МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа экономики и управления
Кафедра «Информационные технологии в экономике»

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой, д.т.н., с.н.с.
_____/ Б.М. Суховилов /
«_____»_____2021г.

Разработка игрового приложения «Котики vs пришельцы»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.03.2021.265.ВКР

Руководитель, доцент
_____/ Е.А. Конова /
«_____»_____2021 г.
Автор
студент группы зЭУ – 582
_____/ Я.С. Янковская /
«_____» _____2021 г.
Нормоконтролер, доцент
_____/Е.А. Конова/
«_____»_____2021 г.

Челябинск 2021

АННОТАЦИЯ

Янковская Я.С. – Разработка игрового приложения «Котики vs пришельцы» – Челябинск: ЮУрГУ, зЭУ-582, 40 с., 10 ил., 4 табл., библиогр. список - 7 наименований.

Выпускная квалификационная работы выполнена с целью разработки демонстрационной версии трехмерной игры в жанре стратегии.

В ходе работы проведен анализ предметной области. Смоделированы объекты игры и запрограммированы взаимодействия пользователя с игровым миром.

В результате создана демонстрационная версия трехмерной игры. Приложение предназначено для реализации на торговой площадке Steam.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день компьютерные игры прочно заняли свою позицию в индустрии развлечений. Существуют попытки выделить компьютерные игры как отдельную область искусства, наряду с театром, кино и тому подобному. Соревнования по киберспорту, еще 20 лет назад считавшиеся невозможными, на сегодняшний день проходят регулярно на всех уровнях, от районов и городов до международного уровня. Член Международного олимпийского комитета Тони Эстангет в интервью Associated Press рассказал, что киберспортивные соревнования могут включить в список Олимпийских игр 2024 г.

Разработка игр может оказаться не только увлекательным, но и прибыльным делом, примеров этому предостаточно в истории.

В настоящее время, разработка серьезной игры – это процесс, в котором задействована целая команда разработчиков, сложные современные технологии и маркетинговые ходы. Однако существует множество примеров, когда удачно появлялись на рынке игры, сделанные либо маленьким коллективом разработчиков, либо в одиночку, например, всем известный Minecraft.

Целью работы является создание сетевого игрового приложения для 2 игроков. По ходу игры необходимо выбрать сторону противостояния и затем победить, используя игровую механику.

Для достижения цели работы необходимо решить следующие задачи:

- провести анализ предметной области и спроектировать логику игры;
- определить основной функционал и возможности среды разработки;
- разработать приложение и пользовательский интерфейс.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Обзор актуального состояния предметной области

В качестве прототипа использована стратегия реального времени Planetary Annihilation. Общая идея похожа на разрабатываемый продукт: есть трехмерные планеты в качестве игрового поля. Трехмерность с одной стороны усложняет игровой процесс (противник может напасть с невидимой стороны), с другой стороны – повышает интерес пользователя.

У каждого игрока имеется стартовая база, с которой начинается развитие. В дальнейшем, по мере развития появляются различные юниты, которые могут воевать на поверхности, в космосе, либо в принципе уничтожать часть планет. Всего юнитов в базовой версии игры – 150 видов. Благодаря этому игровой процесс очень не предсказуемый и долгий. В среднем одна игра занимает порядка 40-80 минут. Данная игра не подходит для небольшого перерыва и отдыха, требует долгого изучения и постоянной практики для успешной игры.

Кроме того, обилие игровых юнитов заставило разработчиков продумывать достаточно сложную экономическую модель, позволяющую постепенно развивать добычу ресурсов, а затем изучение и постройку боевых юнитов.

О востребованности подобных игр говорит способ, которым разработчики нашли инвестиции для разработки игры – вместо поиска инвесторов они обратились к краудфандингу на платформе Kickstarter для финансирования проекта. На пятый день кампании они собрали $ 450 000, на пятнадцатый день – нужные $ 900 000.

Значительно сокращая количество юнитов (со 150 до 3) достигаем значительного упрощения игрового процесса, экономической модели, при этом оставляя главную суть игры: противостояние 2х игроков с некоторым разнообразием инструментов (юнитов). Сам игра при это становится очень легкой в освоении и быстрой по времени – порядка 15-20 минут на каждую сессию. Можно сказать, что игра после подобных упрощений становится миниигрой.

Миниигры в последнее время стали очень популярны благодаря сервису Steam, в котором каждый разработчик может выложить свой программный продукт, назначив за него определенную цену. Каждый слот для публикации игры стоит $ 100, плюс комиссия 30% за каждую продажу. Разработчики игр, создавая миниигры и выкладывая их для покупки в сервисе Steam, стараются вставлять доступную цену, которая позволит максимизировать количество покупателей. Обычный диапазон цен на миниигры – 35/100 рублей. Даже при выставлении очень маленькой цены достаточно несколько тысяч покупателей чтобы добиться рентабельности, что вполне легко зная количество участников сообщества Steam. По статистике самого сервиса за последний год в среднем на нем регистрируется более 2 миллионов новых пользователей ежедневно, среднемесячная активность пользователей – более 120 миллионов человек (только что-то покупающие или продающие пользователи), постоянно он-лайн от 15 до 25 миллионов пользователей. Такая аудитория позволяет вывести на прибыль любой качественный игровой продукт, особенно в свете последних событий с коронавирусом. Массовый карантин и переход на удаленную работу (учебу) привел к тому что продажи игр выросли на 21%, а время, которое пользователи проводят в играх – на 50%.

Далее кратко проанализированы миниигры, в которых графика и сложность геймплея соизмерима с разрабатываемой в ходе данной работы игрой.

Первая игра – Anomaly Korea, однопользовательская стратегия. Основной сюжет завязан на противостояние игрока и роботов, управляемых компьютером. Задачей игрока является прокладка маршрута для своего отряда с максимальным захватом вражеской территории. В игре использована 3Д графика, вид на карту сверху.

Следующая игра – Turmoil, экономическая стратегия, симулятор нефтедобычи в 19 веке. Игрок берет в аренду участок земли где-то на Западе, нанимает разведчиков, которые могут подсказать начальную точку изысканий, и начинает бурить. Находит нефтяной карман и начинает качать, строить танки для хранения

черного золота, покупать конные «нефтевозы», улучшать трубное хозяйство, использовать дрессированных кротов (этот момент послужил источником многих положительных отзывов об игре), продавать нефть куда выгодней, одним словом, заниматься развитием нефтяного бизнеса и получением сверхприбылей. Через год лицензия заканчивается, и игрок возвращается в маленький пыльный городок, чтобы отдохнуть в салуне, а заодно заключить парочку не очень законных сделок, улучшить свое оборудование, посетить конюха, механика и безумного ученого, а затем снова принять участие в земельном аукционе, стараясь занять самый выгодный участок, перебив ставки соперников. И так по кругу. В игре использована 2Д графика, вид сбоку.

Последняя игра для сравнения – Train Valley, симулятор железной дороги, экономическая стратегия. Каждый уровень начинается примерно одинаково: у игрока есть минимум две станции и большая площадка под строительство железных дорог. Места, на которых можно строить дороги, покрыты квадратной сеткой. Одна клеточка – один кусочек дороги. Каждый элемент имеет определенную стоимость. Есть бюджет, который расходуется на строительство, а если он будет меньше нуля, игрока признают банкротом, и уровень будет проигран.

Периодически звучит сигнал, означающий, что на одной из станций есть поезд, который готов к отправке. Нужно кликнуть на его иконку, и он поедет. А для того, чтобы он доехал на нужную нам станцию, надо поставить в правильное положение стрелки, которые находятся на любых пересечениях путей. В целом основной геймплей построен на переключении стрелок и на рациональном строительстве железных дорог.

За каждый поезд, доставленный на соответствующую станцию, начисляются деньги в бюджет. Чем быстрее – тем больше денег. В игре использована 3Д графика.

Все эти три игры объединяет то, что они очень простые, графика в них так же простая, однако все они оказались очень востребованы покупателями. Сравнительные характеристики представлены в таблице 1.

Таблица 1 – Параметры сравниваемых игр

№	Название	Жанр	Сетевая	Графика	Цена	Количество скачиваний
1	Anomaly Korea	Стратегия	-	3Д	99р.	≈350000
2	Turmoil	Экономическая стратегия	-	2Д	49р.	≈750000
3	Train Valley	Экономическая стратегия	-	3Д	249р.	≈750000
4	Проект «Котики vs Пришельцы»	Стратегия	+	3Д	?	-

Достаточно часто в сервисе Steam проходят различные акции и распродажи со скидками, поэтому реальная цена, по которой в среднем покупают игру покупатели на 25-30% ниже.

1.2 Постановка задачи

В качестве разрабатываемого приложения выбрана сетевая стратегия в реальном времени под рабочим названием «Котики vs Пришельцы». Выбор жанра обосновывается тем, что стратегии входят в топ-3 игровой индустрии (после RPG и шутеров). К тому же любая игра с соперничеством интереснее для потребителя, чем просто игра для одного игрока. В качестве персонажей для игры выбраны с одной стороны не теряющие популярности уже более 50 лет НЛО и пришельцы, с другой стороны – однозначные лидеры всех развлекательных ресурсов сети Интернет – коты. На сегодняшний день и те и другие являются популярными героями различных интернет-мемов, видео, темами для смайлов и прочего.

Сюжет игры достаточно прост, представители каждой игровой расы имеют 3 вида: один производственный юнит и 2 атакующих юнита с различными видами атак. Главная цель игроков состоит в уничтожении всех производственных юнитов соперника, что автоматически приведет к прекращению противостояния. Более подробно различные юниты описаны ниже:

Юниты котов:

- коты в коробке – бегают, могут превращаться в коробки – заводы;

- завод – здания, позволяющие со временем производить разных котов;

- маленькие коты – быстрые юниты, закидывающие инопланетян кокосами;

- черные коты – бьют по инопланетянам в зоне действия мява.

Юниты инопланетян:

- маленькое НЛО – стреляет в котов;

- среднее НЛО – стреляет в котов, имеет функцию самоуничтожения;

- колосс (производственный корабль) – разрушает поверхность острова, получая за это ресурсы для постройки новых НЛО.

Самоуничтожение и другие взрывы наносят дополнительный урон окружающим юнитам.

Следует конкретизировать функциональные требования:

- игра должна представлять собой сетевую стратегию реального времени для 2 игроков;

- в игре должен быть простой, интуитивно понятный интерфейс;

- геймплей должен быть простым, аналогичным подобным играм;

- все юниты должны быть управляемыми игроком, в отсутствии прямого управления юниты либо ожидают, либо сканируют ближнее окружающее пространство на наличие врага с последующей атакой;

- роли юнитов должны соответствовать описанным в сюжетной части;

- победа/поражение должны сопровождаться соответствующим сообщением.

Для более подробного описания требований к игре, ее логики и структуры всего проекта удобно использовать такой инструмент проектирования как UML-диаграммы.

1.3 Проектирование программного продукта с использованием UML

Для лучшего понимания логики приложения удобно представить ее в виде диаграмм.

Стартовое описание удобно начинать с диаграммы прецедентов, которая показывает варианты использования для каждого участника. Диаграмма представлена на рисунке 1.

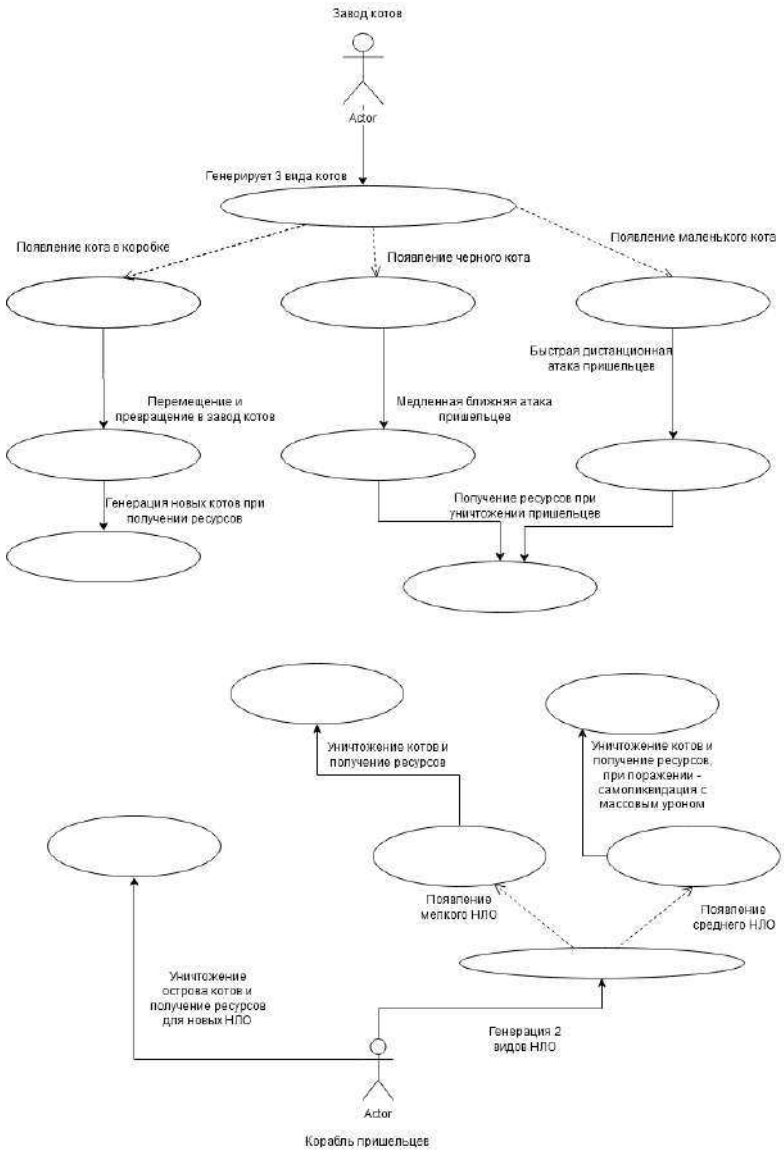

Рисунок 1 – Диаграмма прецедентов

Построена диаграмма последовательностей действий различных объектов. Диаграммы последовательностей используются для уточнения диаграмм прецедентов, более детального описания логики сценариев использования. Это удобное средство документирования проекта с точки зрения сценариев использования.

Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые

результаты обозначают лишь в том случае, если это не очевидно из контекста. Две диаграммы представлены на рисунке 2.

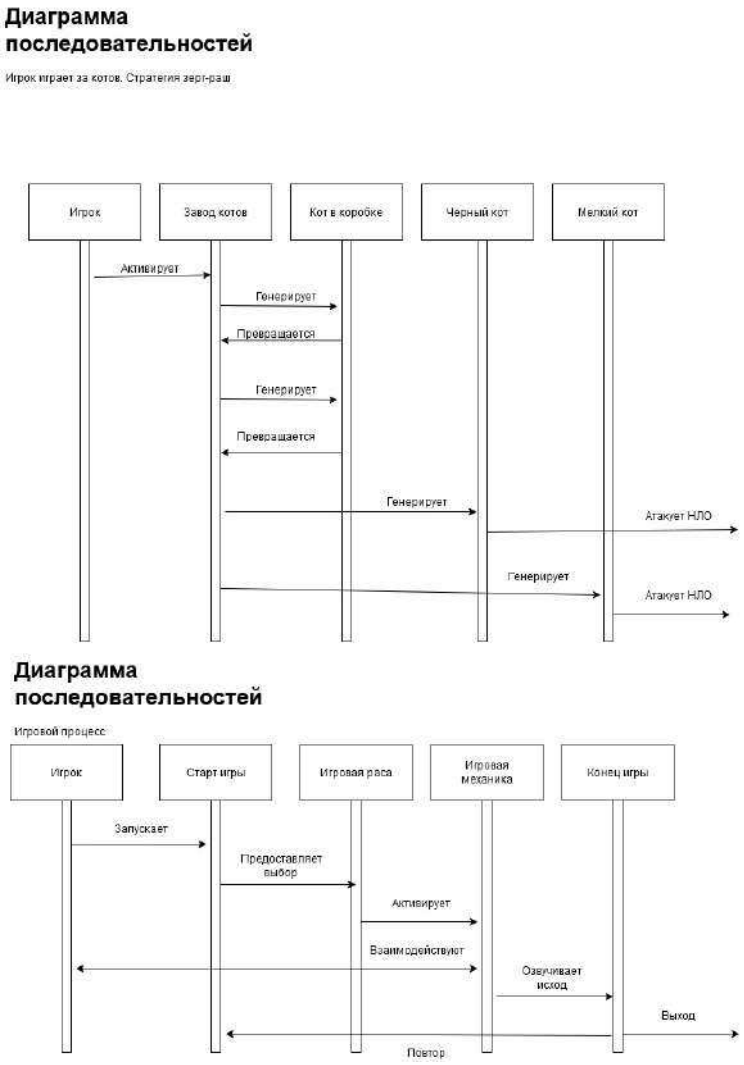

Рисунок 2 – Диаграммы последовательностей.

Составлена диаграмма деятельности. Диаграмма деятельности - один из доступных видов диаграмм, который отражает динамические аспекты поведения системы. По существу, эта диаграмма представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой. Две диаграммы представлены на рисунке 3.

Рисунок 3 – Диаграммы деятельности.

Разработана диаграмма классов. Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня

абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации). В дальнейшем наличие этой диаграммы упрощает процесс разработки. Диаграмма представлена на рисунке 4.

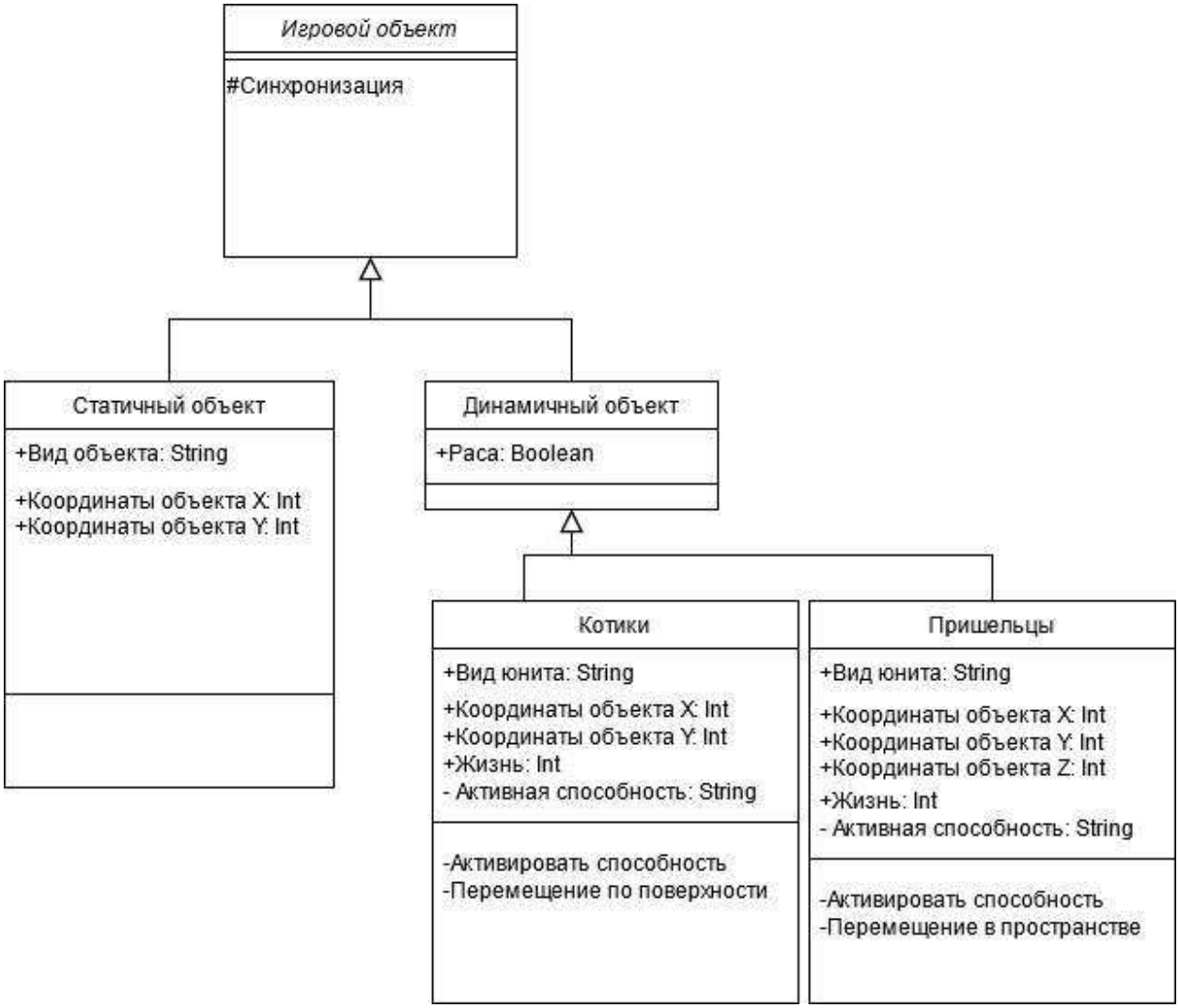

Рисунок 4 – Диаграмма классов.

Поскольку игра представляет собой многопользовательское приложение с отдельным сервером в качестве инструмента синхронизации, так же важно наглядно показать диаграмму развертывания (рисунок 5).

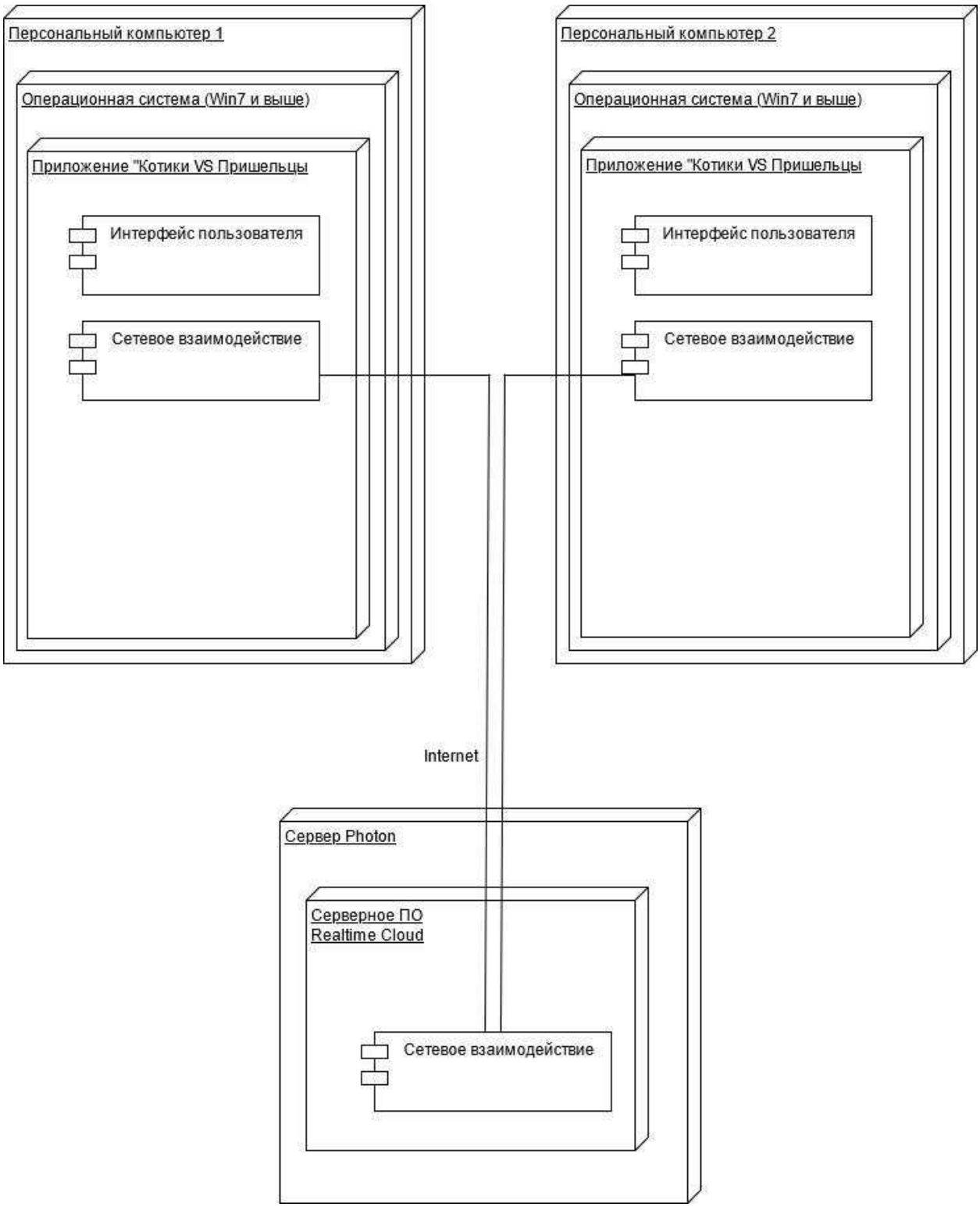

Рисунок 5 – Диаграмма развертывания.

Завершающим этапом теоретической части является обоснование инструментальных средств, с помощью которых должны быть реализована игра.

1.4 Инструментальные средства разработки

В качестве основного инструментального средства выбрана среда разработки Unity. Она позволяет писать игры под 25 платформ: от Windows, Linux и macOS, мобильных платформ и консолей до различных VR и Smart TV. Выбор был основан на нескольких факторах, таких как:

- специализация среды именно на разработке игровых приложений;
- большой набор инструментов;
- кроссплатформенность;
- условная бесплатность программы (при использовании бесплатной версии пред стартом игры будет показан логотип Unity.

В современной версии Unity имеется много важных и полезных инструментов. Новые инструменты графического интерфейса отдельно для 3D и 2D игр. Полноценный звуковой редактор – стало возможным в реальном времени объединять различные звуки, добавлять эффекты, связывать их с событиями в игре. Поддержка WebGL – игры работают напрямую в браузере без установки веб-плеера. Глобальное освещение в реальном времени для консолей нового поколения, ПК и мобильных платформ. Отражение света в реальном времени. Физически корректные материалы , например объект «дерево» теперь плавает в воде сам, без дополнительных настроек и сценариев. Новые возможности 2D физики: точечные силы притягивания и отталкивания; тангенциальные силы (силы, направленные по касательной к поверхности объекта); силы, направленные вдоль любых осей; одностороннее столкновение. Отслеживание загрузки процессора, видеокарты и памяти на временной шкале в режиме реального времени [2].

На данный момент это один из самых быстроразвивающихся движков, разработчики которого постоянно улучшают его и внедряют новые функции.

Как правило, среда разработки предоставляет множество функциональных возможностей, позволяющих их задействовать в различных играх, в которые входят моделирование физических сред, карты нормалей, динамические тени и многое

другое. В отличие от многих программ, у Unity имеется два основных преимущества: наличие визуальной среды разработки и межплатформенная поддержка. Первый фактор включает не только инструментарий визуального моделирования, но и интегрированную среду, цепочку сборки, что направлено на повышение производительности разработчиков, в частности, этапов создания прототипов и тестирования. Под межплатформенной поддержкой предоставляется не только места развертывания (установка на персональном компьютере, на мобильном устройстве, консоли и т. д.), но и наличие инструментария разработки [3].

Стоит отметить, что среда разработки оснащена модульной системой компонентов Unity, с помощью которой происходит конструирование игровых объектов, когда последние представляют собой комбинируемые пакеты функциональных элементов. В отличие от механизмов наследования, объекты в Unity создаются посредством объединения функциональных блоков, а не помещения в узлы дерева наследования. Такой подход облегчает создание прототипов, что актуально при разработке игр.

Отдельно необходимо обосновать выбор сервиса photonengine.com, который позволяет легко синхронизировать приложения благодаря собственным серверам. В среде Unity в различное время поддержка сетевой разработки то включалась, то отключалась, то модифицировалась с появлением различных коллизий в уже готовых проектах. Ресурс photonengine.com был выбран как альтернатива сетевому функционалу Unity.

Photon Engine это независимый сетевой движок и многопользовательская платформа. Он представляет собой глобальный кроссплатформенный многопользовательский игровой сервер как сервис (SaaS, Cloud) для синхронных и асинхронных игр и приложений. SDK доступны для Android, iOS, .NET., Mac OS, Unity 3D, Windows, Unreal Engine, HTML5 и других.

Движок интегрируется со множеством платформ. Например, Marmalade, Unity3D, iPhone/iPad, Adobe Flash, PC Windows, Mac OS, Microsoft .NET, Java, Brew,

Android, Silverlight и т.д. Ядро движка написано на C/C++ и использует для расширения возможностей .NET/C# с высоким UDP, что очень важно для реализации мультиплеерных шутеров от первого лица, массовых мультиплеерных онлайновых игр и пр.

Photon network engine лицензирован многими известными во всём мире компаниями, такими как: Konami, Taito, CMUNE, BigPoint, World Golf Tour, Brainz Interactive, Minor Studios, GLU и др. Они использовали данный движок в своих популярных игровых проектах, таких как: хитовая MMOG «The Mummy» от BigPoint на технологии движка Unity3D, первый в мире социальный 3D-шутер «Paradise Paintball» от Cmune на движке Unity, мультиплеерный 3DFPS «BeGone» от Nplay на движке Unity, лего-платформенная игра «Atmosphir» от Minor Studios на движке Unity, обучающий проект «Train Your Brain with Doctor Kawashima» от BBG Entertainment/Chimera Entertainment на движке Unity для Mac OS и Windows, F2P шутер для социальной сети Facebook «Social Street Soccer» от Brainz Interactive на движке Unity3D Flash, «World Golf Tour» от Exit Games Development Services на Adobe Flash, танковая аркада для iPhone «Ping Tanks» от Ping pilots и Exit Games на OpenFeint Playtime SDK и т.д. Эти игры объединяют сотни тысяч пользователей-игроков.

Движок Photon network engine поставляется по двум лицензиям, кроме стандартной предлагается Indie. Имеется ограниченная демо-версия, дающая в течение 30 дней бесплатно протестировать возможности движка.

Если есть цель – разработка мультиплеерной игры (например, FPS) или многопользовательской игры (MMOG) на PC, то для обеспечения стабильной и качественной работы сетевой движок Photon – один из лучших вариантов.

Выводы по первому разделу

В разделе проведен анализ предметной области, изучены некоторые аналогичные программные продукты, сформулировано обобщенное техническое задание, спроектированы UML диаграммы и обзорно рассмотрены программные средства, которые использованы при разработке игры.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Моделирование игрового поля

Предварительная работа включает в себя создание игровых объектов, создание локации и настройка камеры, которая будет формировать игровое поле для пользователя. Наглядно этот процесс представлен последовательностью рисунков. На рисунке 6 представлено созданное игровое поле.

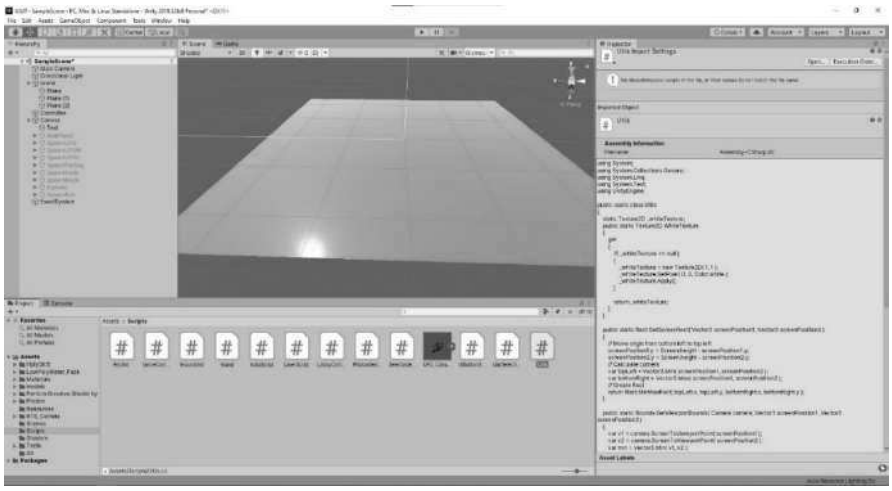

Рисунок 6 – Пустое игровое поле

Это первый этап создания игровой локации. Для создания итоговой игровой локации необходимо изменить структуру стартового игрового поля и добавить на поле созданные модели камней и деревьев.

На рисунке 7 представлена настройка камеры, которая будет формировать изображение для игрока.

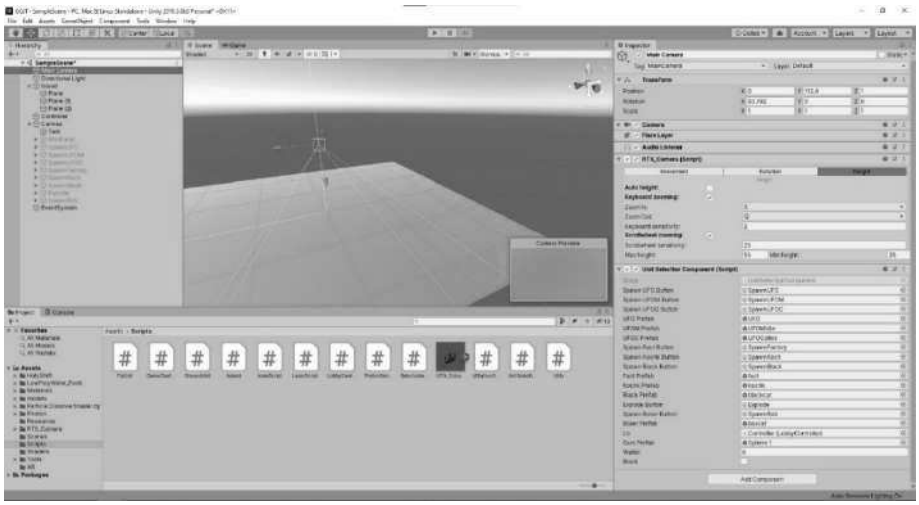

Рисунок 7 – Настройка основной камеры

Камера является отдельным объектов игры и от ее правильной настройки зависит комфорт игрока в игре. Важно настроить линейное и угловое перемещение, а так же физические ограничения, чтобы камера не могла провалиться сквозь остров или улететь в небо.

После настройки локации, добавления моделей, настройки камеры процесс подготовки игровой локации завершен. На рисунке 8 представлена готовая игровая локация после ее программной генерации.

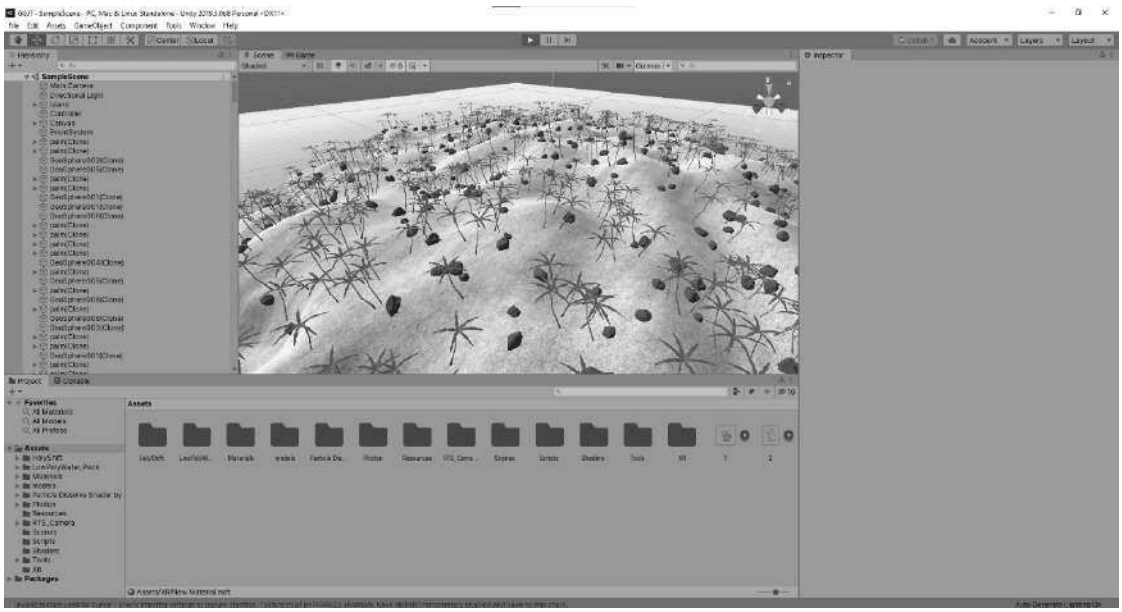

Рисунок 8 – Игровая локация после генерации

Процесс формирования локации и подготовки игрового поля к игре представлен следующим кодом в листинге 1.

Листинг 1 – Код генерации игровой локации

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using UnityEngine;
using Random = UnityEngine.Random;
public class Island : MonoBehaviour
{
    public static int zSize = 250;
```

```
    public static int xSize = 250;
    private static MeshFilter _meshFilter;
    private static MeshCollider _meshCollider;
    public AnimationCurve curve;
    public static Mesh _mesh;
    private Camera _mainCamera;
    private int seed;
    public LobbyController lb;
    public static Dictionary<Vector2Int, Transform> palms = new Diction-
ary<Vector2Int, Transform>();
    public GameObject[] palmPrefabs;
    public GameObject NukeExplosions;
    public GameObject SphereExplosions;
    public static GameObject NukeExplosion;
    public static GameObject SphereExplosion;
    void Start()
    {
        NukeExplosion = NukeExplosions;
        SphereExplosion = SphereExplosions;
        _meshFilter = GetComponent<MeshFilter>();
        _meshCollider = GetComponent<MeshCollider>();
        _mainCamera = Camera.main;
        seed = Random.Range(0, 999999);
    }
    public void Generate()
    {
        var center = new Vector2(zSize / 2f, xSize / 2f);
        var scaler = 6f;
        _meshFilter.mesh = _mesh = new Mesh();
        _mesh.name = "Procedural Grid";
        var vertices = new Vector3[(xSize + 1) * (zSize + 1)];
        for (int i = 0, z = 0; z <= zSize; z++)
        for (int x = 0; x <= xSize; x++, i++)
        {
```

```
            var px = x / (float) xSize;
            var py = z / (float) zSize;
            var no = Mathf.PerlinNoise(px * scaler + seed, py * scaler) *
10f;
            var no1 = Mathf.PerlinNoise(px * scaler * 2 + py * 3, seed + py *
scaler * 2) * 7f;
            var no2 = Mathf.PerlinNoise(px * scaler * 3, py * scaler * 3 + px
* 3 + seed) * 4f;
            var dist = Vector2.Distance(center, new Vector2(x, z));
            var step = curve.Evaluate(dist / 3f + no + no1);
            var h = no - no1 - no2 - dist / 10f + step * 20f;
            if (h > Random.Range(1f, 5f) && no2 > 0.3f && Random.Range(1,
100) > 95)
            {
                palms[new Vector2Int(x, z)] =
                    Instantiate(palmPrefabs[Random.Range(0, palmPre-
fabs.Length)],
                        new Vector3(x - xSize / 2, h, z - zSize / 2),
                        Quaternion.Euler(0, Random.Range(0, 360),
0)).transform;
            }

            vertices[i] = new Vector3(x - center.x, h, z - center.y);
        }

        _mesh.vertices = vertices;
        int[] triangles = new int[xSize * zSize * 6];
        for (int ti = 0, vi = 0, z = 0; z < zSize; z++, vi++)
        for (int x = 0; x < xSize; x++, ti += 6, vi++)
        {
            triangles[ti] = vi;
            triangles[ti + 3] = triangles[ti + 2] = vi + 1;
            triangles[ti + 4] = triangles[ti + 1] = vi + xSize + 1;
            triangles[ti + 5] = vi + xSize + 2;
```

```
        }
        Vector2[] uv = new Vector2[vertices.Length];
        for (int i = 0, z = 0; z <= zSize; z++)
        for (int x = 0; x <= xSize; x++, i++)
            uv[i] = new Vector2(x / (float) xSize, z / (float) zSize);
        _mesh.SetUVs(0, uv);
        _mesh.triangles = triangles;
        _mesh.RecalculateNormals();
        UpdateMesh(_mesh);
        UpdateMeshCollider(_mesh);
    }
```

Следующий метод, представленный на листинге 2, изменяет существующую карту острова как будто на координатах point был взрыв размера size. При этом деформируется сетка полигонов модели острова, а так же статические объекты (камни и пальмы) попавшие в радиус - уничтожаются. Котам попавшим в радиус взрыва наносится урон (вызов функции killCatsInRadius).

Листинг 2 – Деформация острова

```
    public static Mesh Explode(Vector3Int point, int size)
    {
        Mesh newMesh = _mesh;
        var center = new Vector3(xSize / 2f, 0, zSize / 2f);
        var vertices = newMesh.vertices;
        var centerPoint = point + center;
        for (var x = -size; x < size; x++)
        for (var z = -size; z < size; z++)
        {
            var newPoint = centerPoint + new Vector3(x, 0, z);
            if (newPoint.x > 0 && newPoint.x <= xSize + 1 && newPoint.z > 0
&& newPoint.z <= zSize + 1)
            {
                int index = Mathf.FloorToInt(newPoint.z) * (zSize + 1) +
Mathf.FloorToInt(newPoint.x);
                var toDist = new Vector3(newPoint.x, vertices[index].y,
newPoint.z);
                var dist = size - Vector3.Distance(centerPoint, toDist);
                if (dist > 0)
                {
                    var palmPos = new Vector2Int(Mathf.FloorToInt(new-
Point.x), Mathf.FloorToInt(newPoint.z));
                    palms.TryGetValue(palmPos,
```

```
                    out var palm);
                if (palm != null)
                {
                    palms[palmPos] = null;
                    Destroy(palm.gameObject);
                }
                vertices[index].y -= dist + Random.Range(-.5f, .5f);
            }
        }
    }
    killCatsInRadius(point, size);
    newMesh.vertices = vertices;
    newMesh.RecalculateNormals();
    return newMesh;
```

Следующие методы, представленные на листинге 3, реализуют нанесение урона котам и пришельцам в радиусе size от координат pos.

Листинг 3 – Анализ и применение наносимого урона

```
public static void killCatsInRadius(Vector3Int pos, int size)
{
    foreach (var cat in GameObject.FindGameObjectsWithTag("ground"))
        if (Vector3.Distance(pos, cat.transform.position) < size)
            cat.GetComponent<GroundUnit>().hp -= 2 + Random.Range(0,1);
}
public void killUFOInRadius(Vector3Int pos, int size)
{
    foreach (var cat in GameObject.FindGameObjectsWithTag("fly"))
        if (Vector3.Distance(pos, cat.transform.position) < size)
            cat.GetComponent<FlyUnit>().hp -= 2;
}
```

Следующие два метода, представленные в листинге 4, обновляют физическую и графическую модель острова после взрыва.

Листинг 4 – Пересчет геометрии острова.

```
public static void UpdateMeshCollider(Mesh mesh)
{
    _meshCollider.sharedMesh = mesh;
}
public static void UpdateMesh(Mesh mesh)
```

```
    {
        _meshFilter.mesh = mesh;
    }
```

Последний в данном блоке метод, который представлен на листинге 5, отвечает за анимацию взрыва и плавное изменение модели острова (создание кратера).

Листинг 5 – Визуализация взрыва

```
public static IEnumerator peka(Vector3Int point, int size)
    {
        //UpdateMeshCollider(Explode(point, size));
        if (size > 16)
            Destroy(Instantiate(NukeExplosion, point, Quaternion.identity),
10f);
        else
            Destroy(Instantiate(SphereExplosion, point, Quaternion.iden-
tity), 5f);
        for (int i = 0; i < size; i++)
        {
            UpdateMesh(Explode(point, i));
            //lb.sendMapExplosion(point, i);
            yield return new WaitForSeconds(0.005f);
        }
        UpdateMeshCollider(_mesh);
        //updater();
    }
```

2.2 Сетевая синхронизация

Лобби (комната ожидания игроков) реализована через примитивы фреймворка Photon. LobbyController наследует класс MonoBehaviourPunCallbacks и реализует интерфейс IOnEventCallback, что позволяет классу получать и обрабатывать события связанные с мультиплеером.

Созданы следующие события:

MapSync - синхронизирует карту между игроками;

MapExplosion - синхронизация взрывов на карте между игроками;

PalmSync - синхронизация статических объектов между игроками (пальмы, камни).

Win - событие победы игрока

Lose - событие проигрыша игрока

Так же обрабатываются события фреймворка Photon:

OnPlayerEnteredRoom - событие входа игрока в комнату (связь 2 игроков друг с другом и начало игры);

OnDisconnected - отключение игрока от сервера;

OnConnectedToMaster - событие подключения игрока к серверу Photon;

Когда игрок заходит в игру фреймворк Photon подключает его к своему серверу и срабатывает событие OnConnectedToMaster, после чего игрок либо заходит в комнату к другому игру, либо (если игроков нет) ждет их. После того, как игрок зашел в комнату к другому игроку, срабатывает событие OnPlayerEnteredRoom, после чего срабатывают события MapSync, PalmSync для синхронизации карты между игроками. Во время игры происходят взрывы, которые синхронизируются через событие MapExplosion. Если какой-либо игрок выигрывает или проигрывает вызываются события Win/Lose.

Фрагмент кода, организующего сетевое взаимодействие, представлен на листинге 6:

Листинг 6 – Синхронизация двух игроков

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using ExitGames.Client.Photon;
using Photon.Pun;
using Photon.Realtime;
```

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using Random = UnityEngine.Random;
public class LobbyController : MonoBehaviourPunCallbacks, IOnEventCallback
{
    public Island Island;
    public Text text;
    public GameObject[] palmPrefabs;
    public GameObject WaitPanel;
    public bool gameStarted = false;
    public Texture2D catcursor;
    public Texture2D aliencursor;
    void Start()
    {
        if (Screen.fullScreen)
            Screen.fullScreen = false;
        PhotonNetwork.NickName = "a" + Random.Range(1, 1000);
        PhotonNetwork.GameVersion = "2";
        PhotonNetwork.AutomaticallySyncScene = true;
        PhotonNetwork.PhotonServerSettings.AppSettings.FixedRegion = "ru";
        PhotonNetwork.AddCallbackTarget(this);
        PhotonNetwork.ConnectUsingSettings();
    }
    public override void OnConnectedToMaster()
    {
        print("connected to master");
        PhotonNetwork.JoinRandomRoom();
    }
    public void setText(string text)
    {
        WaitPanel.SetActive(true);
        WaitPanel.GetComponentInChildren<Text>().text = text;
        PhotonNetwork.Disconnect();
```

```
    }
    public override void OnJoinedRoom()
    {
        print("joined");
        Island.Generate();
        if (!PhotonNetwork.IsMasterClient)
        {
            var pos = Vector3.right * 80 + new Vector3(0, 25, 0);
            var poscam = pos;
            poscam.y = Camera.main.transform.position.y;
            Camera.main.transform.position = poscam;
            PhotonNetwork.Instantiate("UFOCollos", pos, Quaternion.iden-
tity).GetComponent<FlyUnit>().money=30;
            StartCoroutine(startTheGame());
        }
      if (PhotonNetwork.IsMasterClient)
            Cursor.SetCursor(catcursor, new Vector2(0, 0), Cursor-
Mode.ForceSoftware);
      else Cursor.SetCursor(aliencursor, new Vector2(0, 0), Cursor-
Mode.ForceSoftware);
    }
    public IEnumerator startTheGame()
    {
        yield return new WaitForSeconds(3f);
        gameStarted = true;
    }
```

Остальные события работают как написано в описании и тут не приводятся.

2.3 Управление юнитами

Управление игровыми юнитами реализовано с помощью 2 разных классов для наземных и летающих существ.

Летающие юниты (FlyUnit):

Start - метод, запускаемый при создании юнита. Описывает его инициализацию и прибавление счетчика юнитов.

OnDestory - обработчик события уничтожения объекта. Убавляет счетчик количествава юнитов.

Update - метод вызываемый каждый кадр. Управляет движением юнита и проверками здоровья юнита.

dig - метод вызывает взрыв под юнитом.

shot - выстрел в заранее определенную цель.

OnPhotonSerializeView – синхронизация.

OnTriggerStay - метод заранее пускается каждый кадр, когда рядом находится объект, который можно считать целью. Если у юнита нет цели в данный момент, то появившийся рядом кот станет целью.

OnTriggerExit - если цель далеко ушла, то она перестаёт быть целью.

Наземные юниты (GroundUnit):

Start - метод, запускаемый при создании юнита. Описывает его инициализацию и прибавление счетчика юнитов.

OnDestory - обработчик события уничтожения объекта. Убавляет счетчик количествава юнитов.

Update - метод вызываемый каждый кадр. Управляет движением юнита и проверками здоровья юнита.

meow - метод мяукания. способность одного вида котов. Наносит урон НЛО в радиусе поражения.

shot - метод выстрела путем кидания кокоса в заранее выбранную цель.

OnPhotonSerializeView – синхронизация.

OnTriggerEnter - нанесение урона НЛО рядом при мяукании.

OnTriggerStay - метод заранее пускается каждый кадр, когда рядом находится объект, который можно считать целью. Если у юнита нет цели в данный момент, то появившееся рядом НЛО станет целью.

OnTriggerExit - если цель далеко ушла, то она перестаёт быть целью.

Фрагмент кода управления котами (Листинг 7) представлен ниже:

Листинг 7 – Управление юнитами котов

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Photon.Pun;
using UnityEngine;
using Random = UnityEngine.Random;
public class GroundUnit : MonoBehaviour, IPunObservable
{
    public Island Island;
    public Rigidbody enemyTarget;
    public Vector3 goTarget;
    private Rigidbody rb;
    public int hp = 10;
    public GameObject kokPrefab;
    public float cooldown = 0f;
    public bool isDead = false;
    public int type = 0; // 0 box 1 koch 2 black 3 factory
    public bool meowing = false;
    public SphereCollider trigger;
    public float money = 0;
    public static int totalFact = 0;
    public GameObject sphere;
    private void Start()
    {
        if (type == 3) totalFact += 1;
        Island = GameObject.FindWithTag("island").GetComponent<Island>();
        rb = GetComponent<Rigidbody>();
        var photonView = GetComponent<PhotonView>();
        if (!photonView.IsMine) rb.constraints = RigidbodyCon-
straints.FreezeAll;
```

```
        trigger = GetComponents<SphereCollider>().First(p => p.isTrigger);
        if (type == 2) trigger.radius = 0;
    }
    private void OnDestroy()
    {
        if (type == 3) totalFact -= 1;
    }
    private void Update()
    {
        money += Time.deltaTime;
        meowing = false;
        if (type == 2) trigger.radius = 0;
        rb.AddForce((goTarget - transform.position).normalized / 5f, Force-
Mode.VelocityChange);
        switch (type)
        {
            case 1:
                if (enemyTarget != null && Time.time - cooldown > 0.3f)
shot();
                break;
            case 2:
                if (Time.time - cooldown > 1f) meow();
                break;
        }
        if (hp <= 0 || transform.position.y < 0) isDead = true;
    }
    private void meow()
    {
        meowing = true;
        trigger.radius = 10;
    }

    private void shot()
    {
```

```
        cooldown = Time.time;
        var position = transform.position;
        var rb = Instantiate(kokPrefab, position, Quaternion.identity).Get-
Component<Rigidbody>();
        rb.velocity =
            (enemyTarget.transform.position - position + enemyTarget.veloc-
ity * Random.Range(0f, 2f)).normalized * 30f +
            Random.insideUnitSphere * 2f;
        Destroy(rb.gameObject, 5f);
    }
```

Остальные события и методы работают как написано в описании и тут не приводятся.

После создания всех моделей, программирования поведения объектов на карте и создания пользовательского меню для управления юнитам итоговое игровое приложение для игрока, играющего за пришельцев, выглядит как показано на рисунке 9.

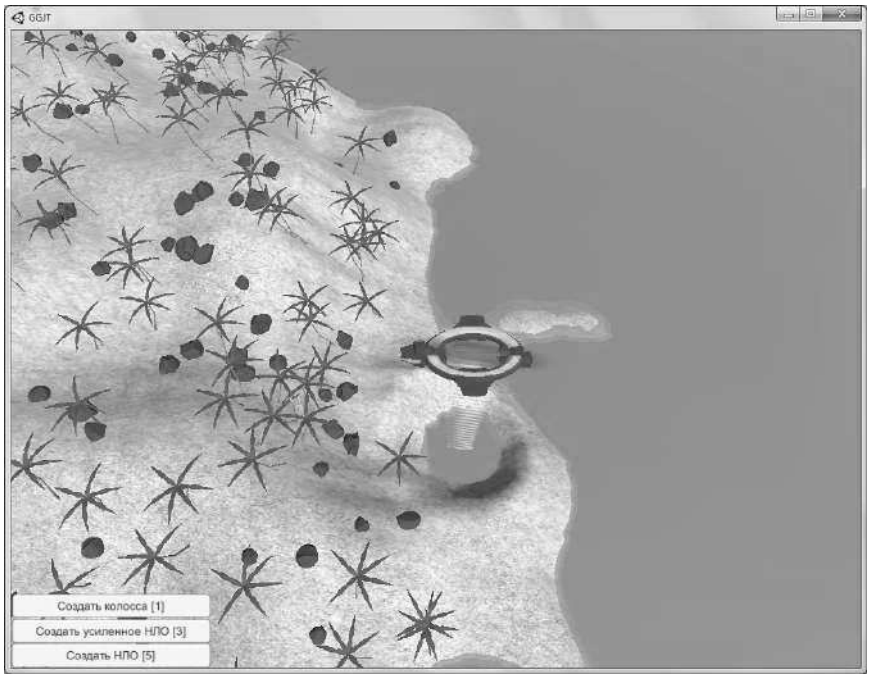

Рисунок 9 Окно игрока за расу пришельцев

Итоговое игровое приложение для игрока, играющего за котов, выглядит как показано на рисунке 10.

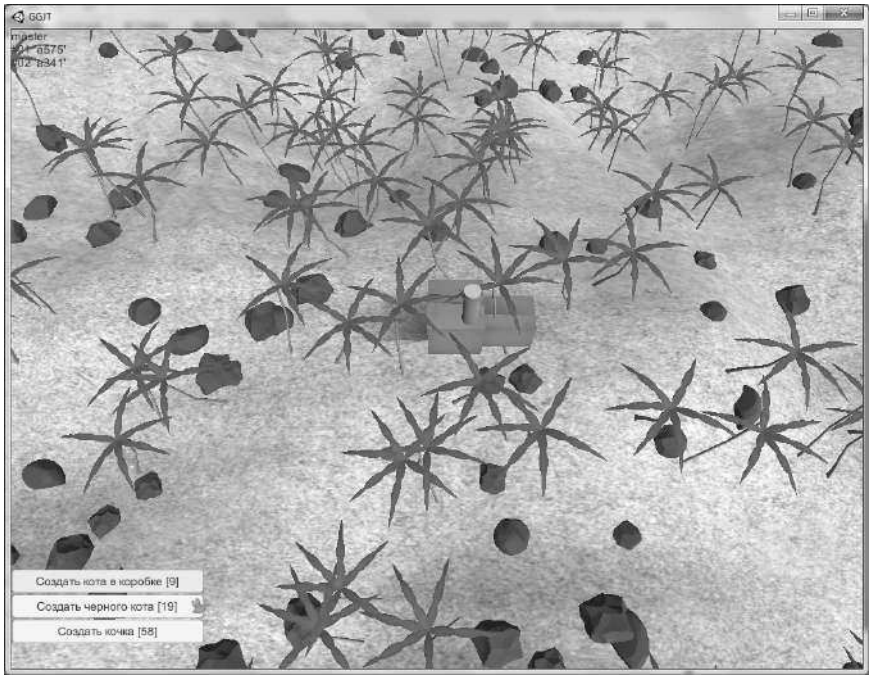

Рисунок 10 Окно игрока за расу котиков

Визуально картинка у обоих игроков выглядит одинаково, однако функционал и управление немного отличаются из-за особенностей игровых рас.

Выводы по второму разделу

Во втором разделе описана программная структура игры и все использованные в программе методы. Ключевые методы представлены фрагментами кода в виде листингов.

3 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

Разрабатываемая игра является достаточно простой, поэтому разработчик выполняет функции всех необходимых членов команды – дизайнера и программиста. Для реализации любого подобного проекта, необходимо достаточно мощное оборудование. разместить готовый проект предполагается на торговой площадке Steam. Далее подсчитаем расходы и ожидаемые доходы после выставки проекта на продажу.

3.1 Расчет трудозатрат

На разработку такого небольшого проекта необходимо около четырех недель. Сколько бы стоило нанять трех специалистов из областей гейм-дизайна, программирования и бухгалтера приведено ниже в таблице 2, взята средняя зарплата по региону Челябинской области с сайта поиска работ headhanter.ru .

Таблица 1 − Расчет трудозатрат

Показатель	1 месяц
Заработная плата дизайнера , руб.	40000
Заработная плата программиста C#, руб.	50000
Бухгалтер, руб.	25000
Итого, руб.	115000

3.2 Расчет стоимости необходимого оборудования

Для работы с выбранными программами необходимо мощное оборудование, чтобы программы не зависали при вычислениях текстур и полигонов. Если нанимается 2 специалиста, которым необходимо мощное оборудование, то бухгалтер может работать на среднем, пригодного для офисной работы персональном компьютере. Стоимость представлена в таблице 3.

Таблица 2 − Стоимость оборудования

Показатель	Стоимость
Стоимость ПК для дизайнера, руб.	80000
Стоимость ПК для программиста, руб.	60000
Стоимость ПК для бухгалтера, руб.	20000

Для работы дизайнера и программиста требуется примерно одинаковой мощности персональные компьютеры. дизайнеру, возможно понадобится чуть мощнее, для создания и обработки 3Д моделей.

Нанимать и обустраивать рабочие места следует только, если в планах открыть студию разработки или иметь несколько проектов на реализацию, потому что для одного такого маленького, в масштабах игровой индустрии, проекта, можно просто нанять работников удаленно, чтобы сотрудники работали на собственных машинах. В этом случае не придется затрачиваться на покупку оборудования и аренду места для офиса.

3.3. Расчет ожидаемой прибыли

Прибыль ожидается от проданных копий готовой игры. Так как проект считается небольшим по своим меркам и имеет достаточно простой геймплей, цена не может быть очень высокой, но должна окупить затраты на производство и принести прибыль. Попробуем провести расчеты при минимальной цене для игр подобного рода – 30р.

Проект планируется выложить на продажу на популярную торговую площадку Steam, которая забирает 30% от продаж, плюс 100$ стоимость слота для размещения игры. Итоговый заработок за 3 года с предполагаемым количеством проданных копий представлен в таблице 4. Предполагаемое количество реализованных копий берется как 5% от игр-аналогов, рассмотренных в первой главе. Это является очень осторожным прогнозом.

Таблица 3 − Заработок от продаж

Показатель	1 год	2 год	3 год
Количество проданных копий, шт.	10000	10000	5000
Продажа в Steam при цене 30р за игру	300000	300000	150000
Продажа в Steam с учетом распродаж и падением цены на 30%	210000	210000	105000
Доходы от Steam после снятия комиссии площадки (25-30%)	150000	150000	75000
Итог, руб.			375000

По таблице можно увидеть, что если после запуска проекта реализация составит 5% от подобных игр, то выручка окупит все затраты и прибыль составит 100 тыс.руб.

Однако важно понимать, что успех или неудача каждой конкретной игры зависит от многих факторов. Одним из важных на сегодняшний день факторов является информационное сопровождение игры с момента старта и до выхода на пик продаж с помощью различных обзоров и демонстраций на видеосервисе youtube.com. В силу специфики работы автора, есть возможность продвигать игру с помощью различных блогеров и стримеров, что повысит шансы на успех проекта.

Выводы по третьему разделу

В разделе приведены расчеты по затратам и возможной прибыли от реализации игры. Учтены факторы снижения прибыли, такие как торговые комиссии и акционная продажа продукта. Объем реализации выбран из расчета 5% от сравниваемых аналогов.

ЗАКЛЮЧЕНИЕ

В ходе работы проведен анализ предметной области, изучены некоторые аналогичные программные продукты, сформулировано обобщенное техническое задание, спроектированы UML диаграммы и описаны программные средства, которые будут использованы при разработке игры.

Кроме того, описана программная структура проекта и представлены фрагменты кода, позволяющие понять принцип работы приложения и особенности сетевого взаимодействия внутри игры.

В экономической части проведен краткий анализ технико-экономических показателей, связанных с проектом. По осторожным оценкам проект окупаемый и прибыльный, однако прибыль не велика.

Используя возможности своей работы (игровая индустрия, продвижение игр) разработчик данного продукта может значительно повысить эффективность реализации своей игры, однако для этого целесообразно усовершенствовать игровой продукт (возможно увеличить количество игроков, возможно немного усложнить геймплей и увеличить количество юнитов).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Специальная литература

1 Акмаева, Р.И. Экономика организаций (предприятий): учебное пособие / Р.И. Акмаева, Н. Ш. Епифанова. – Рн/Д: Феникс, 2014;

2 Торн А. Основы анимации в Unity [Текст]: учебное пособие/ Алан Торн– ред.: Д. Мовчан, переводчик: Р. Рагимов –Москва: ДМК, 2016 –176с.

3 Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C# [Текст]: учебное пособие/ Джозеф Хокинг – Санкт-Петербург: Питер, 2016 – 336с.

Интернет-источники

4 Аббасов, И.Б. Основы трехмерного моделирования в графической системе 3ds Max 2018 [Электронный ресурс] : учебное пособие / И.Б. Аббасов. – Электрон. дан. – Москва : ДМК Пресс, 2017. – 186 с. – Режим доступа: https://e.lanbook.com/book/97355. – Загл. с экрана.

5 Дэвис, А. Асинхронное программирование в C# 5.0. [Электронный ресурс] – Электрон.дан. – М. : ДМК Пресс, 2013. – 120 с.

6 Паласиос, Х. Unity 5.x. Программирование искусственного интеллекта в играх [Электронный ресурс] / Х. Паласиос ; пер. с англ. Рагимова Р.Н.. – Электрон. дан. – Москва : ДМК Пресс, 2017. – 272 с.

7 Самохвалов, Э.Н. Введение в проектирование и разработку приложений на языке программирования C# [Электронный ресурс]: методические указания / Э.Н. Самохвалов, Г.И. Ревунков, Ю.Е. Гапанюк. – Электрон. дан. – Москва : МГТУ им. Н.Э. Баумана, 2018. – 244 с. – Режим доступа: https://e.lanbook.com/book/103555. – Загл. с экрана.