условие вено, то проводится луч из камеры через курсор мыши, координаты точки пересечения луча с каким-либо объектом передают в качестве переменной для вызываемого метода “SetPointNav”.

Метод “SetPointNav” принимает в себя переменную типа “Vector3”, при вызове обращается к “NavMechAgent” и задаёт цель передвижение на полученных координатах.

Код в файлах “artTarget” и “tankTarget” одинаковы за исключением строчки с переменной t, это связано с тем, что в файлах “Turrel” и “TurrelArt” имели разные классы и по которым к ним можно обращаться. Сам код проверяет какие объекты вошли в прозрачную сферу вокруг техник.

public void OnTriggerEnter(Collider other)

{

if (other.gameObject.GetComponentInParent<storega>() != null && other.gameObject.GetComponentInParent<storega>().team != gameObject.GetComponent<storega>().team)

{

AllTar.Add( other.gameObject.transform.parent.transform);

}

}

И собирает в список целей все объекты в файле “Storega” параметр команда такого же параметра в технике, к которой файлы “artTarget” и “tankTarget” прикреплены. Проводится проверка на выход целей из прозрачной сферы.

public void OnTriggerExit(Collider other)

{

if (other.gameObject.GetComponentInParent<storega>() != null && other.gameObject.GetComponentInParent<storega>().team != gameObject.GetComponent<storega>().team)

{

AllTar.Remove( other.gameObject.transform.parent.transform);

}

}

Первый элемент получившегося списка передаётся в файл файлах “Turrel” или “TurrelArt” в качестве цели.

public TurrelArt t;

public List<Transform> AllTar = new List<Transform>();

public ControlUnit s;

void Update()

{

if (AllTar.Count > 0)

{

t.target = AllTar[0];

}

else

{

t.target = null;

}

}

Код файла “Build” создаёт сетку вокруг игрового здания, которая создаётся на основании полученных значений ширены и длины из параметра “Size”. Созданная сетка видна только в Unity и нужна для визуального отображения занимаемой зданием площади

private void OnDrawGizmosSelected()

{

//Отрисовка сетки здания

for(int x = 0; x < Size.x; x++)

{

for(int y = 0; y < Size.y; y++)

{

Gizmos.color=new Color(0f,1f,0f,0.3f); //Цвет

Gizmos.DrawCube((transform.position)+ new Vector3(x,0,y),new Vector3(1,.1f,1)); //Отрисовка кубов

}

}

}

Также два метода: “SetTransparent”, перекрашивающий здание в зелёный или красный, путём обращения к переменной цвет в материале и установления значения на нужный.

public void SetTransparent(bool avaible)

{

if (avaible)

{

Mainrenderer.material.color=Color.green;

}

else

{

Mainrenderer.material.color=Color.red;

}

}

“SetNormal” перекрашивающий здание в исходный цвет, путём обращения к переменной цвет в материале и установления значения на белый.

public void SetNormal()

{

Mainrenderer.material.color = Color.white;

}

Оба метода нужны для файла “Building”.

Код файлы “Building” отвечает логику строительства. Первым делом код передаёт в лист значения предельных точек зоны строительства.

grid = new build[GridSize.x, GridSize.y];

находит камеру игрока и игрока.

mainCamera = Camera.main;

player = FindObjectOfType<playerScript>();

Так же игровая сцена проверяется на наличие уже установленных зданий для того, чтобы избежать проблем с проверкой на занятость территории. Если установленные со старта здания имеются, то их координаты округляются до одной условной клетки строительства и используются для внесения в лист всех занимаемых зданием координат.

build[] startedBuilds = FindObjectsOfType<build>();

if (startedBuilds.Length > 0)

{

for (int q = 0; q < startedBuilds.Length; q++)

{

int placex = Mathf.RoundToInt(startedBuilds[q].transform.position.x / 100) \* 100;

int placey = Mathf.RoundToInt(startedBuilds[q].transform.position.z / 100) \* 100;

for (int i = 0; i < startedBuilds[q].Size.x; i++)

{

for (int y = 0; y < startedBuilds[q].Size.y; y++)

{

grid[placex + i, placey+y] = startedBuilds[q];

}

}

}

}

Далее идёт метод для кнопок интерфейса, который устанавливает новое здание для строительства, проверяя наличие уже выбранного для строительства здания, но неустановленного и удаляет его. Так же выбранное здание становится не осязаемым для техники для нормального функционирования их системы передвижения.

public void StartPlacingBuilding(build buildPref)

{

if(flyingBuilding != null)

{

Destroy(flyingBuilding.gameObject);

}

flyingBuilding = Instantiate(buildPref);

flyingBuilding.gameObject.GetComponent<NavMeshObstacle>().enabled = false;

}

Далее код проверяет наличие выбранного для строительства здания, если данное здание существует, то код создаёт плоскость строительства и пускает луч из камеры через мышку. Проверяет наличие пересечений луча с плоскостью.

private void Update()

{

if (flyingBuilding != null)

{

var groundPlane = new Plane(Vector3.up, Vector3.zero);

Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);

При наличии которых получает точку пересечения, координаты которой округляются до значений одного шага сетки и вводится булевая переменная отображающая возможность установить здание на поверхность игровой сцены.

if(groundPlane.Raycast(ray, out float position))

{

Vector3 worldPosition = ray.GetPoint(position);

int x = Mathf.RoundToInt(worldPosition.x/100)\*100;

int y = Mathf.RoundToInt(worldPosition.z/100)\*100;

bool available = true;

Получившиеся координаты проверяются с крайними точками зоны строительства. Если координаты находятся вне зоны строительства, то булевая переменная принимает значение ложь.

if (x < 0 || x > GridSize.x - flyingBuilding.Size.x) available = false;

if ( y < 0 || y > GridSize.y - flyingBuilding.Size.y) available = false;

Далее идёт проверка на занятость территории и близости других строений игрока.

if(available && IsPlaceTaker(x, y)) available = false;

if(flyingBuilding.GetComponentInChildren<fatoryTriger>().around == false) available = false;

Метод получает точку установки здания и проверяет наличие данных с такими координатами в массиве. Проверяются все координаты, которые занимает здание, которые находятся с помощью размеров здания. Если все занимаемые координаты отсутствуют в массиве, то булевая переменная принимает значение правда.

private bool IsPlaceTaker(int placeX, int placeY)

{

for (int i = 0; i < flyingBuilding.Size.x; i++)

{

for (int y = 0; y < flyingBuilding.Size.y; y++)

{

if(grid[placeX + placeX, placeY] != null)

{

return true;

}

}

}

return false;

}

Далее код перемещает выбранное здание на свободные координаты и проверяет булевую переменную и нажатие левой кнопки мыши, если условия верны, то выполняется метод установки здания.

flyingBuilding.transform.position = new Vector3(x, 0, y);

flyingBuilding.SetTransparent(available);

if (available && Input.GetMouseButtonDown(0))

{

PlaceFlyingBuilding(x, y);

}

}

}

}

Данный метод сперва проверяет наличие необходимой для постройки суммы внутри игровой валюты у игрока. Если у игрока есть нужная сумма внутри игровой валюты, то вычитает её из средств игрока.

private void PlaceFlyingBuilding(int placeX, int placeY)

{

if (flyingBuilding.gameObject.GetComponent<build>().costBuild <= player.cash)

{

player.cash -= flyingBuilding.gameObject.GetComponent<build>().costBuild;

Далее метод проходит по всем координатам условной сетки строительства и вносит их в массив вместе с ссылкой на здание. Далее здание становится осязаемым для техники и в файле “storega” вводится название команды, к которой должно принадлежать здание. Переменная хранящая выбранное здание приравнивается значению “null”.

for (int i = 0; i < flyingBuilding.Size.x; i++)

{

for (int y = 0; y < flyingBuilding.Size.y; y++)

{

grid[placeX + placeX, placeY] = flyingBuilding;

}

}

flyingBuilding.SetNormal();

flyingBuilding.gameObject.GetComponent<NavMeshObstacle>().enabled = true;

flyingBuilding.gameObject.GetComponent<storega>().team = "player";

flyingBuilding = null;

Код файла “factoryScript” отвечает за логику производства заводами игровой техники и изменение игрового интерфейса для создания возможности выбора производства игровой техники игроком.

Первым делом код чистит лист очереди производства игровой техники. Потом находит игровой интерфейс, графический raycast, систему событий и игрока.

void Start()

{

test.Clear();

canvas = FindObjectOfType<Canvas>();

//получение графического raycast из конваса

raycaster = canvas.GetComponent<GraphicRaycaster>();

//получение событий системы из сцены

m\_EventSystem = GetComponent<EventSystem>();

//нахождение игрока

player = FindObjectOfType<playerScript>();

}

// Update is called once per frame

Далее код проверяет выбрал ли игрок какой-нибудь завод и нажал левую кнопку мыши. Если условие верно, то происходит проверка нажатие игроком на игровой интерфейс, осуществляемое с помощью графического raycast и систему событий. Если проверка выявляет, что игрок нажал на игровой интерфейс, то булевая переменная “clickInterfase” становится правдой.

void Update()

{

if (Input.GetMouseButtonDown(0) && gameObject.GetComponent<build>().active == true)

{

clickInterfase = false; // Графический raycast проверяющий обращается ли игрок к интерфейсу

m\_PointerEventData = new PointerEventData(m\_EventSystem);

m\_PointerEventData.position = Input.mousePosition;

List<RaycastResult> results = new List<RaycastResult>();

raycaster.Raycast(m\_PointerEventData, results);

foreach (RaycastResult result in results)

{

clickInterfase = true;

}

Дальше алгоритм пускает луч из камеры через мышку и проверяет пересечение этого луча с каким-либо объектом, если этот объект имеет третий или пятый слой. Если луч сталкивается с таким объектом, то происходит проверка, равен ли тэг этого объекта значению “factory”, и булевой переменной “clickInterfase”, если условия верны, то происходит обращение к переменной “selected” в файле “factoryScript” и она становится равна правде. Если нет, то во всех файлах “factoryScript” переменная “selected” становится равна лжи.

Ray movePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(movePosition, out var hitInfo, Mathf.Infinity, (1 << 3) | (1 << 5)))

{

GameObject q = hitInfo.collider.gameObject;

if (q.CompareTag("factory") || clickInterfase)

{

q.GetComponent<factoryScript>().selected = true;

}

else

{

selected = false;

}

Далее происходит изменение игрового интерфейса, а именно текущее меню выбора строительства зданий становится не активным и не отображается, а меню выбора производства техники становится активным и отображается.

if((q.CompareTag("factory")||clickInterfase)&&q.GetComponent<build>()!=null&&q.GetComponent<build>().active&&q.GetComponent<build>()!=null)

{

canvas.transform.GetChild(2).gameObject.SetActive(false);

canvas.transform.GetChild(3).gameObject.SetActive(true);

}

else

{

canvas.transform.GetChild(2).gameObject.SetActive(true);

canvas.transform.GetChild(3).gameObject.SetActive(false);

}

Далее идёт подсчёт стоимости всей заказанной игроком техники для более удобного сравнения с количеством внутри игровой валюты.

int qw = 0;

for(int i =0; i < test.Count; i++)

{

qw += test[i].GetComponent<storega>().cost;

}

После идёт проверка на переменной “selected”, листа, хранящего выбранную для производства технику, булевую переменную “wait”, активности завода и количества внутри игровой валюты у игрока. Если условия выполнены, то вызывается перегрузка метода “SpawnUnit”

if (selected && test.Count != 0 && wait == false && gameObject.GetComponent<build>().active == true && player.cash >= qw)

{

SpawnUnit(player);

}

Этот метод предназначен для игроков, производит техники с интервалом 2 секунды. Перед созданием игровой техники происходит проверка на нужную сумму внутри игровой валюты на счету игрока, если у игрока хватает её количества, то техника создаётся в точке создания и поворачивается как эта точка. Если у игрока не хватает средств, то метод завершается.

public async Task SpawnUnit(playerScript player)

{

wait = true;

for(int i = 0; i < test.Count; i++)

{

UnityEngine.Debug.Log(wait + " | wait");

await Task.Delay(TimeSpan.FromSeconds(2f));

if ((player.cash -= test[i].GetComponent<storega>().cost) >= 0)

{

player.cash -= test[i].GetComponent<storega>().cost;

b = Instantiate(test[i], pointSpawn.transform.position, pointSpawn.transform.rotation);

b.GetComponent<storega>().team = gameObject.GetComponent<storega>().team;

test.Remove(test[i]);

}

UnityEngine.Debug.Log(wait + " | wait qqq");

}

wait = false;

}

Далее идёт проверка для заводов искусственного оппонента, проверяется команда завода и длинна листа, хранящего выбранную игровую технику. Если условия верны то вызывается перегрузка метода “SpawnUnit”.

if (gameObject.GetComponent<storega>().team == "enemy" && test.Count >= 0 && wait == false)

{

SpawnUnit();

}

}

Данная перегрузка предназначена для использования искусственным оппонентом, ввиду чего нет проверки количества внутри игровой волюты.

public async Task SpawnUnit()

{

wait = true;

for (int i = 0; i<test.Count; i++)

{

await Task.Delay(TimeSpan.FromSeconds(10f));

b = Instantiate(test[i], pointSpawn.transform.position, pointSpawn.transform.rotation);

b.GetComponent<storega>().team = gameObject.GetComponent<storega>().team;

test.Remove(test[i]);

}

wait = false;

}

Код файла “SpawnBut” прикрепляется к кнопка игрового интерфейса отвечающих за выбор производства заводов. Сам код ищет все выбранные заводы, после чего передаёт в эти обращается к их переменной “test” в файле “factoryScript”. В переменную “test” добавляется переменная “unit”. Переменная “unit” вводится в качестве константы для каждой кнопки.

public void SpawnBut(GameObject unit)

{

var arr = FindObjectsOfType(typeof(factoryScript));

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

{

if (arr[i].GetComponent<factoryScript>().selected){

arr[i].GetComponent<factoryScript>().test.Add(unit);

}

}

}

Код файла “playerScript” отвечает за логику перемещения, вращения камеры игрока и выбора техники, так же хранит количество внутри игровой валюты у игрока.

public class playerScript : MonoBehaviour

{

private Rigidbody \_rb;

public float speed = 10f;

public int cash = 1000;

public selector sel;

private CharacterController controller;

public GameObject pointRotate;

public Text cashIm;

public Canvas interfaces;

public Vector3 pos1;

public Vector3 pos2;

public GameObject cube;

public RectTransform graph;

public Vector2 pos;

Первым делом код находит контролер персонажей, необходимый для реализации передвижение камеры игрока.

void Start()

{

controller = gameObject.GetComponent<CharacterController>();

}

void Update()

{

Затем код проверяет нажатие левой кнопки мыши, если игрок нажал на неё, то проводится луч из камеры через курсор мыши, координаты точки пересечения луча с каким-либо объектом на игровой сцене записывают в переменную “pos1”.

//Выделение техники или строений с помощью рамки.

if (Input.GetMouseButtonDown(0))

{

Ray movePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(movePosition, out var hitInfo))

{

pos1 = hitInfo.point;

}

}

Далее код зажал ли игрок левую кнопку мыши, если игрок её зажал, то проводится луч из камеры через курсор мыши, координаты точки пересечения луча с каким-либо объектом на игровой сцене записывают в переменную “pos2”, после чего происходит нахождение координат середины кратчайшего пути между двумя координатами при помощи формулы (x1 + x2)/2 = x и (y1 +y2)/2 =y, найденные координаты записываются в переменную “s”.

if (Input.GetMouseButton(0))

{

Ray movePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(movePosition, out var hitInfo))

{

pos2 = hitInfo.point;

Vector3 s = new Vector3((pos2.x + pos1.x)/2, 10, (pos2.z + pos1.z)/2); //нахождение середины между первоначальным нажатием мыши и текущим её положением

Далее код ставит куб в координаты “s” и задаёт размеры путём, вычитание координат второй точки из первой. Куб является объектом с помощью которого осуществляется выделение.

cube.transform.position = new Vector3(s.x , pos1.y,s.z );

cube.transform.localScale = new Vector3(pos2.x-pos1.x, 100, pos2.z-pos1.z);

}

}

Если игрок отпустит левую кнопку мыши, то переменные “pos1” и “pos2” обнуляются, происходит вызов метода “OnSelect” в файле “selector”, а куб перемещается на нулевые координаты и изменяет размеры по всем осям на ноль.

if (Input.GetMouseButtonUp(0))

{

sel.OnSelect();

pos2 = new Vector3(0, 0, 0);

pos1 = new Vector3(0, 0, 0);

cube.transform.position = pos1;

cube.transform.localScale = pos1;

}

//преобразовывает локальный вектор в глобальный получение вектора направления, получаемого от нажатия клавишь по осям x и z

Далее происходи получение и преобразование локального вектора в глобальный с помощью встроенной функции TransformDirection. Докальный вектор получается с помощью проверки нажатия клавиш категории "Horizontal" и "Vertical". Дальше идет обращение к функции контролера персонажей "Move" в которую передаётся произведение полученного глобального вектора, интервала между кадрами и скорости передвижения.

Vector3 move = transform.TransformDirection(new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical")));

controller.Move(move \* Time.deltaTime \* speed); // обращение в контроллеру персонажей для передвижения камеры

Дальше идут условия, проверяющие нажатие игрока на кнопки "Q", "E", "R", "F". Если какое – то из условий верно, то вызывается метод "Rotaite" с числовой переменной. Если переменная равна 0 или 1, то происходит поворот по оси x на ноль целых двадцать пять сотых и минус ноль целых двадцать пять сотых соответственно. Если переменная равна 2 или 3, то происходит поворот по оси y на ноль целых двадцать пять сотых и минус ноль целых двадцать пять сотых соответственно.

if (Input.GetKey(KeyCode.Q)){Rotaite(0);}

if (Input.GetKey(KeyCode.E)){Rotaite(1);}

if (Input.GetKey(KeyCode.R)){Rotaite(2);}

if (Input.GetKey(KeyCode.F)){Rotaite(3);}

Далее код выводит на игровой интерфейс информацию о количестве внутри игровой валюты.

cashIm.text = cash + "$";

}

public void Rotaite(int lR)

{

if (lR == 0) {pointRotate.transform.Rotate(((float)(0.25)), 0, 0);}

if (lR == 1) {pointRotate.transform.Rotate(((float)(-0.25)), 0, 0);}

if (lR == 2)

{transform.Rotate(0, ((float)(-0.25)),0, Space.World);}

if (lR == 3)

{transform.Rotate(0, ((float)(0.25)),0, Space.World);}

}

}

Файла “Storega" находится в каждой игровой технике и здании, хранит такие переменные как цену – “cost”, здоровье – “hp”, команда – “team”.

public class storega : MonoBehaviour

{

public int cost;

public int hp;

public string team;

private int oldHp;

Первым делом код записывает изначальное количество здоровья для дальнейшего сравнения с текущим, это необходимо для изменения объекта, выполняющего роль визуального отображения количества здоровья.

void Start()

{

oldHp = gameObject.GetComponent<storega>().hp;

}

if(oldHp != hp)

{

oldHp = hp;

gameObject.GetComponent<healbar>().SetHealbar();

}

Если количество здоровья меньше ровно нулю, то игровая техника или здание удаляется с игровой сцены.

if(hp <= 0)

{

Destroy(gameObject);

}

Код файла “TurrelArt” отвечает за наведение башни и орудия, создание снаряда и придание ему такой скорости, с которой снаряд долетит до цели. Первым дело код проверяет наличие целей. Если они есть то, происходит вызов метода “Vector2d” с переданным значением “y”.

void Update()

{

if(target != null)

{

Vector3 navAtTargetXZ = Vector2d("y");

Метод “Vector2d” получает направление на цель путём вычитания позиции цели из позиции башни техники, дальше код проверяет переменную “axis”, и в зависимости от того какую букву обозначения координат она хранит, ту плоскость координат и выкидывает из полученного направления на цель, путём умножения значений этой плоскости на ноль. После чего возвращает получившиеся координаты.

public Vector3 Vector2d(string axis)

{

Vector3 nav = target.position - transform.position;

switch (axis)

{

case "x":

nav.x = 0;

break;

case "y":

nav.y = 0;

break;

case "z":

nav.z = 0;

break;

}

return nav;

}

Дальше код вычисляет на сколько в квартернионах нужно повернуть башню, чтобы она смотрела в направлении цели, после чего совершает поворот башни со скоростью равной произведению переменной “SpeedRotation” и интервалу между кадрами.

Quaternion targetRotation = Quaternion.LookRotation(navAtTargetXZ);

transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotation, SpeedRotation \* Time.deltaTime);

Далее орудие техники поворачивается на 45 градусов и проверяется булевая переменная “isShoot”, если она равна лжи, то вызывается метод “Shoot”.

PointRotateGun.localEulerAngles = new Vector3(-AngleInDegrees, 0f, 0f);

if (isShoot == false)

{

Shoot();

}

}

}

Метод “Shoot” первым делом задаёт значение переменной “isShoot” правда и ожидает время перезарядки стрельбы. Далее происходит расчёт скорости снаряда для поражения цели по формуле v= √(gx2)/(2(y-tgα\*x) \* cos2α, где v – скорость, g – ускорение свободно падения, у и x это дистанция до цели на осях y и x соответственно, α – это угол под которым снаряд вылетает из ствола. Результат записывается в переменную “v”.

public async Task Shoot()

{

isShoot = true;

await Task.Delay(TimeSpan.FromSeconds(shootSpeed)); // Балистика изменяя скорость снаряда

Vector3 nav = target.position - shootEl.position;

Vector3 navXZ = new Vector3(nav.x, 0f, nav.z);

float x = navXZ.magnitude;

float y = nav.y;

float AngleInRadians = AngleInDegrees \* Mathf.PI / 180;

float v2 = (g \* x \* x) / (2 \* (y - Mathf.Tan(AngleInRadians) \* x) \* Mathf.Pow(Mathf.Cos(AngleInRadians), 2));

float v = Mathf.Sqrt(Mathf.Abs(v2));

Дальше создаётся снаряд и ему придаётся скорость в виде произведения глобального вектора локальной по оси z и переменной “v”. Переменная “isShoot” получает значение ложь.

GameObject b = Instantiate(bullet, shootEl.position, shootEl.rotation);

b.GetComponent<Rigidbody>().velocity = shootEl.transform.forward \* v;

isShoot = false;

Код файла “Turrel” отвечает за наведение башни и орудия на цель и отправки снаряда в цель. Если они есть то, происходит вызов метода “Vector2d” с переданным значением “y”.

void Update()

{//Наведение турели на цель

if(target != null)

{

Vector3 navAtTargetXZ = Vector2d("y");

Метод “Vector2d” получает направление на цель путём вычитания позиции цели из позиции башни техники, дальше код проверяет переменную “axis”, и в зависимости от того какую букву обозначения координат она хранит, ту плоскость координат и выкидывает из полученного направления на цель, путём умножения значений этой плоскости на ноль. После чего возвращает получившиеся координаты.

public Vector3 Vector2d(string axis)

{

Vector3 nav = target.position - transform.position;

switch (axis)

{

case "x":

nav.x = 0;

break;

case "y":

nav.y = 0;

break;

case "z":

nav.z = 0;

break;

}

return nav;

}

Дальше код вычисляет на сколько в квартернионах нужно повернуть башня, чтобы она смотрела в направлении цели, после чего совершает поворот башни со скоростью равной произведению переменной “SpeedRotation” и интервалу между кадрами.

Quaternion targetRotation = Quaternion.LookRotation(navAtTargetXZ, Vector3.up);

transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotation, SpeedRotation \* Time.deltaTime); targetRotation, SpeedRotation \* Time.deltaTime);

Далее код получает вектор на цель путём вычитания координат цели из координат башни. После чего вычисляет на сколько в квартернионах нужно повернуть орудие. После чего совершает поворот орудия со скоростью равной произведению переменной “SpeedRotation” и интервалу между кадрами. Далее проводится луч от орудия в направлении локальной оси z. Если луч пересекается с объектов с третьим слоем, является частью цели и переменная “isShoot” имеет значение ложь, то вызывается метод “Shoot”.

Vector3 navAtTargetYZ = target.position - transform.position;

Quaternion t = Quaternion.LookRotation(navAtTargetYZ);

PointRotateGun.rotation = Quaternion.RotateTowards(PointRotateGun.rotation, t, 10 \* Time.deltaTime);

Ray traicShell = new Ray(shootEl.position, shootEl.forward);

RaycastHit hit;

if (Physics.Raycast(traicShell, out hit, Mathf.Infinity, (1 << 3)))

{

if (hit.collider.gameObject.transform.parent.gameObject == target.gameObject)

{

if(isShoot == false)

{

Shoot();

}

}

}

Метод “Shoot” первым делом задаёт значение переменной “isShoot” правдой. После код ожидает перезарядку орудия равную переменной “shootSpeed” в секундах. Находится глобальный вектор локальной оси z и создаётся снаряд. Снаряду задаётся скорость, равная произведению глобального вектора локальной оси z и скорости снаряда.

public async Task Shoot()

{

isShoot = true;

await Task.Delay(TimeSpan.FromSeconds(shootSpeed));

Vector3 n = navBulletPos.position - shootEl.position;

GameObject b = GameObject.Instantiate(bullet, shootEl.position, transform.rotation) as GameObject;

b.GetComponent<Rigidbody>().velocity = n \* 1;

isShoot = false;

}

Код в файле “Enemy” отвечает за логику искусственного оппонентов, а именно: создание игровой техники с помощью заводов и её отправки на место нахождения зданий и техники игрока. Он содержит: переменную “unit” хранящую массив игровых объектов, переменную “allfactory” хранящую лист игровых объектов, переменную “ListPointNAv” хранящую лист игровых объектов, переменную “ListAllUnitEnemy” хранящую лист игровых объектов, переменную “sovpadenie” хранящую булевую переменную, переменную “Problem” хранящую массив файлов “ControlUnit”, переменную “numberNav” хранящую целочисленное значение, переменную “q” хранящие целочисленное значение.

public class EnemyAi : MonoBehaviour

{

public GameObject[] unit;

public List<GameObject> allfactory = new List<GameObject>();

public List<GameObject> ListPointNav = new List<GameObject>();

public List<GameObject> ListAllUnitEnemy = new List<GameObject>();

public bool sovpadenie = false;

public ControlUnit[] Problem;

public int numberNav = 0;

public int q = 0;

Первым дело создаётся переменная “AllUnit” в значение которой записывается вся игровая техника с помощью функции “ FindObjectsOfType”, который выводит все игровые объекты с заданным файлом, так же задаётся значение переменной “Problem”. Далее создаётся “allFactorio” в значения которой записывается вся игровая техника с помощью функции “ FindObjectsOfType”. Далее запускается цикл, если значение переменной “i”, изначально равное нулю буде больше количества хранящихся в переменной “allFactorio” игровых объектов при этом после каждого интервала значение переменной “i” увеличивается на единицу. В теле цикла происходит проверка значения в переменной “team”, которая находится файле “storega”, которой находится внутри игрового объекта под индексом равным значению переменной “i”. Если значение в переменной “team” равно значению “enemy”, то в лист, хранящийся переменной “allfactory” добавляется игровой объект из переменной “allfactoio” под индексом равным значению переменной “i”.

void Start()

{

var AllUnit = FindObjectsOfType<ControlUnit>();

Problem = FindObjectsOfType<ControlUnit>();

var allfactorio = FindObjectsOfType<factoryScript>();

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

{

if (allfactorio[i].GetComponent<storega>().team == "enemy")

{

allfactory.Add(allfactorio[i].gameObject);

}

}

Далее запускается цикл, если значение переменной “i”, изначально равное нулю будет больше количества хранящихся в переменной “AllUnit” игровых объектов при этом после каждого интервала значение переменной “i” увеличивается на единицу. В теле цикла происходит проверка значения в переменной “team”, которая находится файле “storega”, которой находится внутри игрового объекта под индексом равным значению переменной “i”. Если переменная “team” равна значению “enemy”, то игровой объект, хранящийся в внутри переменной “AllUnit” под индексом равным значению переменной “i”. После цикла значение переменной “AllUnit” становится равно нулю.

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

{

if (AllUnit[i].GetComponent<storega>().team == "enemy")

{

ListAllUnitEnemy.Add(AllUnit[i].gameObject);

}

}

AllUnit = null;

Дальше код переменная “AllUnit” в значение которой записывается вся игровая техника с помощью функции “ FindObjectsOfType”. Потом происходит проверка значения переменной “AllUnit” и значения переменной “Problem”, если они не равны, то запускается метод “NewUnitEnemy” с переданной переменной “AllUnit”, после переменная “Problem” принимает в значение значение переменной “AllUnit”.

void Update()

{

var AllUnit = FindObjectsOfType<ControlUnit>();

if (AllUnit != Problem)

{

NewUnitEnemy(AllUnit);

Problem = AllUnit;

}

Метод “NewUnitEnemy” первым делом запускается цикл, если значение переменной “i”, изначально равное нулю будет больше количества хранящихся в переменной “AllUnit” игровых объектов при этом после каждого интервала значение переменной “i” увеличивается на единицу. В теле цикла значение переменной “numberNav” увеличивается на единицу и проводится проверка переменной “numberNav”, если значение этой переменной больше числа двух, то значение переменной устанавливается на ноль. Далее происходит проверка значения в переменной “team”, которая находится файле “storega”, которой находится внутри игрового объекта под индексом равным значению переменной “i” и вызов метода “IndexOf” у переменной “ListAllUnitEnemy” с передачей значения переменной “AllUnit” под индексом равным значению переменной “i”. Если значение переменной “team” равно значению “enemy” и возвращённое методом “IndexOf” меньше нуля, то в значение переменной “ListAllUnit” добавляется игровой объект в переменной “AllUnit” под индексом равным значению “i”. Далее идёт проверка значения переменой “navtarget”, который находится внутри файла “ControlUnit”, который находится внутри игрового объекта, который был получен с помощью метода “Last” применённой к переменной “ListAllUnitEnemy”, если значения переменой “navtarget” равно значению вектора с значениями ноль по всем осям.

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

{

numberNav += 1;

if (numberNav > 2)

{

numberNav = 0;

}

if (AllUnit[i].gameObject.GetComponent<storega>().team == "enemy" && ListAllUnitEnemy.IndexOf(AllUnit[i].gameObject) < 0)

{

ListAllUnitEnemy.Add(AllUnit[i].gameObject);

if (ListAllUnitEnemy.Last().GetComponent <ControlUnit>().navtarger == new Vector3(0, 0, 0))

{

То происходит вызов метода “SetPointNav”, который находится внутри файла “ControlUnit”, который находится внутри игрового объекта, который был получен с помощью метода “Last” применённой к переменной “ListAllUnitEnemy”, так же в метод передаётся значение из переменной “position”, который находится в “transform”, который находится в игровом объекте, который находится в переменной “ListPointNav” под индексом равным значению переменной “numberNav”.

ListAllUnitEnemy.Last().GetComponent<ControlUnit>().SetPointNav(ListPointNav[numberNav].transform.position);

}

}

}

Далее запускается цикл, если значение переменной “i”, изначально равное нулю будет больше количества хранящихся в переменной “ListAllUnitEnemy” игровых объектов при этом после каждого интервала значение переменной “i” увеличивается на единицу. В теле цикла проверка значения переменной “x”, который находится в переменной “navtarger”, которая находится в файле “ControlUnit”, который находится в игровом объекте в переменной “ListAllUnitEnemy” под индксом равным значению переменной “i” и значения переменной “z”, который находится в переменной “navtarger”, которая находится в файле “ControlUnit”, который находится в игровом объекте в переменной “ListAllUnitEnemy” под индксом равным значению переменной “i”, если значение переменной “x” равно значению переменной “x”, которая находится в переменной “position”, которая находится в переменной “transform”, которая находится в игровом объекте в переменной “ListAllUnitEnemy” под индексом равным значению переменной “i” и значение переменной “z” равно значению переменной “z”, которая находится в переменной “position”, которая находится в переменной “transform”, которая находится в игровом объекте в переменной “ListAllUnitEnemy” под индексом равным значению переменной “i”, то вызывается “SetPointNav”, который находится в файле “ControlUnit”, который находится в игровом объекте в переменной “ListAllUnitEnemy” под индексом равным значению переменной “i”, в метод передаётся переменная “position”, которая находится в переменной “transform”, которая находится в переменной “ListPointNav” под индексом равным трём.

for (int i = 0; i < ListAllUnitEnemy.Count; i++)

{

if (ListAllUnitEnemy[i].GetComponent<ControlUnit>().navtarger.x == Mathf.Round(ListAllUnitEnemy[i].transform.position.x) && ListAllUnitEnemy[i].GetComponent<ControlUnit>().navtarger.z == Mathf.Round(ListAllUnitEnemy[i].transform.position.z))

{

ListAllUnitEnemy[i].GetComponent<ControlUnit>().SetPointNav(ListPointNav[3].transform.position);

}

}

Далее происходит проверка значения переменной “q”, если оно больше количества хранящихся в переменной “unit” значений, то значение переменной “q” устанавливается нулём. После запускается цикл, если значение переменной “i”, изначально равное нулю будет больше количества хранящихся в переменной “allfactory” игровых объектов при этом после каждого интервала значение переменной “i” увеличивается на единицу. В теле цикла происходит добавление значения переменной “unit” под индексом равным значению переменной “q” в переменную “test”, которая находится в файле “factoryScript”, который находится в игровом объекте в переменной “allfactory” под индексом равным значению переменной “i”. Далее значение переменной “q” увеличивается на единицу, после значение переменной“AllUnit” становится равным значению “null”

Далее переменная “AllUnit” обнуляется.

if (q > unit.Count())

{

q = 0;

}

for (int i = 0; i < allfactory.Count; i++)

{

allfactory[i].GetComponent<factoryScript>().test.Add(unit[q]);

}

q++;

AllUnit = null;

Код в файле “bulletF” отвечает за логику поведения игрового боеприпаса. Файл содержит: переменную “damage” – хранящую числовой показатель наносимого урона, переменную “art” – хранящую булевое значение, определяющее тип снаряда, переменную “col” – хранящую ссылку на игровой объект определяющий зону поражения снаряда, параметр “target” – хранящую лист игровых объектов, которые считаются попавшими в зону поражения снарядами.

public class bulletF : MonoBehaviour

{

public int damage;

public bool art = true;

public GameObject col;

public List<GameObject> target=new List<GameObject>();

Первым делом код проверяет столкновение с каким-либо объектом. Если столкновение было, то проверяется параметр “Слой” у объекта, столкнувшегося с игровым боеприпасом, если этот параметр равен числовому значению три, то проводится проверка параметра “Tag” у того же объекта.

private void OnTriggerEnter(Collider other)

{

if (other != null && other.gameObject.layer == 3)

{

if(other.gameObject.tag != "earth")

{

Если значение этого параметра не равна значению “earth”, то происходит обращение к родительскому элементу объекта, столкнувшегося с игровым боеприпасом. У этого родительского элемента запрашивается переменная “hp” файла “storega”, от значения запрошенной переменной отнимается значения параметра “damage”.

other.gameObject.GetComponentInParent<storega>().hp -= damage;

Далее проверяется значение переменной “col”, если оно не равно значению “null”, то проверяется значение переменной “art” у объекта к которому прикреплён данный файл.

if (col != null)

{

Если значение переменной “art” равно значению “true”, то происходит вызов метода “Splash”

if(gameObject.GetComponent<bulletF>().art)

{

gameObject.GetComponent<bulletF>().Splash();

}

}

если значение переменной “col” равно значению “null”, то происходит удаление с игровой сцены объекта к которому прикреплён описываемый файл.

else

{

Destroy(gameObject);

}

}

Если значение параметра “Tag” у объекта, столкнувшегося с игровым боеприпасом, равен значению “earth”, то происходит удаление с игровой сцены объекта к которому прикреплён описываемый файл.

else

{

Destroy(gameObject);

}

}

}

Метод “Splash” устанавливает значение переменной “art” на значение “false”. Далее происходит обращение к игровому объекту по ссылку в переменной “col” и вызов метода “SetActive” с переданным в этот метод значением “true”. Далее происходит ожидание выполнения метода “Delay” с переданным значением, которое равняется результату выполнения метода “FromSecond”, который находится в объекте “TimeSpan”, с переданным значением равным единице.

public async Task Splash()

{

gameObject.GetComponent<bulletF>().art = false;

col.SetActive(true);

await Task.Delay(TimeSpan.FromSeconds(1));

Далее происходит проверка значения в свойства “Count”, находящегося в значении переменной “target”. Если значение свойства “Count” больше нуля, то происходит запуск цикла. Данный цикл будет повторятся до тех пор, пока целочисленная переменная “i”, изначально равная нулю будет больше значения в свойства “Count”, находящегося в значении переменной “target”, после каждого интервала цикла значение переменной “i” будить увеличиваться на единицу.

if (target.Count > 0)

{

for(int i = 0; i < target.Count; i++)

{

В самом цикле происходит обращение к переменной “hp”, которая лежит в файле “storega”, который прикреплён игровому объекту, лежащему в листе переменной “target” под индексом равным значению переменной “i”. У значения переменной “hp” вычитается число пятнадцать.

target[i].GetComponent<storega>().hp -= 15;

}

}

После завершения цикла происходит обращений к переменной “art” и устанавливается значение “true”. После чего происходит удаление с игровой сцены объекта к которому прикреплён описываемый файл.

gameObject.GetComponent<bulletF>().art = true;

Destroy(gameObject);

}

Код в файле “splashArt” за определение игровых объектов попавших в зону поражения боеприпаса. Код содержит переменную “bull” хранящую ссылку на игровой объект.

public class splashArt : MonoBehaviour

{

public GameObject bull;

Далее происходит проверка на столкновение какого-либо объекта с объектом “Collider”, который обозначает зону поражения боеприпасом.

private void OnTriggerEnter(Collider other)

{

После происходит проверка файла “build” в объекте, который столкнулся с объектом “Collider”, если это запрос этого файла не возвращает значение “null” и значение переменной “active” равно “true” или запрос этого файла возвращает значение “null”, то происходи следующая проверка.

if ((other.gameObject.GetComponent<build>() != null && other.gameObject.GetComponent<build>().active == true) || other.gameObject.GetComponent<build>() == null)

{

Проверяется результат запроса к файлу “storega”, который находится в родительском объекте объекта, который столкнулся с объектом “Collider”.

if (other.gameObject.GetComponentInParent<storega>() != null)

{

Происходит проверка результатов метода “IndexOf” с переданным в качестве переменной родительским элементом объекта, который столкнулся с объектом “Collider”, метод применяется на листе в переменной “target”, которая находится в файле “bulletF”, который находится в игровом объекте переменной “bull”. Если получившееся значение меньше нуля, то в лист в переменной “target” добавляется родительским элементом объекта, который столкнулся с объектом “Collider”.

if (bull.GetComponent<bulletF>().target.IndexOf(other.gameObject.transform.parent.gameObject) < 0)

{

bull.GetComponent<bulletF>().target.Add(other.gameObject.transform.parent.gameObject);

}

}

}

}

}

2.3 Создание моделей

Все трёхмерные модели были созданы в программе “MagicaVoxel”. Данная программа проста в использовании и бесплатна, что идеально подходит для использования в данной работе.

Первым делом была сделана модель танка. Внешний вид модели представлен в рисунке 1.

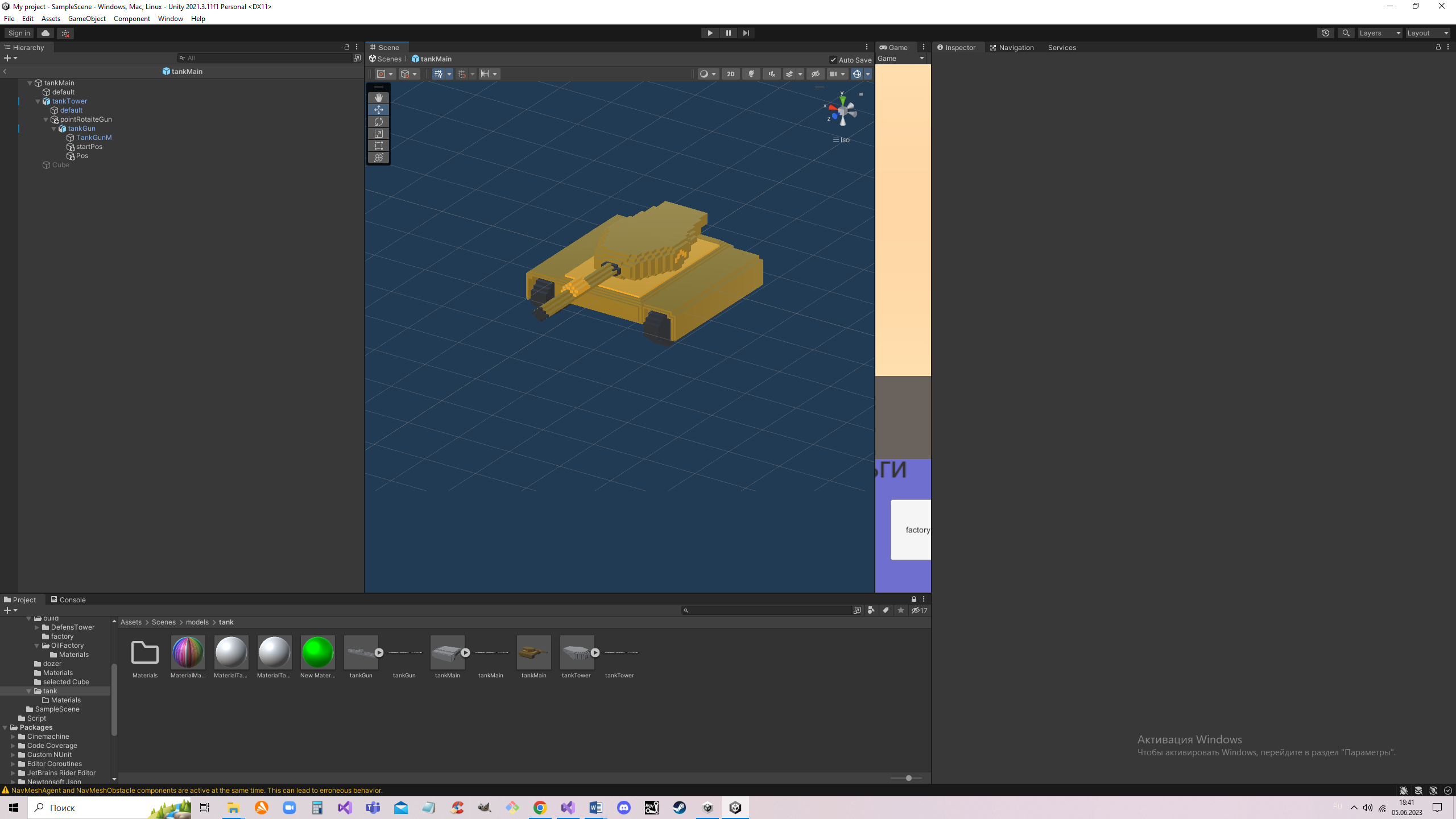


Рисунок 1. Танк

В качестве образа для создания был использован образ танка “M1 Abrams”, внешний вид этой техники представлен в рисунке 2. Причина данного выбора кроется в угловатости внешнего вида техники, что упрощает создание трехмерной модели. Правда это не относится к корпусу – он создан без оглядки на какой-то конкретный образ. Выбор цветов имитирует песчаный окрас танка “M1 Abrams”.



Рисунок 2. танк Abrams

Следующей трёхмерной моделей стал артиллерийский танк (САУ), внешний вид представлен в рисунке 3. Корпус этой модели аналогичен танку, рисунок 1. Орудие техники сделано более крупным для создания представления, что оно стреляет более крупными снарядами. Округлая башня делает технику отличной от другой.

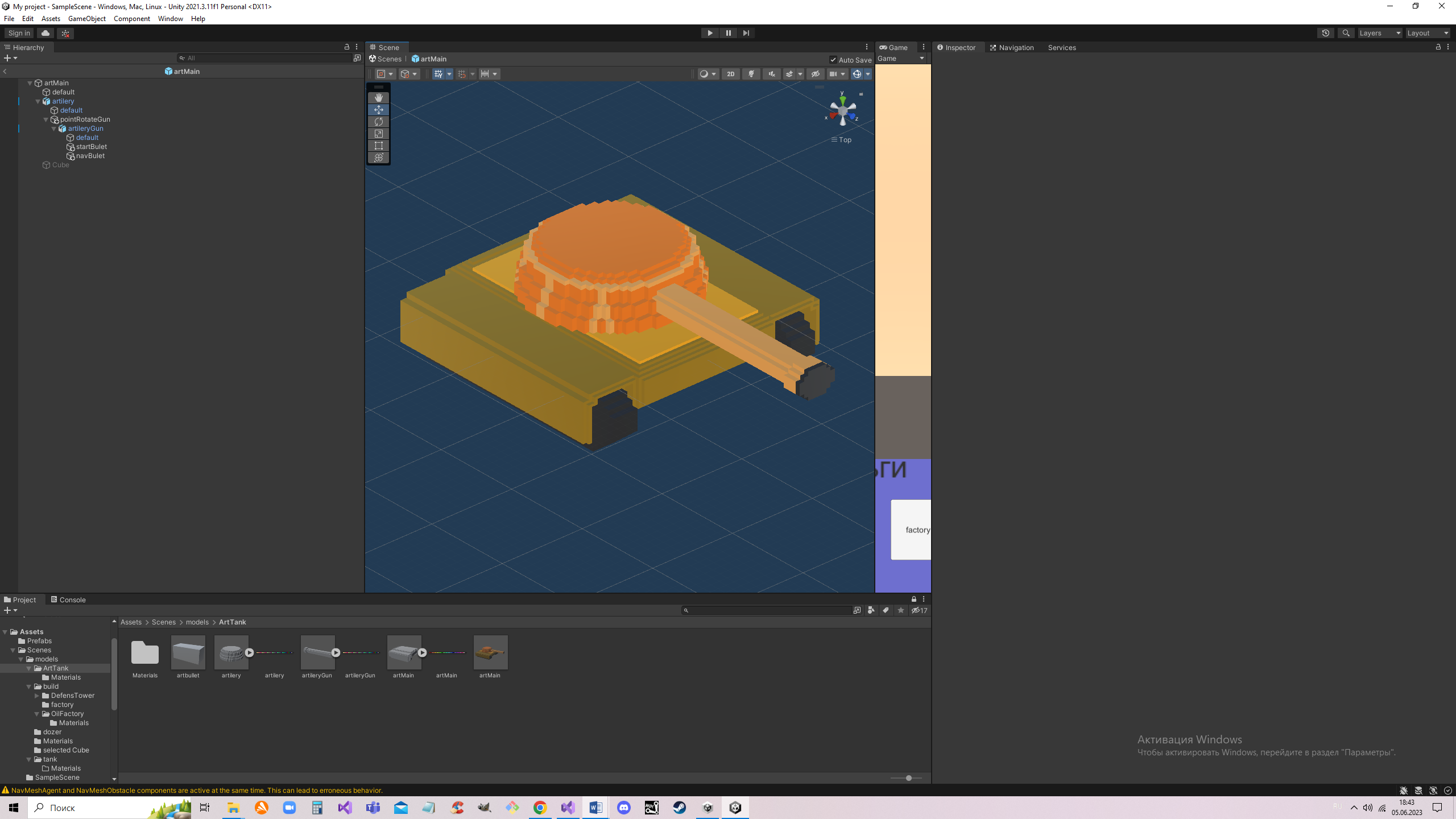


Рисунок 3. Артиллерийский танк (САУ)

Далее была сделана трёхмерная модель инженерной техники, внешний вид которой представлен в рисунке 4. Корпус этой модели аналогичен танку, рисунок 1. На задней части изображён прикреплённый к корпусу контейнер. На переднею часть установлен отвал для расчистки.

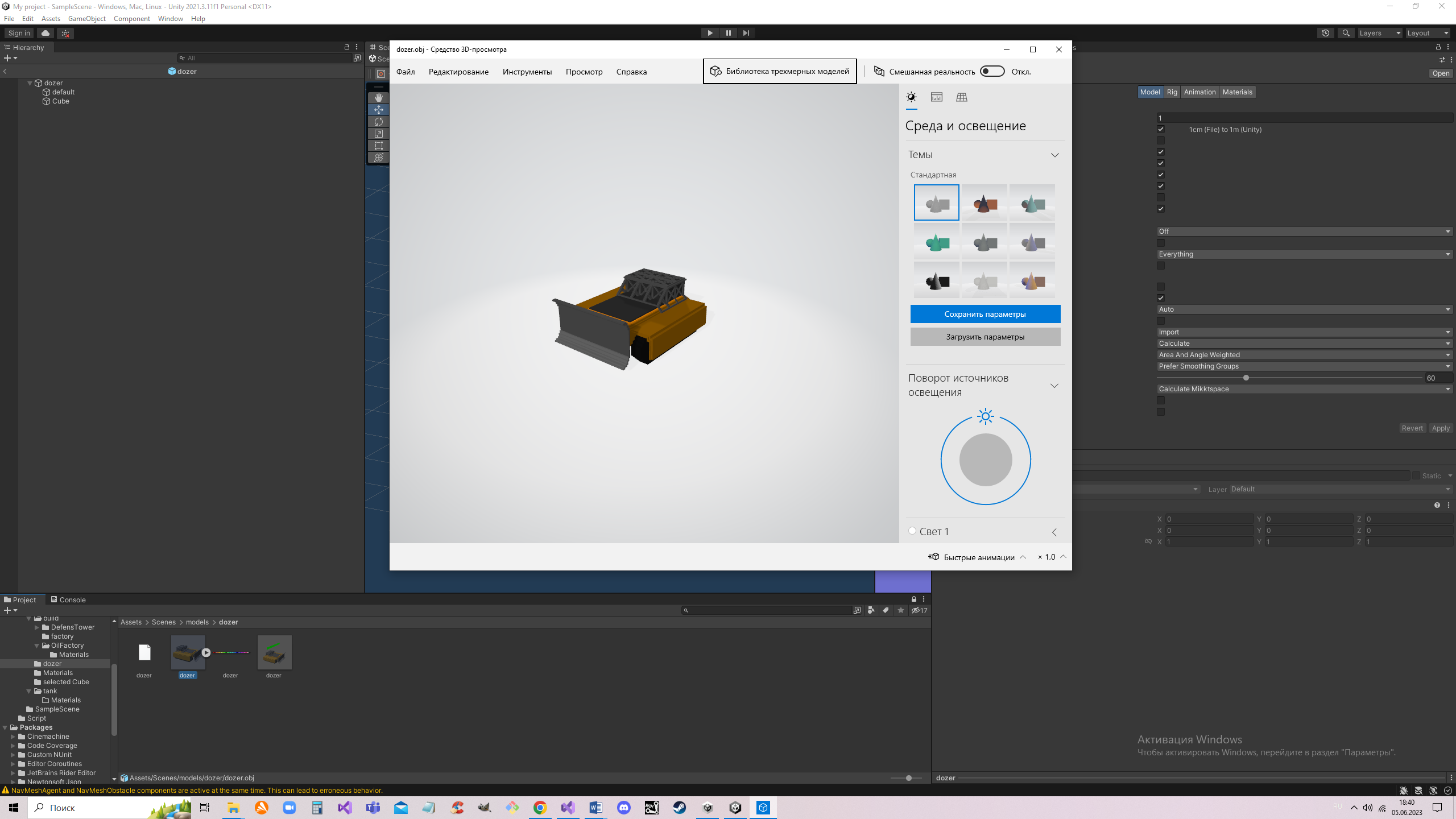


Рисунок 4. Инженерная техника

Далее была создана модель Завода. Внешний вид которой представлен в рисунке 5. Для создания данного завода использовался образ крупной промышленности, монолитных заводов. Именно поэтому основной цвет модели серый. Ворота завода выделены светло серым цветом.

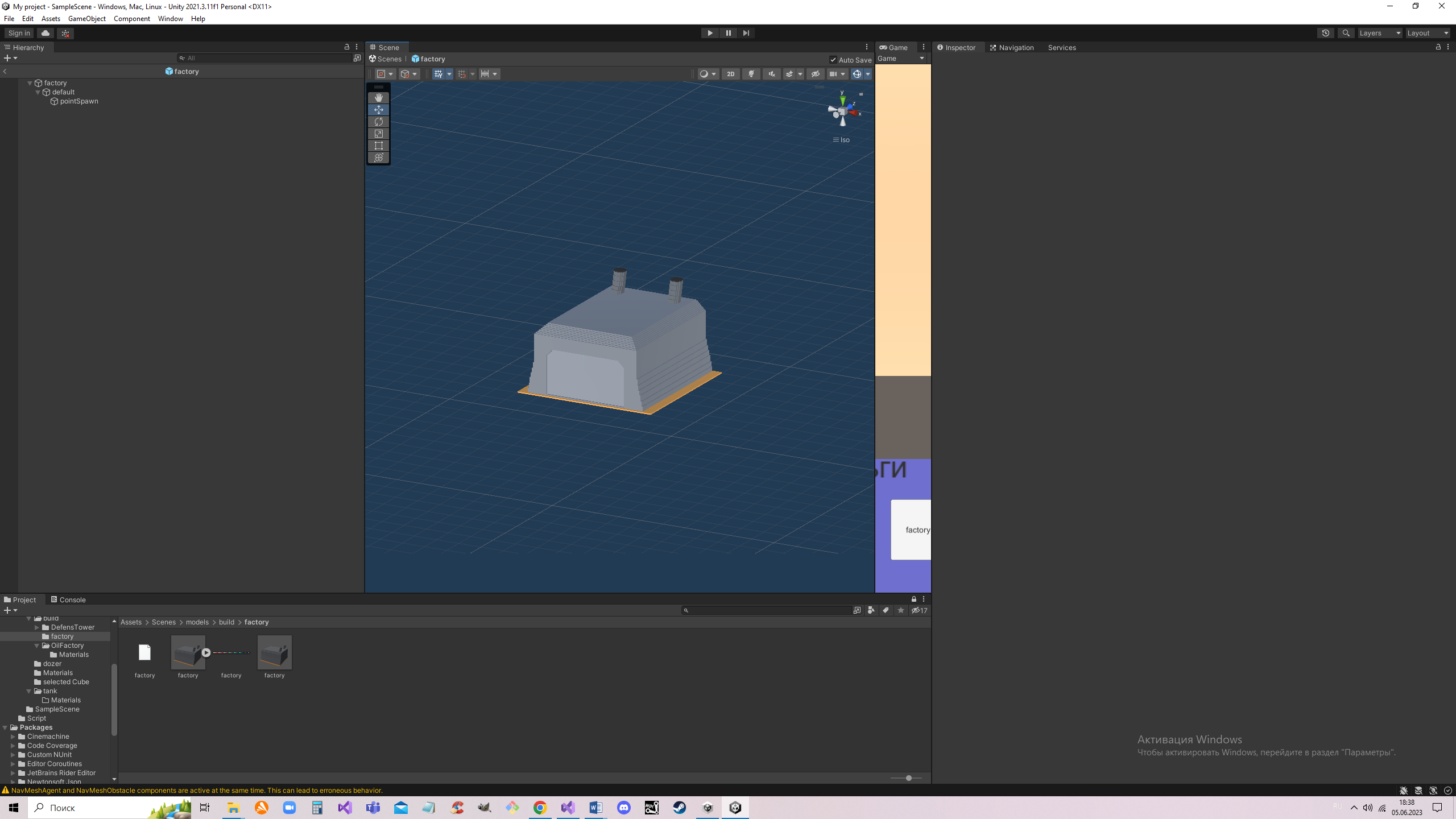


Рисунок 5. Завод

Далее была создана модель Нефтеперерабатывающий завод. Внешний вид этой модели представлен в рисунке 6.

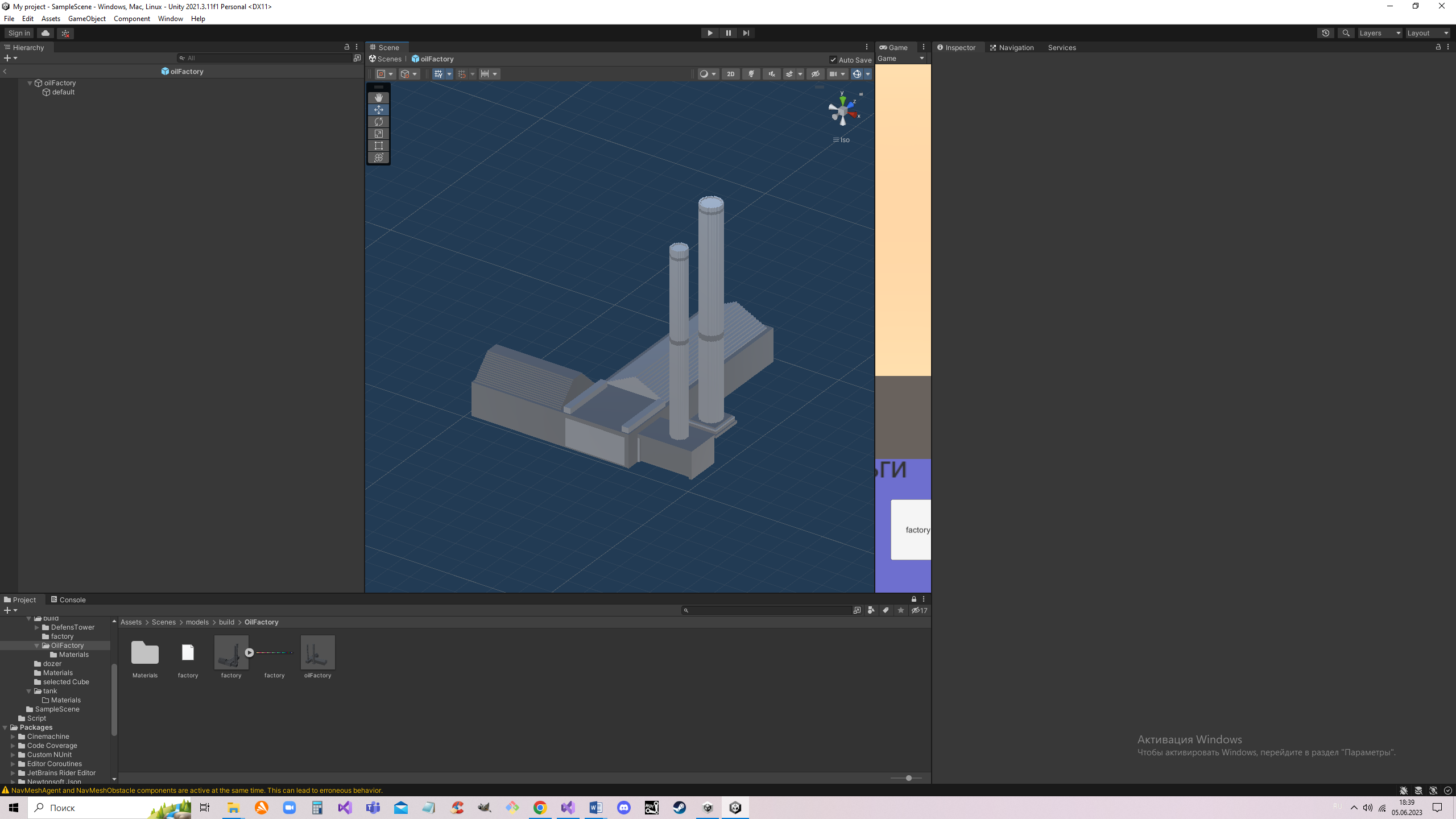


Рисунок 6. Нефтеперерабатывающий завод

В качестве образа для создания был использован образ установки производства серы, взятый с схемы нефтеперерабатывающего завода компании “Газпром”, внешний вид которой представлен в рисунке 7. Причиной данного выбора служит простота внешнего вида здания, что делает создание модели проще, так как убирает необходимость размещения овальных фигур в модели. Так же отсутствие большого числа элементов здания не будет отвлекать игрока и заострять на себе внимание.

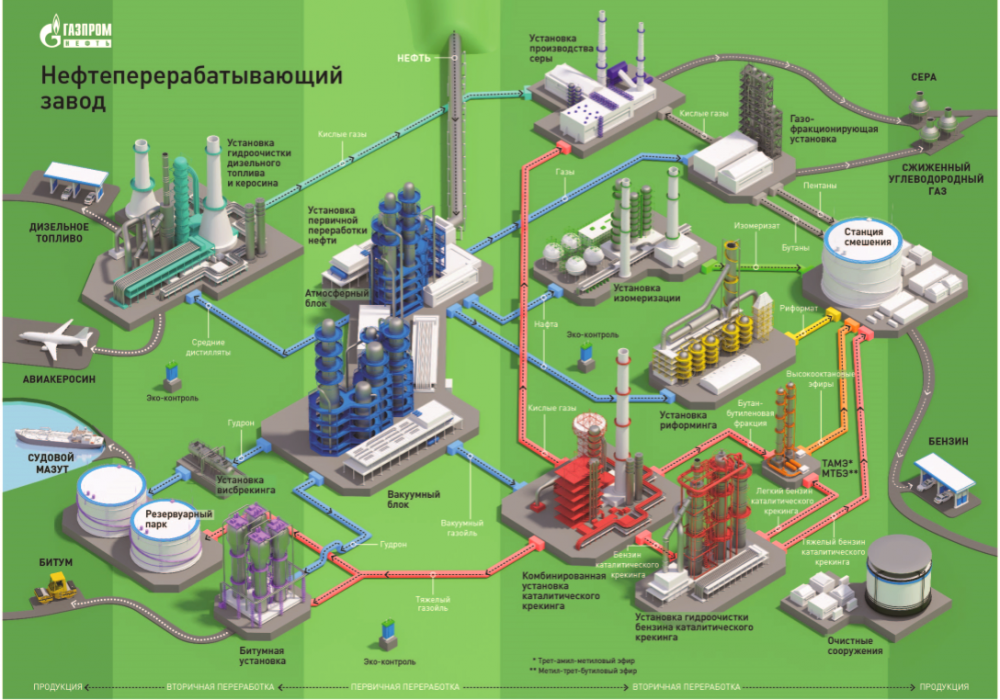


Рисунок 7. Схема нефтеперерабатывающего завода

Далее была создана модель защитной турели. Внешний вид этой модели представлен в рисунке 8.

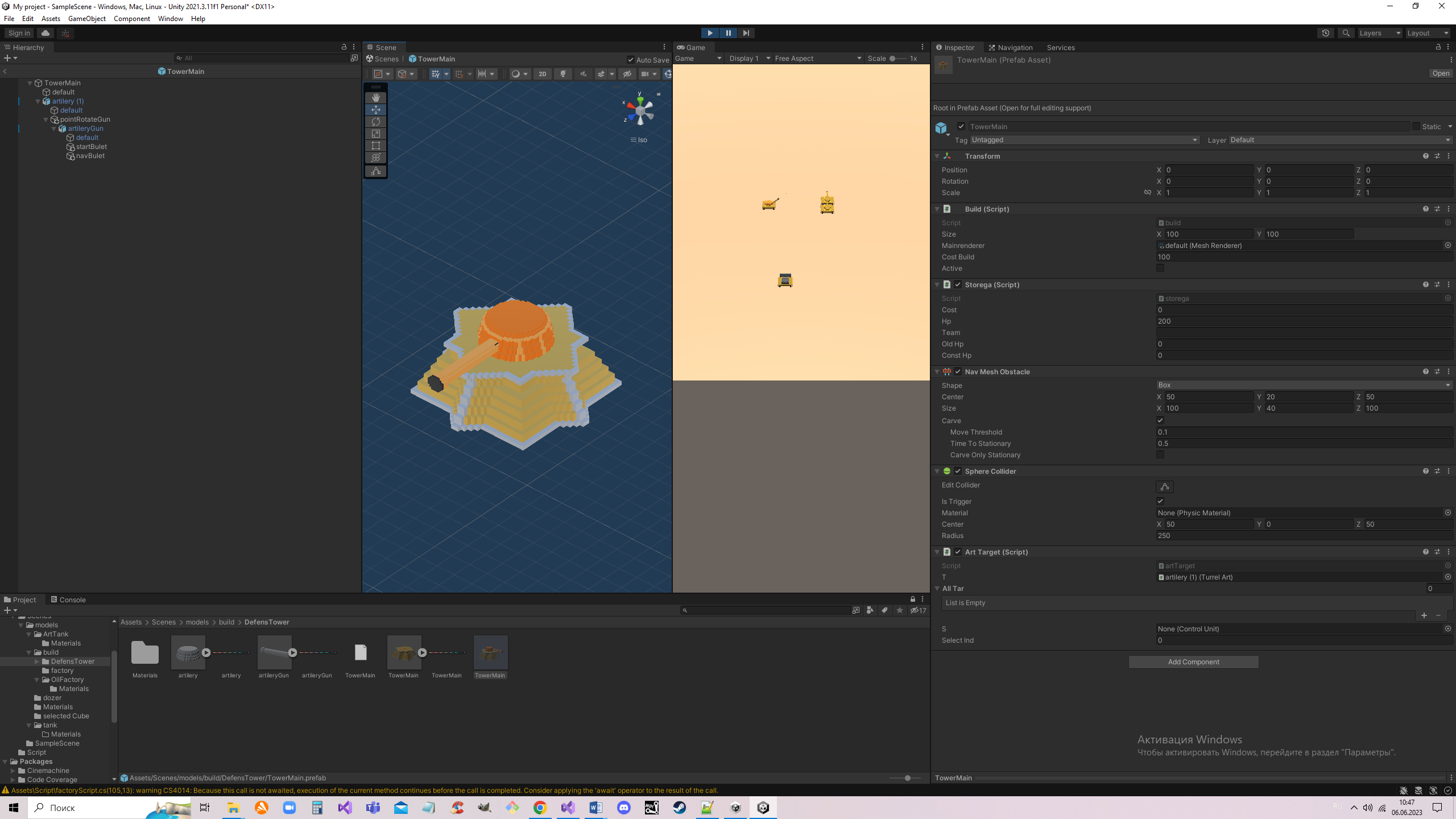


Рисунок 8. Защитная турель

В качестве образа для создания основания используется образ восьмиконечной звёздочной крепости, внешний вид данной крепости представлен в рисунке 8. Это должно делать внешний вид турели более монолитным и внушающим. Орудие и башня модели взяты с модели артиллерийского танка, внешний вид которой представлен в рисунке 4.



Рисунок 9. Манжарабад: звездная крепость «майсурского тигра»

ЗАКЛЮЧЕНИЕ

При создании данной дипломной работы, были использованы знания: системы NavMech в Unity (принцип работы, составные элементы и способы обращения к ним), системы Time в Unity (способы обращения и работы с составными элементами), элемента collider в Unity (возможности и способы применения), системы Canvas (составные элементы и способ обращения к ним). Была изучена программа “MagicaVoxel”.

В ходе дипломной работы был получен опыт в сфере создания трёхмерных моделей в программе “MagicaVoxel”, написания алгоритмов для совместной работы с другими кодами в рамка одной системы игровой механики, создания пользовательского интерфейса, оформления дипломной работы. Так же возникли трудности с реализацией игровых механик, для решения которых использовалась документация “Unity” или разделение задачи с одного кода на несколько.

В итоге получился проект с потенциалом к развитию. Данная работа позволила оценить свои силы в разработке видеоигр в течении ограниченного времени, что поможет в реализации в профессии в будущем.

.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. [Программирование на C# для начинающих](https://www.litres.ru/aleksey-nikolaevich-/programmirovanie-na-c-dlya-nachinauschih-osn-34338191/?lfrom=973672683), В.А. Николаевич ,Изд-во Бомбора, 2023 - 592ст.
2. Head First. Изучаем C#. 4-е издание , Стиллмен Эндрю, .Грин Дженнифер, Изд-во Питер, 2022 – 768ст.
3. Unity и C# геймдев от идеи до реализации. 2-е издание, Бонд Джереми Гибсон, Изд-во Питер, 2022 – 928ст.
4. Кровь, пот и пиксели. Обратная сторона индустрии, Шрейер Джейсон Изд-во Бомбора, 2019 - 368ст.
5. Нажми Reset. Как игровая индустрия рушит карьеры и дает второй шанс, Шрейер Джейсон Бомбора, 2021 – 368ст.
6. Изучаем C# через разработку игр на Unity. 5-е издание, Ферроне Харрисон, Изд-во Питер, 2022 – 400ст.
7. Книга Геймдизайн. Как создать игру, в которую будут играть все, Джесси Шелл, Изд-во Альпина Паблишер, 2022 – 640ст.
8. https://ru.wikipedia.org/wiki/
9. https://docs.unity3d.com/ScriptReference/
10. [https://ru.wikipedia.org/wiki/Компьютерная\_стратегическая\_игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0)

ПРИЛОЖЕНИЯ

**ПРИЛОЖЕНИЕ А**

Код в файле “Control Unit”

using System;

using System.Collections;

using System.Collections.Generic;

using System.Net.WebSockets;

using UnityEditor.PackageManager;

using UnityEngine;

using UnityEngine.AI;

using static UnityEngine.UI.Image;

public class ControlUnit : MonoBehaviour

{

public NavMeshAgent agent;

public bool select = false;

public Transform enemy;

public Vector3 navtarger;

public int startedHp;

void Start()

{

startedHp = gameObject.GetComponent<storega>().hp;

}

void Update()

{

// Выбор юнита

if (Input.GetMouseButton(0) && Input.GetKey(KeyCode.LeftShift))// Проверка на выбор группы техники при помощи комбинации: Левый шифт + левая кнопка мыши

{

Ray movePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(movePosition, out var hitInfo, Mathf.Infinity, 1 << 3))

{

if (hitInfo.collider.gameObject.transform.parent.CompareTag("unit") && hitInfo.collider.gameObject.GetComponentInParent<storega>().team == "player")

{

hitInfo.rigidbody.gameObject.GetComponentInParent<ControlUnit>().select = true;

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А**

Код в файле “Control Unit”

}

else

{

select = false;

}

}

}

else if (Input.GetMouseButtonDown(0))

{

Ray movePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if(Physics.Raycast(movePosition, out var hitInfo, Mathf.Infinity, 1<<3))

{

if (hitInfo.collider.gameObject.CompareTag("earth"))

{

select = false;

}

else if (hitInfo.collider.gameObject.transform.parent.CompareTag("unit") && hitInfo.collider.gameObject.GetComponentInParent<storega>().team == "player")

{

select = false;

hitInfo.rigidbody.gameObject.GetComponentInParent<ControlUnit>().select = true;

}else{

select = false;

}

}

}

if (Input.GetMouseButtonDown(1) && select == true)

{

Ray movePosition = Camera.main.ScreenPointToRay(Input.mousePosition);

if(Physics.Raycast(movePosition, out var hitInfo))

{

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А**

Код в файле “Control Unit”

SetPointNav(hitInfo.point);

}

}

}

public void SetPointNav(Vector3 pos)

{

navtarger = pos;

agent.SetDestination(pos);

}

}