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

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

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2019 г.

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

Разработка игры в жанре «Симулятор выживания» по мотивам «интернет вещей» на платформе Unity

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

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

Автор работы,
студент группы КЭ-222
_____ А.В. Левкин
«___»_____2019 г.

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

Челябинск-2019

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

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

ЗАДАНИЕ

на выпускную квалификационную работу магистра

студенту группы КЭ-222
Левкин Александр Владимирович
обучающемуся по направлению
09.04.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка игры в жанре «Симулятор выживания» по мотивам «интернет вещей» на платформе Unity» утверждена приказом по университету от 25 апреля 2019 г. № 899

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

3. **Исходные данные к работе:** статьи, книги, техническое задание.
   - Хокинг, Д. Unity в действии. Мультиплатформенная разработка на C#. – СПб.: Изд-во Питер, 2016. – 336 с.;
   - Арлоу, Д. Нейштадт, А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. – М.: Изд-во Символ-Плюс, 2007. – 624 с.;
   - Официальный сайт Unity3D. – https://unity3d.com.

4. **Перечень подлежащих разработке вопросов:**
   - рассмотрение существующих игровых проектов, выполненных в подобном жанре;
   - анализ современных платформ для разработки игровых приложений;
   - разработка проекта игры «Arkady Survive»;
   - реализация всех спроектированных частей;
   - выпуск игры «Arkady Survive» в сервисе Steam.

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

Руководитель работы _____/ И.Л. Кафтанников /

Студент _____/ А.В. Левкин /

КАЛЕНДАРНЫЙ ПЛАН

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

Руководитель работы _____/ И.Л. Кафтанников /

Студент _____/ А.В. Левкин /

Аннотация

А.В. Левкин. Разработка игры в жанре «Симулятор выживания» по мотивам «интернет вещей» на платформе Unity. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 77 с., 22 ил., библиогр. список – 16 наим.

В рамках выпускной квалификационной работы производится анализ современных средств разработки компьютерных игр. Организуется проектирование и разработка игрового приложения на платформе Unity. В ходе реализации был выполнен ряд задач, связанных с созданием концепции и графической составляющей игры. Выполнена работа по созданию анимации. Производится выборка и анализ результатов работы приложения. Приложение было выпущено в онлайн-сервисе цифрового распространения компьютерных игр Steam.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

Игровая индустрия занимает значительную часть электронного бизнеса в современных компьютерных сетях. Существует множество игр различных жанров для всех видов электронных устройств: персональных компьютеров, ноутбуков, планшетов, смартфонов, приставок и т.д. В настоящее время игровая индустрия активно развивается, поэтому имеются перспективы создания новых компьютерных игр.

Цели и задачи

Целью работы является разработка игры «Arkady Survive» в смешанном жанре «симулятор выживания» и «три в ряд». Для достижения данной цели должны быть решены следующие задачи:

- провести анализ аналогичных проектов;
- провести анализ и выбор средств реализации игры;
- описать концепцию игры;
- спроектировать программную систему;
- реализовать игру;
- добавить визуальные эффекты и звуковое сопровождение;
- провести тестирование реализованной игры;
- выпустить игру в сервисе цифрового распространения компьютерных игр.

Структура и объем работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 77 страниц, список литературы содержит 16 источников.

Содержание работы

В первой главе проведен обзор аналогичных проектов и инструментов, необходимых для реализации игры.

Во второй главе описывается игровой мир, возможности персонажа и интерфейса игры.

В третьей главе для формализации функциональных требований были применены две UML-диаграммы вариантов использования, с их последующим описанием.

В четвертой главе описывается файловая структура, игровые объекты, использованные классы. Также представлен игровой процесс.

В пятой главе приведены результаты тестирования игры.

В заключении описаны результаты, полученные в ходе выполнения работы, а также рассмотрены перспективы дальнейшего развития.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор аналогичных проектов

В начале следует дать определения жанрам, к которым будет относится будущая игра, чтобы определить аналогичные проекты. Симулятор выживания — жанр компьютерных игр, разновидность симуляторов жизни, в которых основной целью игрока является сохранение жизни виртуального персонажа на фоне множества угрожающих ему опасностей. Три в ряд – жанр логических игр, характеризующийся тем, что игровой мир состоит из таблицы или сетки элементов, а задачей игрока является манипулирование элементами таким образом, чтобы совпали заданные игрой шаблонные комбинации, и после выполнения условия собранные элементы исчезают.

Bejeweled 3

Рисунок 1 — Экранная форма из игры Bejeweled 3

Bejeweled 3 — классический представитель жанра Три в ряд. Данная игра имеет 8 режимов игры, среди которых режим «молния», в котором все элементы разлетаются на множество осколков и от разрядов молний, прохождение на время, режим «квест», секретный режим, режим «Дзен», ультра медленный, в котором появление и перемещение новых элементов происходит очень медленно. Экранная форма из игры Bejeweled 3 придавлена на рисунке 1.

PuzzleQuest: Challenge of the Warlords

Рисунок 2 — Экранная форма из игры PuzzleQuest: Challenge of the Warlords

PuzzleQuest: Challenge of the Warlords — это игра в смешанном жанре «Три в ряд» и «Ролевая игра». Данная игра предлагает классическую головоломку, совмещенную с историей о борьбе добра и зла. В ней необходимо испытать свои навыки игры в «три в ряд» в битве против компьютерных оппонентов для спасения Королевства Этерия от злого лорда Бейна, создавая сочетания различных типов кристаллов, пополняя свой инвентарь заклинаний. В игре необходимо обыгрывать своих противников и пускать в ход заклинания, чтобы разбить их шансы на составление комбинаций и уменьшить их результаты. В данной игре есть возможность выигрывать схватки в бою или с помощью головоломки Три в ряд. Экранная форма из игры PuzzleQuest: Challenge of the Warlords придавлена на рисунке 2.

The Long Dark

Рисунок 3 — Экранная форма из игры The Long Dark

The Long Dark — это игра в жанре симулятора выживания, где игроку придется бороться с суровой зимней природой. Это игра, фокусирующаяся на выживании и исследовании, где игрокам придется самим принимать решения, путешествуя по огромной холодной дикой местности, пережившей геомагнитную катастрофу. В данной игре главными врагами являются лишь стужа, голод и агрессивные животные. В игре есть 3 режима: сюжетный режим, режим испытаний и бессюжетный режим «песочница». Экранная форма из игры The Long Dark придавлена на рисунке 3.

1.2. Анализ существующих платформ для разработки игр

Игровой движок - центральный программный компонент компьютерных и мобильных игр. Игровой движок – это тот модуль игры, который включает в себя игровую логику. Он значительно облегчает процесс разработки приложения за счёт экономии времени и трудовых ресурсов и даёт игре возможность запускаться на нескольких платформах. В настоящее время существует огромное количество средств разработки игр.

В ходе выполнения выпускной квалификационной работы магистра был проведен анализ наиболее популярных бесплатных или условно бесплатных средств разработки компьютерных игр. Для сравнения был выбран список из 4 самых популярных средств разработки игр, предоставляющих свою бесплатную версию пользователю. Эти платформы доступны для свободного использования. Также были отмечены особенности работы с каждой из них.

Unity3D

Рисунок 4 — Интерфейс платформы Unity

Unity 3D — это мощная среда для разработки 3D игр и приложений. Данная платформа создана в 2005 году, в Дании. Главный плюс Unity 3D это простота разработки приложений. В данной среде разрабатывается огромное количество игр под различные платформы.

Unity очень тесно сотрудничают с Oculus, Apple и рядом других компаний, имеет очень мощный магазин Unity Asset Store, в котором есть абсолютно все: от текстур и 3D моделей до готовых проектов.

Unity поставляется с полностью интегрированными физическими движками NVIDIA® PhysX® и Box2D. При этом Unity сам выбирает, какой физический движок использовать в зависимости от контекста. Одним из

несомненных преимуществ использования платформы Unity является ее подробная документация с исчерпывающим описанием всех функциональных возможностей и примеры их применения. Также на сервисе Youtube есть официальный канал Unity с руководствами по применению функций Unity и огромное количество обучающих видео, записанных самими пользователями. Интерфейс платформы Unity придавлен на рисунке 4.

Основные возможности и плюсы Unity 3D:

- доступный и понятный интерфейс;
- поддержка двух языков программирования: C# и JavaScript, на которых пишутся скрипты;
- большое сообщество;
- полная интеграция платформы со средой разработки;
- поддержка перетягивания объектов в редакторе;
- поддержка импорта большого количества форматов;
- поддержка физики ткани (PhysX Cloth);
- возможность дополнения функционала;
- возможность использования систем контроля версий.

Минусы:

- ограниченный набор инструментов;
- процесс изготовления игры отнимает много времени;
- не самая лучшая графика;
- сложность адаптации продукта, написанного на нем под конкретного пользователя.

CryEngine 3

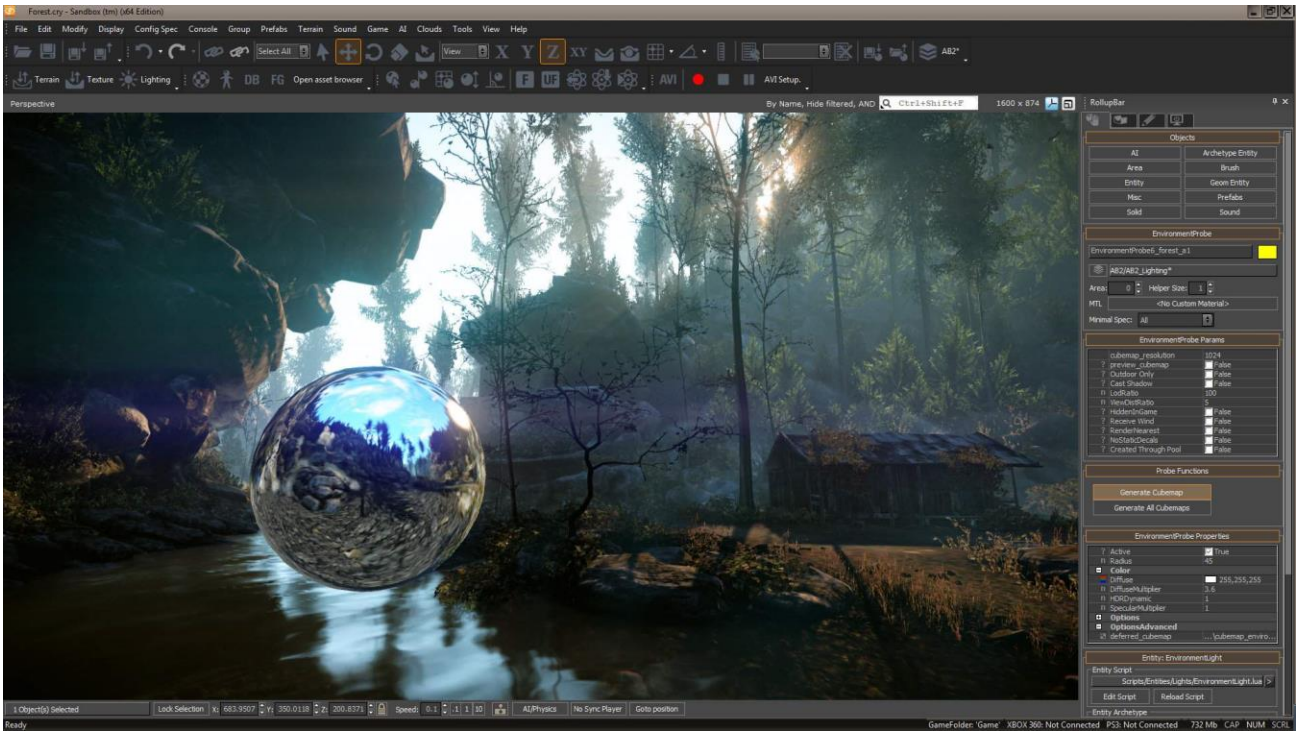

Рисунок 5 — Интерфейс платформы CryEngine 3

CryEngine 3 — среда разработки, разработанная компанией Crytek. CryEngine, в отличие от других платформ, нацелен на создание игр для PC и консолей. Crytek сделали мощную платформу, которая выдает лучшую графику из всех 4 выбранных сред разработки. В данной среде разработки имеется динамический свет, затенение, затуманивание в реальном времени и управление уровнем детализации ландшафта. Лица и персонажи в CryEngine получаются по-настоящему захватывающими. С помощью данной среды разработки были созданы такие игры как: Crysis, Far Cry, Warface, Star Citizen, Ryse: Son of Rome, Homefront: The Revolution, Evolve, Проект Армата, Kingdom Come: Deliverance.

Cry Engine 3 берет начало своей истории в 2001 году, когда была анонсирована первая разрабатываемая на нем игра Far Cry. Разработчики этой

платформы с самого начала преследовали цель не самим создавать на нем игры, а продавать его как технологию.

Существует несколько способов использования платформы третьими студиями. Первый — это для использования дома или для издания бесплатных игр, с целью изучения и без прибыли. Второй вариант - для коммерческого использования. В этом случае используется лицензия royalty only 20%. То есть необходимо отдать 20 процентов дохода. Третий - для инди и загружаемых игр. Четвертый и последний вариант лицензии - для крупных игр, стоимость которой необходимо обговаривать индивидуально, и она является коммерческой тайной. CryEngine 3 ориентирован на создание кросс-платформенных игр, предназначенных для PC и консолей. В нем изначально присутствует поддержка глобальных мультиплеерных игр. CryEngine 3 обладает большим списком технологий визуализации. Физический компонент движка CryPhysics также работает независимо от физических API, таких как PhysX. Встроенная система анимации предлагает несколько отличных подсистем: индивидуализация персонажей, параметрическая скелетная анимация, процедурное деформирование движения.

В его комплекте есть то, что необходимо для реализации AAA-класса игр, в том числе редактор уровней, который способен создавать карты в реальном времени. Интерфейс платформы CryEngine 3 придавлен на рисунке 5.

Плюсы:

- возможность создать игру с отличной графикой;
- создание мощного звукового сопровождения;
- самый простой процесс создания искусственного интеллекта в сегменте;
- начинающему разработчику будет легко сделать пользовательский интерфейс;

– разработки игр для PC, Xbox 360, PlayStation 3 с поддержкой DirectX9 и 10.

Минусы:

– слабая техническая поддержка бесплатной версии;
– поддерживаемая ОС только Microsoft Windows;
– относительно высокий порог вхождения;
– Непопулярность Cry Engine 3.

Unreal Engine

Рисунок 6 — Интерфейс платформы Unreal Engine

Unreal Engine 4 – среда разработки, созданная Epic Games. Unreal Engine 4 – самая популярная среда разработки для создания фильмов и AAA-проектов.

Данная платформа обладает высокими графическими возможностями. С Unreal Engine 4 есть возможность разрабатывать игры под PC, Mac, консоли, IOS, Android.

В отличие от Unity, UE4 имеет мощный инструмент для дизайна игровых уровней прямо в сцене, достаточно удобную систему Blueprint, не имеющую аналогов, красивый дизайн самой платформы и интуитивность в использовании. Из всех сред разработки, Unreal Engine 4 является самым инновационным. Он сочетает в себе высокую производительность, лучшую графику, простой язык программирования и удобность в использовании. Есть очень мощное сообщество, помогающее решить все проблемы. Epic Games раздает UE4 абсолютно бесплатно, весь исходный код открыт, за это разработчики просят лишь 5% с продажи одной копии. Команда Epic тесно сотрудничает с разработчиками VR технологий, а значит, что уже сейчас можно создавать игры под PlayStation VR, Oculus и т.д.

На официальном сайте Unreal Engine есть огромное количество различной документации. Также есть раздел для тех, кто перешел с Unity3d. Магазин в Unreal Engine, отстает от конкурентов. Если необходимо найти готовые 3D модели или сцены, то есть шанс, что нужные объекты не найдутся в фирменном магазине и придется их искать на различных интернет ресурсах или создавать самому. Кроме этого, если получится найти необходимые ресурсы, то придется отдать немалые деньги за возможность их загрузить, так как цены в официальном магазине достаточно высокие. Магазин Unity Asset Store в этом плане лучше, так как выбор в нем намного больше, а цены более демократичные.

На Unreal Engine 4 было сделано огромное количество AAA-проектов, таких как: Mortal Kombat X, Dead Island 2, Smite, Paragon, Infinity Blade, Gears of War, Batman: Arkham Asylum, Mass Effect. Данный движок написан на языке C++, разработан и поддерживается на протяжении 18 лет компанией Epic Games.

Созданные игры и приложения на Unreal Engine работают на различных операционных системах и платформах. Интерфейс платформы Unreal Engine придавлен на рисунке 6.

Плюсы:

- структура ценообразования проста для понимания;
- большое сообщество;
- возможно, напрямую использовать в проекте файлы с исходным кодом на C++;
- отличная техническая поддержка и механизм апдейта;
- широкий ассортимент инструментов для различных целей;
- совместим с различными платформами.

Минусы:

- сложно привыкнуть к определенным инструментам;
- небольшой выбор готовых инструментов в официальном магазине;
- относительно высокий порог вхождения.

Torque 2D/3D

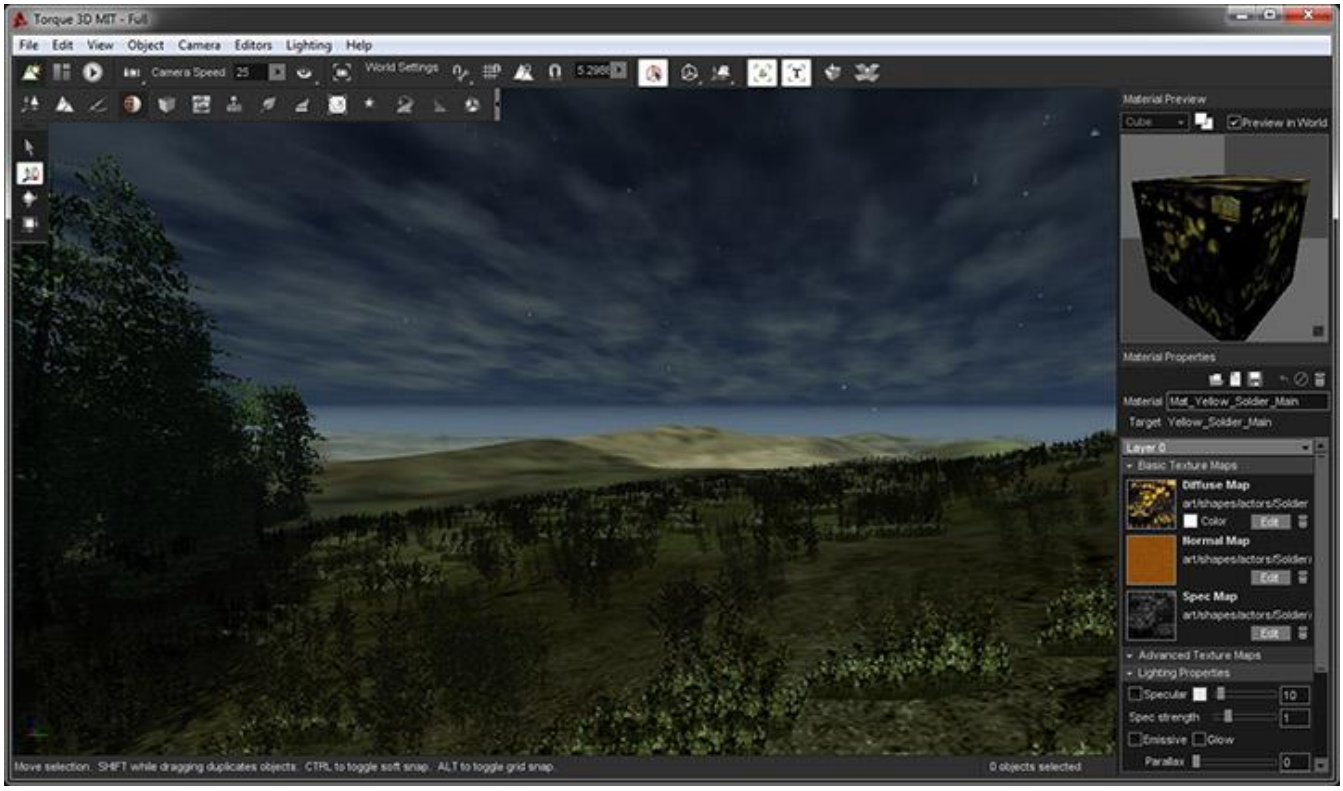

Рисунок 7 — Интерфейс платформы Torque 2D/3D

Torque 2D/3D – среда для разработки компьютерных игр с открытым исходным кодом. Был лидером в своем классе, но после развития Unity утратил свои позиции. Тем не менее до сих пор на нем разрабатывается множество успешных проектов, поскольку он активно развивается сообществом. Различия между двумерной и трехмерной версиями весьма значительны, но есть и общие элементы, например, развитая сетевая подсистема. После выхода в мир продуктов с открытым исходным кодом Torque3D сохранил и даже увеличил свои возможности, а Torque2D, напротив, многое потерял. Например, он утратил все встроенные редакторы, которые, были изъяты из-за определенных юридических соглашений. На нем можно разрабатывать игры для трех платформ: Windows, OS X и iOS.

Фундаментальные различия 2D- и 3D-версий заключаются в графической подсистеме: T2D для визуализации использует OpenGL, а T3D DirectX. В качестве скриптового языка в T2D, как и в T3D, используется Torque Script. Вместе с тем в T2D для описания игровых элементов служит XML-подобный язык TAML. Он позволяет определить свойства объектов на стадии инициализации уровня игры. Для воспроизведения звуков T2D использует библиотеку OpenAL. Симуляция физики осуществляется посредством движка B-x2D, ставшего стандартом в двумерных физических исчислениях. Вместе с T2D поставляется коллекция из огромного количества комплектов.

Сравнение и выбор среды разработки

При анализе средств разработки основными критериями были:

- новизна. – даты рассматриваемых релизов и их практическое соответствие своему времени выхода;
- порог вхождения –оценка необходимых знаний для работы с выбранной платформой;
- исходный код;
- поддерживаемые платформы и используемые языки;
- практическая ценность результатов – произведено сравнение визуальных характеристик рассматриваемых типов;
- цена – хотя мы рассматриваем условно бесплатные среды разработки, у каждого есть полноценная коммерческая версия, цена на которую тоже имеет важное значение.

Исходя их данных параметров больше всего подходят платформы Unity и Unreal Engine.

Продолжим сравнение между данными средами разработки. Представленные платформы схожи по функционалу и возможностям: у обеих

существуют собственные магазины, где доступны: игровой контент, набор объектов, документация, примеры проектов, руководства и демо. Обе платформы бесплатны и позволяют создавать игры под портативные устройства. Кроме этого, обе платформы имеют хорошую документацию. Однако среда разработки Unity зарекомендовала себя как среда разработки с менее сложным языком проектирования. Таким образом, выбор остановлен на игровой платформе Unity.

1.3. Вывод

В ходе анализа предметной области были рассмотрены игровые проекты в жанре «симулятор выживания» и «Три в ряд», их ключевые особенности и механики. Были рассмотрены средства для полного цикла разработки игры. Из рассмотренных платформ особенно выделились Unity и Unreal Engine 4, так как они понятны для использования, в них схожие возможности и они бесплатны, что очень важно для начинающих программистов. Остальные платформы не подошли так как они уступали Unity и Unreal Engine 4 по сравниваемым параметрам. В итоге, в качестве платформы для разработки игры будет использоваться Unity.

2. ПОСТАНОВКА ЗАДАЧИ

Для того чтобы перейти к этапу проектирования будущей игры необходимо сформулировать, каким должен быть конечный продукт. Для этого создадим эскизный проект игры и макеты интерфейса игры.

2.1. Концепция игры

Действие игры разворачивается в квартире молодого социофоба - Аркаши, решившего полностью абстрагироваться от внешнего мира и перейти на использование умного дома. На протяжении длительного времени ему удавалось успешно избегать контактов с внешним миром, пока в один прекрасный день все приборы не стали выходить из строя, и спокойная, размеренная жизнь Аркаши не оказалась под угрозой. Все это усложняется тем, что в стране каждый день начали принимать новые законы, которые усложняют и без того не легкую жизнь Аркаши.

Игра будет представлять собой изображения комнат в квартире с умными приборами, с которыми можно взаимодействовать. Взаимодействие с приборами необходимо для пополнения параметров выживания, починки и модернизации умных предметов.

Перемещение между комнатами будет реализовано с помощью на экранных кнопок.

Параметры выживания:

- деньги – тратятся на оплату коммунальных услуг и еду;
- еда – расход калорий на выполнение различный действий;
- кофе – Аркаша кофеман и не может жить без кофе;
- гигиена – Аркаша мизофоб;
- сон – расход энергии на выполнение различный действий.

Игра будет проиграна в случае, если какой-то из этих параметров опустится до 0.

Устройства умного дома:

– умный холодильник – закупает продукты (Аркаша сам не может это сделать так как он социофоб);
– умная ванна(душ) – включает и настраивает воду (Аркаша не может это сделать сам, так как он мизофоб);
– умная кофеварка – варит кофе;
– умная мультиварка – готовит еду;
– умный компьютер – сам зарабатывает деньги;
– умная кровать – возможность поспать есть только в определенное время (в другое время из кровати торчат шипы).

Эти устройства будут выходить из строя и их нужно чинить, иначе не будет возможности восстановить какой-нибудь параметр выживания. Настройка будет происходить в игре три в ряд. Вместо количества ходов будут отниматься параметры – еда, кофе, сон, гигиена. Во время игры можно попытаться закончить настройку раньше, тем самым потратив меньше энергии, но при этом шанс успешной настройки будет прямо пропорционален тому, сколько процентов настройки было завершено.

В игре три в ряд элементами выступают предметы, которые каким-либо образом связаны с умным устройством. Кроме того, будет изображен само умное устройство, которое чинится в процессе игры.

Увеличение сложности по мере прохождения:

– увеличение квартплаты;
– увеличение цен на продукты;
– увеличение уменьшение зарабатываемых денег;

– увеличение количества очков, необходимых для завершения починки устройства.

Данные усложнения постепенно вступают в силу по мере прохождения, из-за чего каждый следующий день прожить все сложнее, тем самым усложняя задачу игроку.

Цель игры заключается в том, что нужно прожить как можно больше дней.

Игра будет однопользовательской, с возможностью играть только на одном устройстве.

2.2. Концепция интерфейса

Интерфейс пользователя – разновидность интерфейсов, в котором одна сторона представлена человеком (пользователем), другая – машиной/устройством. Представляет собой совокупность средств и методов, при помощи которых пользователь взаимодействует с различными, чаще всего сложными, машинами, устройствами и аппаратурой.

После загрузки игры, пользователь должен иметь доступ в меню: старт игры, загрузку игры, настроек, информации о игре. Также пользователь должен иметь возможность выйти из игры и вернуться в главное меню.

Было создано визуальное представление о будущем интерфейсе игры путем схематичной прорисовки макетов двух главных игровых сцен – комната квартиры и игрового поля игры три в ряд.

В сцене комнаты квартиры планируется реализовать следующие элементы:

– кнопку перехода в другую комнату;
– изображение умного устройства;
– текущие состояние параметров выживания;
– кнопку вызова меню.

На рисунке 8 представлено схематичное изображение создаваемой комнаты с этими элементами.

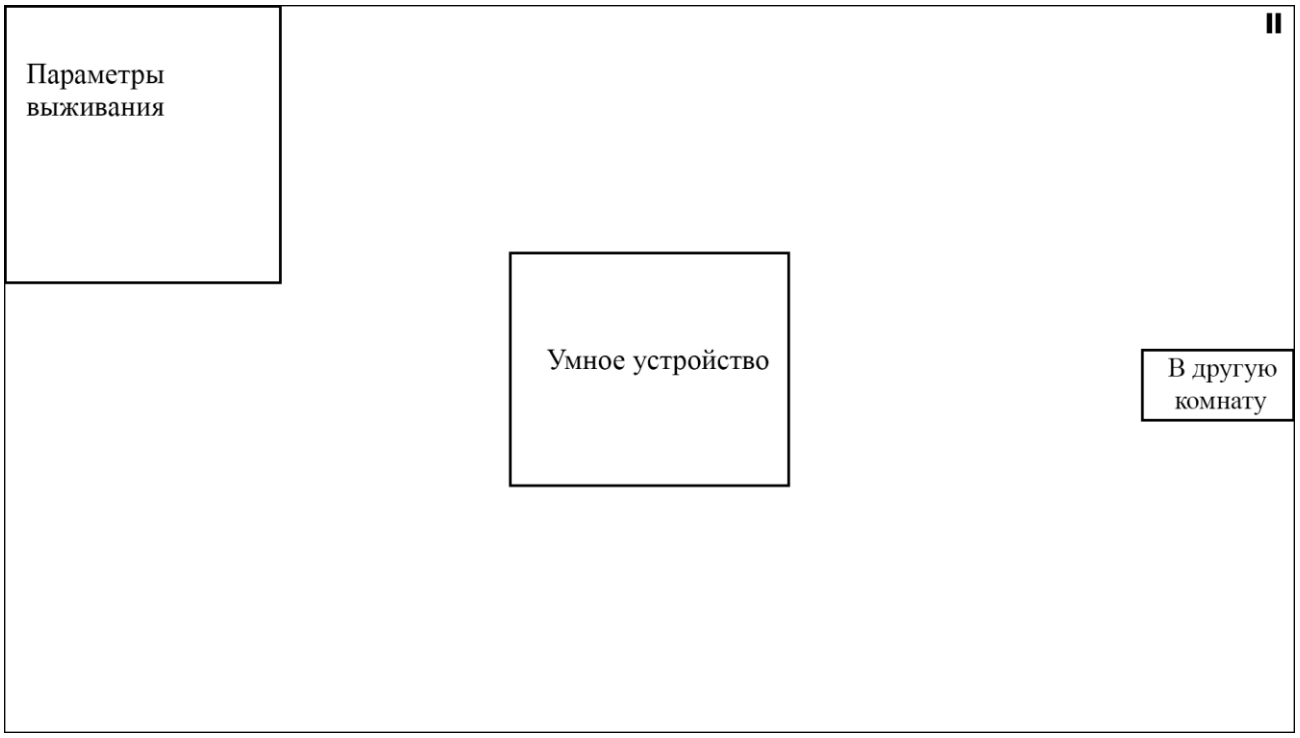

Рисунок 8 – Макет игровой комнаты

В сцене поля игры три в ряд планируется реализовать следующие элементы:

– полосу прогресса;
– изображение умного устройства;
– текущие состояние параметров выживания;
– поле игры;
– кнопку вызова меню.

На рисунке 9 представлено схематичное изображение создаваемой комнаты с этими элементами.

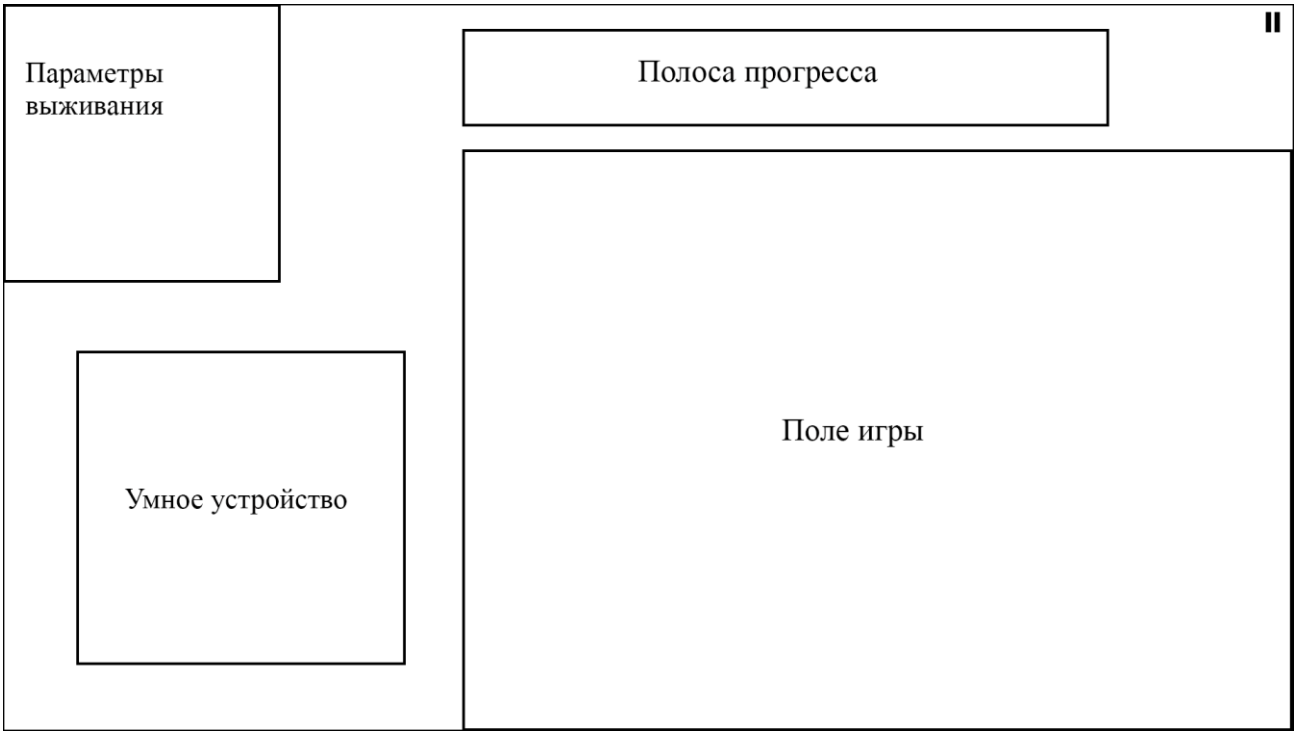

Рисунок 9 – Макет поля игры три в ряд

2.3. Вывод

В процессе постановки задачи была составлена концепция игры, которую необходимо разработать. Определены ключевые особенности поведения главного персонажа и основные объекты на протяжении всей игры. В разделе концепция интерфейса описаны игровые сцены, в которые может переходить пользователь игры, а также то, в каком формате должно все выглядеть.

3. ПРОЕКТИРОВАНИЕ

Проектирование – процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект – целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для последующей реализации.

Для того чтобы приступить к созданию игры определим требования к игре и проанализируем возможные варианты исполнения.

Также проанализируем программную часть создаваемой игры, создав диаграмму прецедентов приложения и определив составляющие каждого из компонентов

3.1. Определение требований

В ходе проектирования приложения были определены функциональные возможности, предоставляемые пользователю, а также нефункциональные требования к разрабатываемому приложению. При загрузке приложения пользователь должен иметь возможность начать новую игру, а также продолжить ее, если играет не в первый раз. Пользователь должен иметь возможность изменить громкость музыки и звуков в игре, а также разрешение экрана. Все перечисленные функции по настройке параметров должны предоставляться как в главном меню, так и в режиме игры. Пользователь должен иметь возможность перемещаться между игровыми комнатами квартиры, читать диалоги главного героя, выбирать умные устройства и взаимодействовать с ними. Играть в игру «Три в ряд», тем самым чинить и модернизировать умные устройства. Игра должна усложняться с каждым днем, различными методами, уменьшающими вириальные денежные средства игрока. Пользователь должен иметь возможность сохраняться только в определённом месте и при определенных условиях, чтобы

предотвратить злоупотребление загрузкой игры для получения преимуществ. Приложение должно функционировать на ОС Windows 7/8/8.1/10.

3.2. Диаграмма прецедентов приложения

Диаграмма прецедентов UML, отражающая отношения между актёрами и прецедентами, и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне [5].

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Сформируем модель вариантов использования разрабатываемой системы.

Основным актером, взаимодействующим с системой, является «Игрок», который взаимодействует с игрой. Данный актер, находясь в игровой комнате может выбирать устройства для взаимодействия, переходить в другую комнату, заходить в меню настроек.

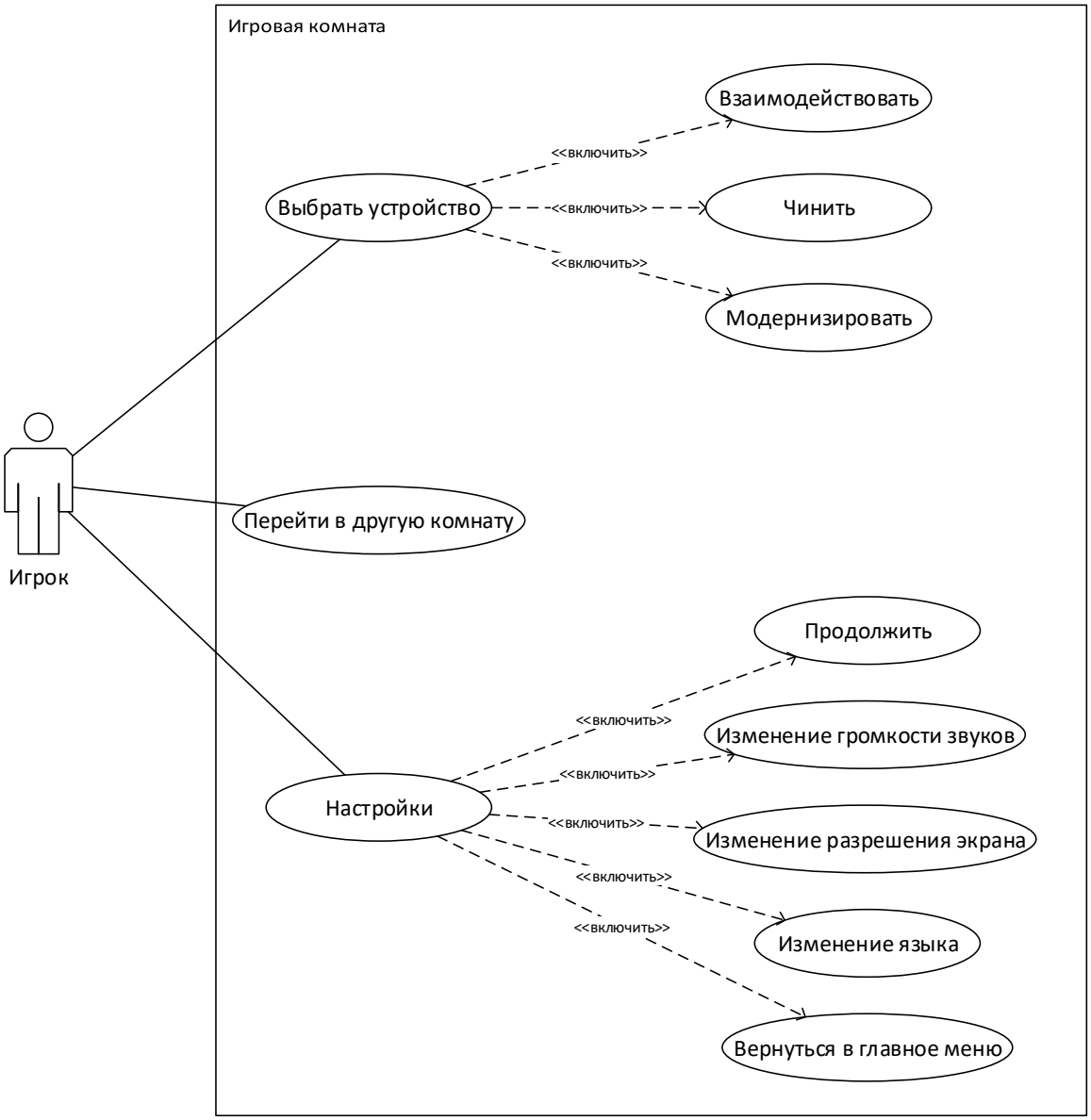

Рисунок 10 – Диаграмма прецедентов игровая комната

На основе данных потребностей выделены следующие варианты использования:

- выбрать устройство;
- перейти в другую комнату;
- взаимодействовать;
- чинить;

– модернизировать;
– настройки;
– продолжить;
– изменение громкости звуков;
– изменение разрешения экрана;
– изменение языка;
– вернуться в главное меню.

Рассмотрим более подробно каждый из этих прецедентов.

Прецедент «выбрать устройство»

Краткое описание

Позволяет игроку выбирать умное устройство для взаимодействия. Этот вариант использования начинается, когда игрок выбирает определение устройство в комнате.

Прецедент «Перейти в другую комнату»

Краткое описание

Позволяет игроку перейти в другую комнату, с другими умными устройствами. Этот вариант использования начинается, когда игрок перемещается между комнатами.

Прецедент «Взаимодействовать»

Краткое описание

Позволяет игроку взаимодействовать с умными устройствами. Вид взаимодействия определяется выбранным устройством. Этот вариант использования начинается, после того как выбрано устройство.

Прецедент «Чинить»

Краткое описание

Позволяет игроку чинить выбранное умное устройство. Запускает игру «Три в ряд». Этот вариант использования начинается, когда выбранное устройство сломано.

Прецедент «Модернизировать»

Краткое описание

Позволяет игроку модернизировать выбранное умное устройство. Запускает игру «Три в ряд». Этот вариант использования начинается, когда выбранное устройство работает.

Прецедент «Настройки»

Краткое описание

Позволяет игроку перейти в меню настроек. Этот вариант использования начинается, когда игрок нажимает на кнопку паузы.

Прецедент «Продолжить»

Краткое описание

Позволяет игроку выйти из меню настроек и продолжить игру. Этот вариант использования начинается, когда игрок нажимает кнопку «Продолжить».

Прецедент «Изменение громкости звуков»

Краткое описание

Позволяет игроку изменять громкость звуков и музыки в игре. Этот вариант использования начинается, когда игрок меняет громкость звуков или музыки.

Прецедент «Изменение разрешения экрана»

Краткое описание

Позволяет игроку изменять разрешение экрана в игре. Этот вариант использования начинается, когда игрок меняет разрешение экрана.

Прецедент «Изменение разрешения экрана»

Краткое описание

Позволяет игроку менять язык интерфейса и диалогов в игре. Этот вариант использования начинается, когда игрок меняет язык.

Прецедент «Вернуться в главное меню»

Краткое описание

Позволяет игроку выйти из игры в главное меню. Этот вариант использования начинается, когда игрок нажимает кнопку «В главное меню».

Далее рассмотрим модель вариантов использования в игре «Три в ряд». Игрок, находясь в игре может перемещать элементы и заходить в меню настроек.

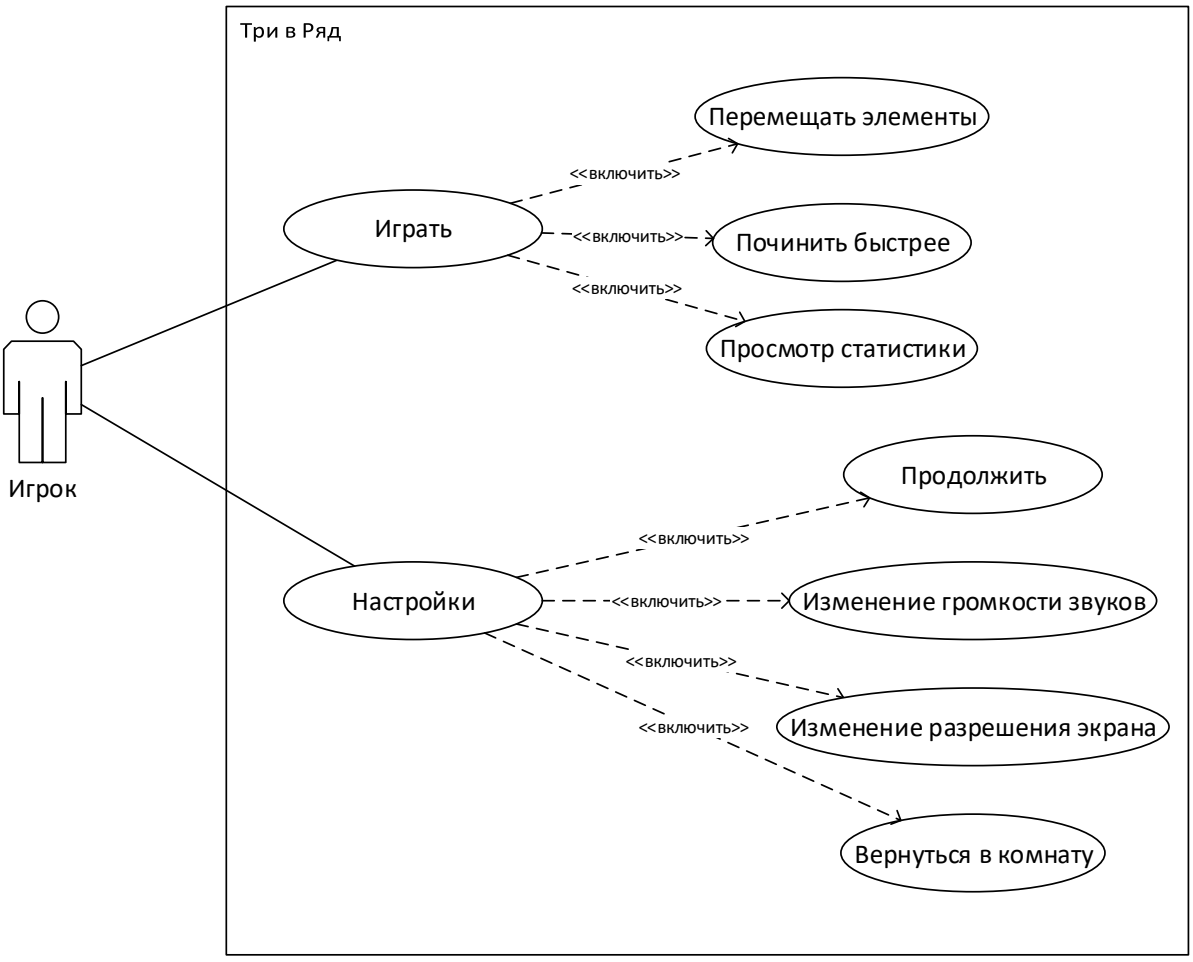

Рисунок 11 – Диаграмма прецедентов три в ряд

На основе данных потребностей выделены следующие варианты использования:

- играть;
- перемещать элементы;
- починить быстрее;
- просмотр статистики;
- настройки;
- продолжить;
- изменение громкости звуков;
- изменение разрешения экрана;
- вернуться в главное меню.

Рассмотрим более подробно каждый из этих прецедентов.

Прецедент «Играть»

Краткое описание

Позволяет игроку играть в игру «Три в ряд». Этот вариант использования начинается, когда игрок чинит или модернизирует умное устройство.

Прецедент «Перемещать элементы»

Краткое описание

Позволяет игроку перемещать элементы на игровом поле. Этот вариант использования начинается, когда перемещает элементы, составляя три или более одинаковых элементов в ряд.

Прецедент «Починить быстрее»

Краткое описание

Позволяет игроку завершить уровень раньше с определенным шансом. Этот вариант использования начинается, когда игрок нажимает на кнопку «завершить с шансом».

Прецедент «Просмотр статистики»

Краткое описание

Позволяет игроку просматривать после игровую статистику. Этот вариант использования начинается, когда игрок проигрывает и ему выводится окно статистики.

3.3. Вывод

В ходе проектирования программной системы были определены требования к игре и разработаны две UML-диаграммы вариантов использования в дух главных сценах: игровая комната и игра «Три в ряд». Так были определены основные действия, требуемые от разрабатываемой игры.

4. РЕАЛИЗАЦИЯ

Для разработки игрового приложения была выбрана платформа Unity. Данная платформа имеет низкий порог вхождения, большое количество обучающих материалов и сообщество разработчиков, в следствии чего с ней можно быстрее начать работать.

Для создания и редактирования графической составляющей игры использовался графический редактор Adobe Photoshop.

Для создания анимации и переходов использовались программа для создания анимации Spine [4] и встроенные инструменты Unity [6].

Рассмотрим подробно все составляющие реализованного приложения: игровые объекты, игровые сценарии, выполненные на языке программирования C# и итоговый вид интерфейса приложения.

4.1. Файловая структура игры

Разработанный проект содержит каталоги, в которых хранятся:

- анимации объектов;
- музыка;
- шрифты;
- шаблоны объектов;
- игровую сцену;
- скрипты;
- изображения.

Файловая структура представлена на рисунке 12.

В папке Animations находятся анимации всех объектов игры. В папке Audio находятся вся музыка и звуки для игры. В директории Prefabs находятся готовые шаблоны игровых объектов, при помощи которых можно составить игровой

уровень после его проектирования. Директория Scene содержит сцену игры, в которой происходят все действия. В папке Scripts находятся скрипты с описанием всех классов и взаимодействий.

В каталоге Sprites содержатся шесть подкаталогов:

– Background в себе все фоновые изображения;

– Animations содержит в себе спрайты необходимые для анимаций;

– Sclots содержит в себе элементы необходимые для формирования поля игры «Три в ряд»;

– UI содержит в себе элементы для графического интерфейса;

– Arts содержит в себе изображения главного героя и всех умных устройств;

– Chips содержит в себе все элементы всех умных устройств игры «Три в ряд».

В каталоге Scripts содержатся четыре подкаталога:

– Animations содержит в себе скрипты, управляющие анимациями в игре;

– Assistants содержит в себе скрипты-помощники, которые управляют логикой всей игры;

– UI содержит в себе скрипты, управляющие графическим интерфейсом;

– Chip содержит в себе скрипты управляющие игрой «Три в ряд».

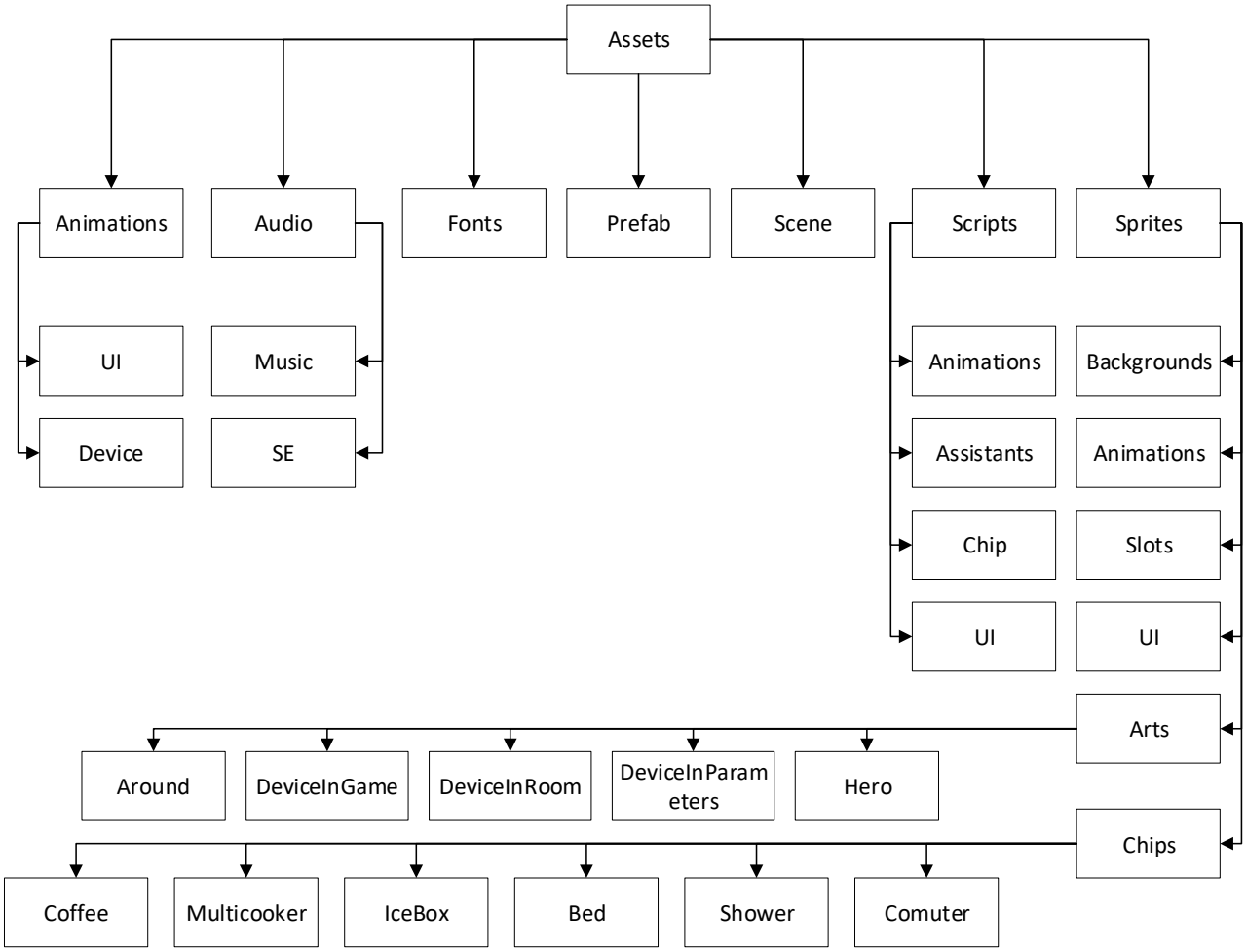

Рисунок 12 – Файловая структура

4.2. Игровые объекты

Все игровые объекты представлены в виде префабов, и включают в себя сопутствующие им изображения, скрипты, физические свойства.

Изображения, используемые для игровых объектов, были куплены в стоке изображений Adobe Stock. Редактировались в графическом редакторе Adobe Photoshop.

Главный герой

Главный герой был нарисован в графическом редакторе Adobe Illustrator[3].

Рисунок 13 – Изображение главного героя

Для игрока заданы следующие характеристики:

– изображения эмоций радости, злости, грусти и спокойное состояние;
– возможность перемещения;
– параметры выживания;
– гибель при обнулении любого из параметров выживания.

Игровые комнаты

Для оформления заднего плана игры и создания визуального отличия игровых уровней добавлены фона игровых комнат, который отображается в зависимости от того в какой комнате находится игрок. На рисунке 14 приведена одна из игровых комнат.

Рисунок 14 – Игровой фон

Умные устройства

В каждой комнате присутствует от 1 до 3 умных устройств, с которыми можно взаимодействовать. На рисунке 15 и 16 изображено работающее и сломанное устройства.

Рисунок 15 – Рабочее устройство

Рисунок 16 – Сломанное устройство

Уровни игры «Три в ряд»

Уровни представляют собой игровое поле с различными начальными размерами и элементами, с которыми нужно взаимодействовать и изображения умного устройства, которое нужно починить. В процессе прохождения уровня изображения умного устройства меняется. Исходный код класса, отвечающего за логику игры приведен в листинге А.1. приложения А. Исходный код всех разработанных скриптов добавлен в сервис GitHub [16].

С целью разнообразия, для каждого устройства имеется по 4 уровня игры и свой набор элементов.

Рисунок 17 – Уровень игры в «Три в ряд»

Элементы игры «Три в ряд»

Для создания визуального отличия игровых уровней на игровую сцену добавлены различные элементы, для каждого умного устройства. Пример

элемента в виде простого элемента, ледяной бомбы и огненной бомбы изображен на рисунке 18.

Рисунок 18 – Пример элемента

4.3. Используемые классы

Для реализации приложения были использованы скрипты, написанные на языке программирования C#.

Всего в данной игре используется 28 скриптов (классов): Animations, DestroyAnimation, AnimationAssistant, AudioAssistant, ControlAssistant, ContentAssistant, FieldAssistant, ProfileAssistant, MatchThree, UIAssistant, Chip, FireBomb, IceBomb, SimpleChip, Slot, SlotGenerator, ScoreBar, NameLabel, ScoreBarStar, ButtonSound, LevelLoad, Level, ChangeBGAndArts, Settings, Parametrs, StatisticMenu, Novella, Localization.

Рассмотрим предназначение каждого из них.

Класс Animations – класс, содержаний все анимации для объектов.

Класс DestroyAnimation – класс, отвечающий уничтожение некоторых объектов после воспроизведение анимаций.

Класс AnimationAssistant – класс, отвечающий за то, логику воспроизведения определенной анимации в зависимости от происходящего на экране.

Класс AudioAssistant – класс, отвечающий за воспроизведение музыки и звуков в игре.

Класс ControlAssistant – класс, описывающий алгоритм поведения игры в зависимости от текущих действий игрока.

Класс ContentAssistant – класс, содержащий в себе все объекты в игре.

Класс FieldAssistant – класс, отвечающий за логику создания игрового поля игры «Три в ряд».

Класс ProfileAssistant – класс, содержащий информацию о сохранённом состоянии всех устройств и параметров игрока.

Класс MatchThree – класс, отвечающий за логику игры «Три в ряд».

Класс UIAssistant – класс, отвечающий за отображение графического интерфейса.

Класс Chip – класс, определяющий текущий объект как игровой элемент игры «Три в ряд» и содержащий свойства этих элементов.

Класс FireBomb – класс, отвечающий за поведение элемента как огненной бомбы.

Класс IceBomb – класс, отвечающий за поведение элемента как ледяной бомбы.

Класс SimpleChip – класс, отвечающий за поведение элемента как обычного элемента.

Класс Slot – класс, содержащий все элементы, необходимые для создания игрового поля.

Класс ScoreBar – класс, отвечающий за отображение информации в полосе прогресса.

Класс NameLabel – класс, отвечающий отображение информации в всплывающих окнах, в зависимости от текущих параметров игрока.

Класс ScoreBarStar – класс, отвечающий за отображение звезд в полосе прогресса.

Класс ButtonSound – класс, воспроизводящий звук, при нажатии на любую кнопку.

Класс Level – класс, содержащий информацию о всех уровнях игры «Три в ряд»».

Класс LevelLoad – класс, отвечающий за загрузку определенного уровня игры, в зависимости от выбранного устройства.

Класс ChangeBGAndArts – класс, отвечающий за изменение заднего фона и устройств в комнатах и игре «Три в ряд».

Класс Settings – класс, отвечающий за настройки в игре.

Класс Parametrs – класс, содержащий текущую информацию о параметрах игрока и состояниях предметов и изменяющий данную информацию в зависимости от действий пользователя.

Класс StatisticMenu – класс, отвечающий за вывод статистики после проигрыша в игре.

Класс Novella – класс, отвечающий за логику работы диалогов в игре.

Класс Localization – класс, отвечающий за локализацию теста и определяющий какой вариант текста отображать, в зависимости от выбранного языка.

4.4. Игровой процесс

Проект состоит из сцены – файл, содержащий игровой мир со всеми объектами, сценарием, и настройками.

Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. У любого объекта на сцене обязательно присутствует компонент Transform – он хранит в себе координаты местоположения, поворота

и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент Mesh Renderer, делающий модель объекта видимой.

На данный момент игра состоит из главного меню, 5 игровых комнат, 24 уровня игры «Три в ряд» и 10 всплывающих окон. При запуске игрового приложения пользователь видит экран главного меню, в котором доступны 3 кнопки: новая игра, настройки и выход. Реализация представлена на рисунке 19.

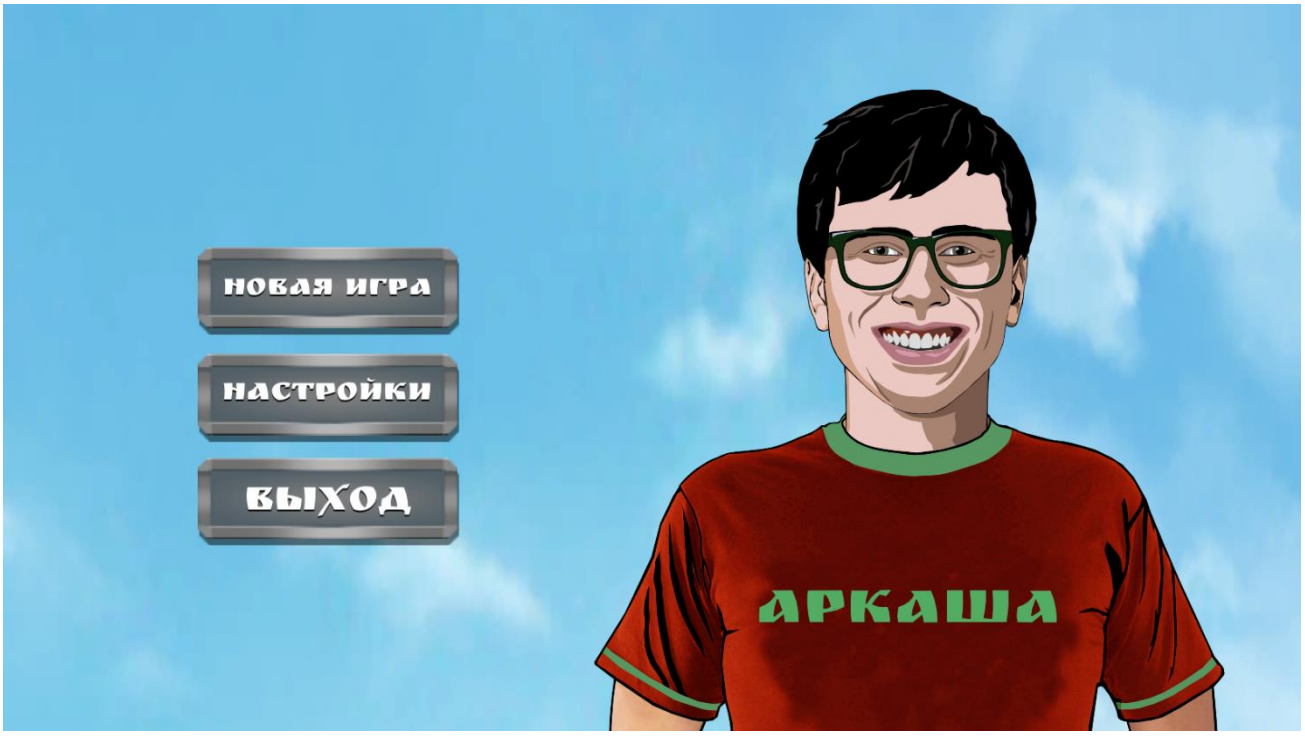

Рисунок 19 – Главное меню

При запуске новой игры отображается начальный диалог главного героя, предыстория, объясняющая идею игры.

Далее игрок появляется в игровой комнате и проходит обучение, в котором объясняются основные принципы игры. После прохождения обучения игрок предоставлен сам себе, и его главная цель в игре прожить как можно больше дней.

Рисунок 20 – Игровая комната

При выборе работающего умного устройства игрок может взаимодействовать с устройством. Взаимодействие отличается в зависимости от выбранного устройства.

Холодильник:

– пополнить количество продуктов питания;
– пополнить количество кофе;
– просмотр остатка продуктов питания и кофе.

Кофеварка:

Приготовить и выпить кофе, пополняющие параметр «кофе». На это действие расходуется кофе из холодильника.

Мультиварка:

Приготовить и покушать, пополняющие параметр «еда». На это действие расходуются продукты из холодильника.

Кровать:

– сон, восстанавливающий параметр «энергия»;
– сохранение игры.

Душ:

Помыться, восстанавливающий параметр «Гигиена».

Компьютер:

Просмотр текущих цен на продукты и квартплату.

Кроме того, компьютер в автоматическом режиме генерирует параметр «деньги», пока он находится в работающем состоянии.

После взаимодействия с любым устройством главный герой комментирует текущую ситуацию.

Если устройство сломано, то игроку необходимо починить устройство, чтобы взаимодействовать с ним. При этом игрок переходит в игру «Три в ряд».

Рисунок 21 – Игра «Три в ряд»

Если игрок проигрывает в игре, то ему выводится после игровая статистика.

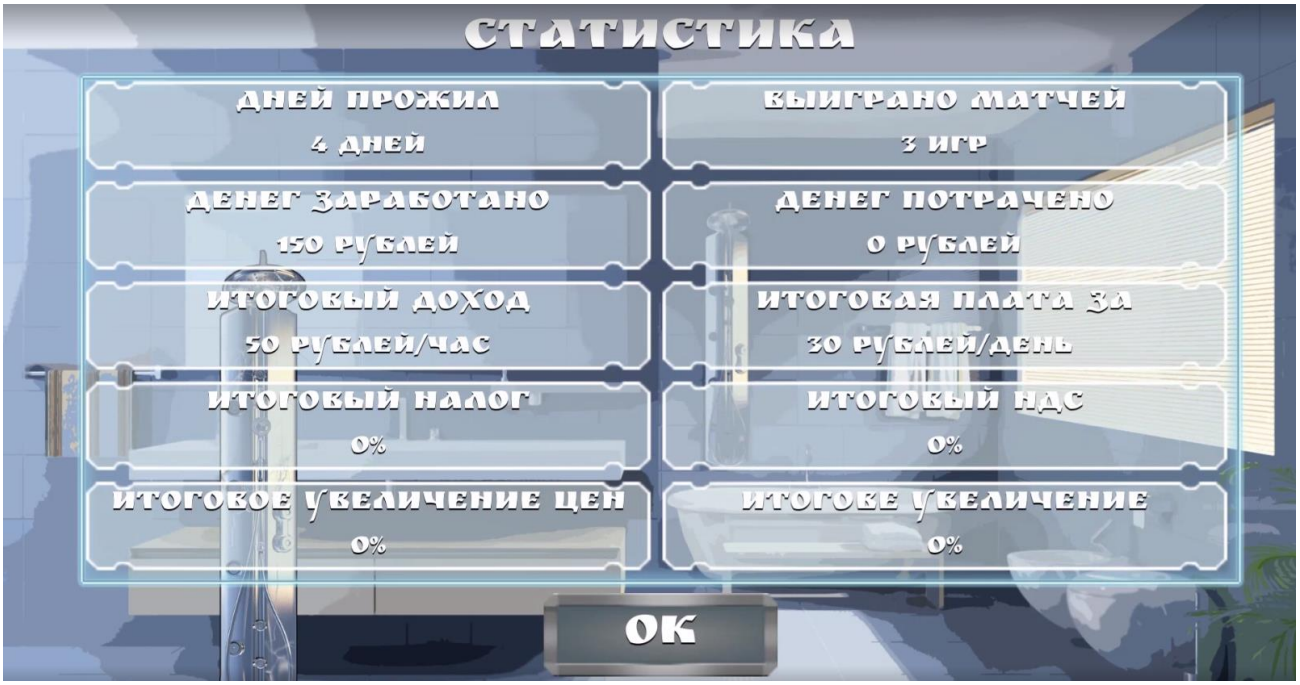

Рисунок 22 – Статистика

4.5. Вывод

Спроектированная ранее игра и все ее компоненты были реализованы. Выполнена возможность управления главным персонажем, который способен передвигаться между комнатами, выбирать, взаимодействовать и чинить умные устройства, играя в игру «Три в ряд», а также был разработан дизайн игры. Полный цикл работ проводился при помощи выбранных инструментов. Для проверки корректности работы игры, необходимо провести тестирование.

5. ТЕСТИРОВАНИЕ

5.1. Функциональное тестирование

Для тестирования приложения использовался метод функционального тестирования.

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных условиях решать задачи, нужные пользователям [10].

Результаты тестирования:

Тест № 1. Навигация в игре.

Входные данные: пользователь игры находится в произвольной сцене.

Ожидаемый результат: при нажатии кнопки паузы, всплывает окно меню.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 2. Выход из игры.

Входные данные: пользователь игры находится в произвольной сцене.

Ожидаемый результат: пользователь нажимает кнопку выхода из игры, и она немедленно закрывается.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 3. Управление персонажем.

Входные данные: пользователь нажимает на кнопки перехода между комнатами.

Ожидаемый результат: Персонаж перемещается в соответствующую комнату.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 4. Использование устройства.

Входные данные: пользователь нажимает на устройство и использует его.

Ожидаемый результат: у главного героя увеличивается определенный параметр.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 5. Уровни три в ряд.

Входные данные: пользователь нажимает на кнопку починки устройства.

Ожидаемый результат: запускается определенный уровень, связанный с выбранным устройством.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 6. Перемещение элементов в игре.

Входные данные: пользователь перемещает элементы по игровому полю.

Ожидаемый результат: если в ряд встали 3 или более элементов, то они исчезают, прибавляются очки прохождения уровня, а параметры выживания уменьшаются.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 7. Загрузка игры.

Входные данные: пользователь загружает ранее сохранению игру.

Ожидаемый результат: загружается игра и пользователь может продолжать игру с сохранённого момента, с восстановлением своего прогресса в игре.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 8. Проигрыш в игре.

Входные данные: у главного героя уменьшился любой параметр до 0.

Ожидаемый результат: всплывает окно, информирующее о том, что пользователь проиграл, без возможности продолжить текущую игру.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

Тест № 9. Статистика.

Входные данные: пользователь проиграл в игре.

Ожидаемый результат: выводится статистика, соответствующая тому, что было сделано пользователем в игре.

Полученный результат: совпадает с ожидаемым.

Тест успешно пройден.

5.2. Вывод

При помощи набора тестов, основанных на функциональных требованиях, было проведено функциональное тестирование разработанной игры. Все тесты из этого набора были успешно пройдены.

6. ЗАКЛЮЧЕНИЕ

Основные результаты

В рамках данной работы была спроектирована и реализована компьютерная игра «Arkady Survive» в жанре «симулятор выживания» на платформе Unity. Для достижения этой цели было необходимо выполнить следующие задачи:

- провести анализ аналогичных проектов;
- провести анализ и выбор средств реализации игры;
- описать концепцию игры;
- спроектировать программную систему;
- реализовать игру;
- добавить визуальные эффекты и звуковое сопровождение;
- провести тестирование реализованной игры;
- выпустить игру в сервисе цифрового распространения компьютерных игр.

В ходе реализации потребовалось выполнить ряд задач, связанных с созданием концепции, функциональной и графической составляющей игры. К таким задачам относится дизайн главного меню игры, главного персонажа, предметов, игровых комнат, поля игры три в ряд. Написаны скрипты, отвечающие за функционирование игры.

Выполнена работа по созданию и внедрению уровней в игре три в ряд, монологов главного героя, звукового сопровождения и множества различных визуальных эффектов – анимаций.

Игра «Arkady Survive» была выпущена в сервисе цифрового распространения Steam 25 мая 2019 года [2]. Кроме этого, игра была переведена на английский язык для привлечения англоязычной аудитории.

В результате выпускной квалификационной работы была разработана игра, в которой пользователь играет за главного персонажа с возможностью перемещения по игровым комнатам, взаимодействия с устройствами и игрой в три в ряд.

Пути совершенствования

Дальнейшее развитие данного игрового приложения может идти по нескольким направлениям, таким как:

- добавление новых умных устройств;
- добавление уровней сложности;
- добавление перевода на другие языки;
- выпуск игры в сервисе Google Play.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Unity Manual. – https://docs.unity3d.com. Дата последнего обращения: 19.05.2019.

2. Steamworks – https://partner.steamgames.com. Дата последнего обращения: 31.05.2019.

3. Официальный сайт Adobe. – https://www.adobe.com. Дата последнего обращения: 20.05.2019.

4. Spine. – http://ninniah.ru/spine.html. Дата обращения: 23.05.2019.

5. Арлоу Д., и Нейштадт А. UML 2 и Унифицированный процесс 2-е издание. – Пер. с англ. – СПб: Символ Плюс, 2007. – 624 с.

6. Игровая анимация и Unity. – https://m.pikabu.ru/story/igrovaya_animatsiya_i_unity_5229367. Дата последнего обращения: 24.03.2019.

7. Как правильно тестировать арт. – http://app2top.ru/marketing/kak-pravil-no-testirovat-art-99211.html. Дата обращения: 12.04.2019.

8. Какой игровой движок выбрать? – https://3dpapa.ru/what-game-engine-to-choose. Дата последнего обращения 15.12.2018).

9. Леоненков А.В. Самоучитель UML 2. – Санкт-Петербург: Изд-во БХВ — Петербург, 2007. – 576 с.

10. Майерс Г., Баджет Т. Искусство тестирования программ. – Москва: Изд-во Вильямс, 2012. – 272 с.

11. Официальный сайт Unity3D. – https://unity3d.com. Дата обращения: 19.12.2019.

12. Рассел Д. Интерфейс пользователя. – Пер. с англ. – Москва: Изд-во Книга по требованию, 2012. – 72 с.

13. Соловьев С.В., Цой Р.И., Гринкруг Л.С. Технология разработки прикладного программного обеспечения. – Москва: Изд-во российской академии естествознания, 2011. – 208 с.

14. Статья об истории развитии компьютерных игр. – http://cpu3d.com/histgame/statya-ob-istorii-razvitiya-kompyuternyh. Дата обращения: 23.05.2019.

15. Язев Ю. Обзор самых популярных движков для разработки игр. – https://xakep.ru/2014/09/05/game-development-engines-review. Дата обращения: 23.05.2019.

16. Исходный код разработанной игры «Arkady Survive». – https://github.com/Alexandrus17/GameDiplom. Дата обращения: 11.06.2019.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД MATCHTHREE СКРИПТА

Листинг А.1 – Исходный код класса MatchThree

```
// Класс, отвечающий за логику игры «три в ряд»
public class MatchThree : MonoBehaviour
{
   public static MatchThree main;
   public bool squareCombination = true;
   public List<Combinations> combinations = new List<Combinations>();
   public List<ChipInfo> chipInfos = new List<ChipInfo>();
   public List<BlockInfo> blockInfos = new List<BlockInfo>();
   public List<Mix> mixes = new List<Mix>();
   List<Solution> solutions = new List<Solution>();
   public int lastMovementId;
   public int movesCount;
   public int swapEvent;
   public int eventCount;
   public int score = 0;
   public int[] colorMask = new int[6];
   public bool isPlaying = false;
   public bool outOfLimit = false;
   public bool reachedTheTarget = false;
   public int creatingSugarTask = 0;
   public bool firstChipGeneration = false;
   public int matchCount = 0;
   public int stars;
   // Параметры
   public int eat;
   public int energy;
   public int money;
   public int coffee;
   public int hygiene;
   //Проверка исправности
   public bool coffeevarka;
   public bool multvarka;
   public bool icebox;
   public bool shower;
```

```
public bool bed;
public bool computer;
//Время
public int TimeMinute;
public int TimeHour;
public int LifeDays;
bool targetRoutineIsOver = false;
bool limitationRoutineIsOver = false;
public static int scoreC = 10;
void Awake() // стандартная функция Unity, запускается 1 раз в самом начале
      {
   main = this;
   combinations.Sort((Combinations a, Combinations b) => {
      if (a.priority < b.priority)
         return -1;
      if (a.priority > b.priority)
         return 1;
      return 0;
   });
}
void Start() // стандартная функция Unity, запускается 1 раз в начале
      {
   DebugPanel.AddDelegate("Complete the level", () => {
      if (isPlaying) {
         reachedTheTarget = true;
         movesCount = 0;
         score = LevelProfile.main.thirdStarScore;
      }
   });
   DebugPanel.AddDelegate("Fail the level", () => {
      if (isPlaying) {
         reachedTheTarget = false;
         limitationRoutineIsOver = true;
         movesCount = 0;
      }
   });
   DebugPanel.AddDelegate("Add a bomb", () => {
      if (isPlaying) {
         List<string> powerups = chipInfos.Select(x => x.name).ToList();
         powerups.Remove("SimpleChip");
         if (powerups.Contains("Sugar"))
```

```
                powerups.Remove("Sugar");
            if (powerups.Contains("Stone"))
                powerups.Remove("Stone");
            FieldAssistant.main.AddPowerup(powerups[Random.Range(0, powerups.Count)]);
        }
    });
}
void OnApplicationPause(bool pauseStatus) // void OnApplicationPause() // Функция, запускаемая при постановке игры на паузу
    {
    if (isPlaying)
        UIAssistant.main.ShowPage("Pause");
}
public static void Reset() // Функция, устанавливающая значение переменных при старте нового уровня
    {
    main.stars = 0;
    main.eventCount = 0;
    main.matchCount = 0;
    main.lastMovementId = 0;
    main.swapEvent = 0;
    main.score = 0;
    main.firstChipGeneration = true;
    main.isPlaying = false;
    main.movesCount = LevelProfile.main.limit;
    main.creatingSugarTask = 0;
    // добавленые данные
    Parametrs.main.Minus();
    Parametrs.main.MovesinGame = 0;
    StartNavel.main.Fin_Activ(false);
    main.reachedTheTarget = false;
    main.outOfLimit = false;
    main.targetRoutineIsOver = false;
    main.limitationRoutineIsOver = false;
    main.iteraction = true;
}
public void MovesFunction (int movess) // Функция, отнимающая параметры, когда игрок делает ход
{
eat -= movess;
energy -= movess;
```

```
coffee -= movess;
hygiene -= movess;

TimeMinute += 5 * movess;
Parametrs.main.MovesinGame += movess;
}
public void MixChips(Chip a, Chip b) // Функция, меняющая местами элементы
      {
   Mix mix = Mix.FindMix(a.chipType, b.chipType);
   if (mix == null)
      return;
   Chip target = null;
   Chip secondary = null;
   if (a.chipType == mix.pair.a) {
      target = a;
      secondary = b;
   }
   if (b.chipType == mix.pair.a) {
      target = b;
      secondary = a;
   }
   if (target == null) {
      Debug.LogError("It can't be mixed, because there is no target chip");
      return;
   }
   b.slot.chip = target;
   secondary.HideChip(false);
   target.SendMessage(mix.function, secondary);
}
public void Win_fifti() // Функция, выдающая победу с определенным шансом
{
   int chance = Random.Range(1, 11);
   if (score > LevelProfile.main.secondStarScore)
   {
      if (chance >= 4)
      {
         Debug.Log("Получилось 70 - "+ chance);
         StartCoroutine(GameCamera.main.HideFieldRoutine());
         FieldAssistant.main.RemoveField();
         StartCoroutine(YouWin());
      }
```

```
        else
        {
            Debug.Log("не получилось 70 - "+ chance);
            StartCoroutine(GameCamera.main.HideFieldRoutine());
            FieldAssistant.main.RemoveField();
            ShowLosePopup();
        }
    }
    else
    {
        if (chance >= 8)
        {
            Debug.Log("Получилось  35 - " + chance);
            StartCoroutine(GameCamera.main.HideFieldRoutine());
            FieldAssistant.main.RemoveField();
            StartCoroutine(YouWin());
        }
        else
        {
            Debug.Log("не получилось 35 - " + chance);
            StartCoroutine(GameCamera.main.HideFieldRoutine());
            FieldAssistant.main.RemoveField();
            ShowLosePopup();
        }
    }
}
public void Win() // Функция, запускающая вывод ифнормации о победе
{
    AnimationAssistant.main.Ferverk(0);
    ChangeBG.main.UpdateStatusDevice();
    StopAllCoroutines();
    StartCoroutine(QuitCoroutine());
}
public void StartForLevel(int level) // функция запускающая уровень - level
{
    Level.LoadMyLev(level);
}
public int GetBestPoints() //Получить лучший результат
{
    return ProfileAssistant.main.local_profile.GetScore(LevelProfile.main.level);
}
```

```
public int GetPoints() //Получить количество очков
{
   return score;
}
IEnumerator PlayLevelRoutine(int level)  // программа, запускающая уровень
      {
   yield return StartCoroutine(QuitCoroutine());
   while (CPanel.uiAnimation > 0)
      yield return 0;
   Level.LoadMyLev(level);
}
public void RestartLevel()     // программа, перезапускающая уровень
      {
   if (CPanel.uiAnimation > 0)
      return;
   StartCoroutine(PlayLevelRoutine(LevelProfile.main.level));
}

public void StartSession()  // Запускается при старте уровня
      {
   StopAllCoroutines();
   GameCamera.main.transform.position = new Vector3(0, 10, -10);
   GameCamera.cam.orthographicSize = 5;
   isPlaying = true;

   StartCoroutine(MovesLimitation());  // Завершение игры при обнулении ходов
   StartCoroutine(TargetSession(() => {  // Завершение игры при достижении цели
      return true;
   }));
   StartCoroutine(BaseSession());
   StartCoroutine(ShowingHintRoutine());
   StartCoroutine(ShuffleRoutine());
   StartCoroutine(FindingSolutionsRoutine());
   ChangeBG.main.ChengeArtAndBG(LevelProfile.main.level);
  GameCamera.main.ShowField();  // перемещает камеру к полю
   UIAssistant.main.ShowPage("Field");
}
IEnumerator BaseSession() // базовая куротина, проверающая победу или порожение
      {

   while (!limitationRoutineIsOver && !targetRoutineIsOver) {
```

```
            yield return 0;
        }
        StartNavel.main.Fin_Activ(false);
        Debug.Log(limitationRoutineIsOver + "   " + targetRoutineIsOver);
        if (!reachedTheTarget) {

            yield return StartCoroutine(GameCamera.main.HideFieldRoutine());
            FieldAssistant.main.RemoveField();
            ShowLoseGamePopup();

            yield break;
        }
        iteraction = false;


        yield return new WaitForSeconds(0.2f);
        UIAssistant.main.ShowPage("TargetIsReached");
        AudioAssistant.Shot("TargetIsReached");
        yield return StartCoroutine(Utils.WaitFor(() => CPanel.uiAnimation == 0, 0.4f));
        UIAssistant.main.ShowPage("Field");

        yield return StartCoroutine(Utils.WaitFor(CanIWait, 1f));

        yield return StartCoroutine(GameCamera.main.HideFieldRoutine());
        FieldAssistant.main.RemoveField();
        StartCoroutine(YouWin());
    }
    // Завершение игры при достижении цели
    IEnumerator TargetSession(System.Func<bool> success, System.Func<bool> failed = null)
    {
        reachedTheTarget = false;
        yield return 0;

        int score_target = LevelProfile.main.thirdStarScore;
        int score_fist_target = LevelProfile.main.firstStarScore;
        bool Fin_activ = false;
        while (!outOfLimit && (score < score_target || !success.Invoke()) && (failed == null ||
!failed.Invoke()))
        {
            if (score > score_fist_target && !Fin_activ)
            {
```

```
                Fin_activ = true;
                StartNavel.main.Fin_Activ(true);
            }

            yield return new WaitForSeconds(0.33f);
            if (movesCount <= 0 || eat <= 0 || energy <= 0 || hygiene <=0 || coffee <= 0)
            {
                StartNavel.main.Fin_Activ(false);
            }
        }
        if (score >= LevelProfile.main.firstStarScore && success.Invoke() && (failed == null ||
!failed.Invoke()))
            reachedTheTarget = true;
        Debug.Log("targetRoutineIsOver - "+movesCount + "  " + eat + "  " + energy);
        targetRoutineIsOver = true;

    }

    // Завершение игры при обнулении ходов
    IEnumerator MovesLimitation()
    {

        outOfLimit = false;

        while (movesCount > 0 && eat > 0 && energy > 0 && hygiene > 0 && coffee > 0)
            {

                yield return new WaitForSeconds(1f);
            if (movesCount <= 0 || eat <= 0 || energy <= 0 || hygiene <= 0 || coffee <= 0)
            {

                do
                {
                    yield return StartCoroutine(Utils.WaitFor(CanIWait, 1f));
                }
                while (FindObjectsOfType<Chip>().ToList().Find(x => x.destroying) != null);
            }
        }

        yield return StartCoroutine(Utils.WaitFor(CanIWait, 1f));
        Debug.Log(movesCount + "  " + eat + "  "+energy);
```

```
    limitationRoutineIsOver = true;
    outOfLimit = true;

}
// программа поиска и уничтожения комбинаций соответствующих правилам игры
IEnumerator FindingSolutionsRoutine() {
    List<Solution> solutions;
    int id = 0;
    while (true) {
        if (isPlaying) {
            yield return StartCoroutine(Utils.WaitFor(() => lastMovementId > id, 0.2f));
            id = lastMovementId;
            solutions = FindSolutions();
            if (solutions.Count > 0) {
                matchCount++;
                MatchSolutions(solutions);
            } else
                yield return StartCoroutine(Utils.WaitFor(() => {
                    return Chip.busyList.Count == 0;
                }, 0.1f));
        } else
            yield return 0;
    }
}

// Выход из уровня
public void Quit() {
    StopAllCoroutines();
    StartCoroutine(QuitCoroutine());
}
// программа выхода из уровня
IEnumerator QuitCoroutine() {
    while (CPanel.uiAnimation > 0)
        yield return 0;
    isPlaying = false;
    if (GameCamera.main.playing) {
        UIAssistant.main.ShowPage("Field");
        yield return StartCoroutine(GameCamera.main.HideFieldRoutine());
        FieldAssistant.main.RemoveField();
        while (CPanel.uiAnimation > 0)
            yield return 0;
```

```
    }
    UIAssistant.main.ShowPage("Loading");
    while (CPanel.uiAnimation > 0)
        yield return 0;
    yield return new WaitForSeconds(0.5f);
    StartNavel.main.MessQuit();
}
// программа активации бомб
IEnumerator CollapseAllPowerups() {
    yield return StartCoroutine(Utils.WaitFor(CanIWait, 0.5f));
    List<Chip> powerUp = FindPowerups();
    while (powerUp.Count > 0) {
        powerUp = powerUp.FindAll(x => !x.destroying);
        if (powerUp.Count > 0) {
            EventCounter();
            Chip pu = powerUp[Random.Range(0, powerUp.Count)];
            pu.DestroyChip();
        }
        yield return StartCoroutine(Utils.WaitFor(CanIWait, 0.5f));
        powerUp = FindPowerups();
    }
    yield return StartCoroutine(Utils.WaitFor(CanIWait, 0.5f));
}
// нахождение необходимой бомбы
List<Chip> FindPowerups() {
    return FindObjectsOfType<IBomb>().Select(x => x.GetComponent<Chip>()).ToList();
}
// показыает всплывающее окно проигрыша в уровне
void ShowLosePopup() {
    AudioAssistant.Shot("YouLose");
    isPlaying = false;
    GameCamera.main.HideField();
    UIAssistant.main.ShowPage("YouLose");
}
// проигрыш в игре
void ShowLoseGamePopup()
{
    AudioAssistant.Shot("YouLose");
    isPlaying = false;
    GameCamera.main.HideField();
    UIAssistant.main.ShowPage("YouLoseGame");
```

```
}
// программа, запускающая определённые действия при выигрыше в игре
IEnumerator YouWin()
        {
    AudioAssistant.Shot("YouWin");
    SteamAchiments.main.TakeAchievement("WIN_GAME");
    PlayerPrefs.SetInt("FirstPass", 1);
    isPlaying = false;

    if (ProfileAssistant.main.local_profile.current_level == LevelProfile.main.level)
        if (Level.all.ContainsKey(ProfileAssistant.main.local_profile.current_level + 1))
            ProfileAssistant.main.local_profile.current_level++;
    ProfileAssistant.main.local_profile.SetScore(LevelProfile.main.level, score);
    GameCamera.main.HideField();
    yield return 0;
    while (CPanel.uiAnimation > 0)
        yield return 0;
    yield return 0;
    UIAssistant.main.ShowPage("YouWin");
    Debug.Log("YouWin");
    AnimationAssistant.main.Ferverk(1);
    UserProfileUtils.WriteProfileOnDevice(ProfileAssistant.main.local_profile);
}
 // показывает окно при победе
        public void ShowYouWinPopup() {
    bool bestScore = false;
    if (ProfileAssistant.main.local_profile.GetScore(LevelProfile.main.level) < score)
    {
        ProfileAssistant.main.local_profile.SetScore(LevelProfile.main.level, score);
        bestScore = true;
    }
    UIAssistant.main.ShowPage(bestScore ? "YouWinBestScore" : "YouWin");
}
// уловия ожидания действий игрока
public bool CanIWait() {
    return isPlaying && Chip.busyList.Count == 0;
}
// счетчик действий
public void EventCounter() {
    eventCount++;
}
```

```
// функция поиска возможных ходов
public List<Move> FindMoves() {
    List<Move> moves = new List<Move>();
    if (!FieldAssistant.main.gameObject.activeSelf)
        return moves;
    if (LevelProfile.main == null)
        return moves;
    Solution solution;
    int potential;
    Side[] asixes = new Side[2] { Side.Right, Side.Top };
    foreach (Side asix in asixes) {
        foreach (Slot slot in Slot.all.Values) {
            if (slot[asix] == null)
                continue;
            if (slot.block != null || slot[asix].block != null)
                continue;
            if (slot.chip == null || slot[asix].chip == null)
                continue;
            if (slot.chip.id == slot[asix].chip.id)
                continue;
            Move move = new Move();
            move.from = slot.coord;
            move.to = slot[asix].coord;
            AnalizSwap(move);
            Dictionary<Slot, Solution> solutions = new Dictionary<Slot, Solution>();
            Slot[] cslots = new Slot[2] { slot, slot[asix] };
            foreach (Slot cslot in cslots) {
                solutions.Add(cslot, null);
                potential = 0;
                solution = MatchAnaliz(cslot);
                if (solution != null) {
                    solutions[cslot] = solution;
                    potential = solution.potential;
                }
                solution = MatchSquareAnaliz(cslot);
                if (solution != null && potential < solution.potential) {
                    solutions[cslot] = solution;
                    potential = solution.potential;
                }
                move.potencial += potential;
            }
```

```
                if (solutions[cslots[0]] != null && solutions[cslots[1]] != null)
                    move.solution = solutions[cslots[0]].potential > solutions[cslots[1]].potential ?
solutions[cslots[0]] : solutions[cslots[1]];
                else
                    move.solution = solutions[cslots[0]] != null ? solutions[cslots[0]] :
solutions[cslots[1]];
                AnalizSwap(move);
                if (Mix.ContainsThisMix(slot.chip.chipType, slot[asix].chip.chipType))
                    move.potencial += 100;
                if (move.potencial > 0)
                    moves.Add(move);
            }
        }
        return moves;
    }
    // меняет местами два элемента в соответствии с ходом
    void AnalizSwap(Move move) {
        Slot slot;
        Chip fChip = Slot.GetSlot(move.from).chip;
        Chip tChip = Slot.GetSlot(move.to).chip;
        if (!fChip || !tChip)
            return;
        slot = tChip.slot;
        fChip.slot.chip = tChip;
        slot.chip = fChip;
    }
    // проверяет, если ли 3 или более элементов в одном ряду и удаляет их
        void MatchSolutions(List<Solution> solutions) {

        if (!isPlaying)
            return;
        solutions.Sort(delegate (Solution x, Solution y) {
            if (x.potential == y.potential)
                return 0;
            else if (x.potential > y.potential)
                return -1;
            else
                return 1;
        });
        int width = LevelProfile.main.width;
        int height = LevelProfile.main.height;
```

```
bool[,] mask = new bool[width, height];
int2 key = new int2();
Slot slot;
for (key.x = 0; key.x < width; key.x++)
   for (key.y = 0; key.y < height; key.y++) {
      mask[key.x, key.y] = false;
      if (Slot.all.ContainsKey(key)) {
         slot = Slot.all[key];
         if (slot.chip)
            mask[key.x, key.y] = true;
      }
   }
List<Solution> final_solutions = new List<Solution>();
bool breaker;
foreach (Solution s in solutions) {
   breaker = false;
   foreach (Chip c in s.chips) {
      if (!mask[c.slot.x, c.slot.y]) {
         breaker = true;
         break;
      }
   }
   if (breaker)
      continue;
   final_solutions.Add(s);
   foreach (Chip c in s.chips)
      mask[c.slot.x, c.slot.y] = false;
}
foreach (Solution solution in final_solutions) {
   EventCounter();
   int puID = -1;
   if (solution.chips.Count(x => !x.IsMatcheble()) == 0) {
      int destroy = 0;
      foreach (Chip chip in solution.chips) {
         if (chip.id == solution.id || chip.IsUniversalColor()) {
            if (!chip.slot)
               continue;
            slot = chip.slot;
            if (chip.movementID > puID)
               puID = chip.movementID;
            chip.SetScore(Mathf.Pow(2, solution.count - 3) / solution.count);
```

```
                        if (!slot.block)
                            FieldAssistant.main.BlockCrush(slot.coord, true);
                        destroy++;

                        chip.DestroyChip();
                    }
                }
                Debug.Log("Destroy Chip - "+ destroy);
            } else
                return;
            solution.chips.Sort(delegate (Chip a, Chip b) {
                return a.movementID > b.movementID ? -1 : a.movementID == b.movementID ? 0
: 1;
            });
            breaker = false;
            foreach (Combinations combination in combinations) {
                if (combination.square && !solution.q)
                    continue;
                if (!combination.square) {
                    if (combination.horizontal && !solution.h)
                        continue;
                    if (combination.vertical && !solution.v)
                        continue;
                    if (combination.minCount > solution.count)
                        continue;
                }
                foreach (Chip ch in solution.chips)
                    if (ch.chipType == "SimpleChip") {
                        FieldAssistant.main.GetNewBomb(ch.slot.coord,          combination.chip,
ch.slot.transform.position, solution.id);
                        breaker = true;
                        break;
                    }
                if (breaker)
                    break;
            }
        }
    }
    // получить количество ходов
        public int GetMovesCount() {
        return movesCount;
```

```
        }
          // получить количество очков
            public float GetResource() {
           return (1f * movesCount *eat * energy*coffee*hygiene) / LevelProfile.main.limit;
        }
        // меняет элементы местами, если нет ходов
        IEnumerator ShuffleRoutine() {
          int shuffleOrder = 0;
          float delay = 1;
          while (true) {
            yield return StartCoroutine(Utils.WaitFor(CanIWait, delay));
            if (eventCount > shuffleOrder && !targetRoutineIsOver && Chip.busyList.Count ==
0) {
              shuffleOrder = eventCount;
              yield return StartCoroutine(Shuffle(false));
            }
          }
        }
        // программа меняющая местами элементы
        public IEnumerator Shuffle(bool f) {
          bool force = f;
          List<Move> moves = FindMoves();
          if (moves.Count > 0 && !force)
            yield break;
          if (!isPlaying)
            yield break;
          isPlaying = false;
          List<Slot> slots = new List<Slot>(Slot.all.Values);
          Dictionary<Slot, Vector3> positions = new Dictionary<Slot, Vector3>();
          foreach (Slot slot in slots)
            positions.Add(slot, slot.transform.position);
          float t = 0;
          while (t < 1) {
            t += Time.unscaledDeltaTime * 3;
            Slot.folder.transform.localScale = Vector3.one * Mathf.Lerp(1, 0.6f,
EasingFunctions.easeInOutQuad(t));
            Slot.folder.transform.eulerAngles = Vector3.forward * Mathf.Lerp(0,
Mathf.Sin(Time.unscaledTime * 40) * 3, EasingFunctions.easeInOutQuad(t));
            yield return 0;
          }
          if (f || moves.Count == 0) {
```

```
                f = false;
                RawShuffle(slots);
            }
            moves = FindMoves();
            List<Solution> solutions = FindSolutions();
            int itrn = 0;
            int targetID;
            while (solutions.Count > 0 || moves.Count == 0) {
                if (itrn > 100) {
                    ShowLosePopup();
                    yield break;
                }
                if (solutions.Count > 0) {
                    for (int s = 0; s < solutions.Count; s++) {
                        targetID = Random.Range(0, slots.Count - 1);
                        if (slots[targetID].chip && slots[targetID].chip.chipType != "Sugar" &&
slots[targetID].chip.id != solutions[s].id)
                            Swap(solutions[s].chips[Random.Range(0, solutions[s].chips.Count - 1)],
slots[targetID].chip);
                    }
                } else
                    RawShuffle(slots);
                moves = FindMoves();
                solutions = FindSolutions();
                itrn++;
                Slot.folder.transform.eulerAngles = Vector3.forward * Mathf.Sin(Time.unscaledTime
* 40) * 3;
                yield return 0;
            }
            t = 0;
            AudioAssistant.Shot("Shuffle");
            while (t < 1) {
                t += Time.unscaledDeltaTime * 3;
                Slot.folder.transform.localScale = Vector3.one * Mathf.Lerp(0.6f, 1,
EasingFunctions.easeInOutQuad(t));
                Slot.folder.transform.eulerAngles = Vector3.forward *
Mathf.Lerp(Mathf.Sin(Time.unscaledTime * 40) * 3, 0, EasingFunctions.easeInOutQuad(t));
                yield return 0;
            }
            Slot.folder.transform.localScale = Vector3.one;
            Slot.folder.transform.eulerAngles = Vector3.zero;
```

```
        isPlaying = true;
    }
    // функция поиска возможных решений
    List<Solution> FindSolutions() {
        List<Solution> solutions = new List<Solution>();
        Solution zsolution;
        foreach (Slot slot in Slot.all.Values) {
            zsolution = MatchAnaliz(slot);
            if (zsolution != null)
                solutions.Add(zsolution);
            zsolution = MatchSquareAnaliz(slot);
            if (zsolution != null)
                solutions.Add(zsolution);
        }
        return solutions;
    }
    // показывает подсказки
    IEnumerator ShowingHintRoutine() {
        int hintOrder = 0;
        float delay = 5;
        yield return new WaitForSeconds(1f);
        while (!reachedTheTarget) {
            while (!isPlaying)
                yield return 0;
            yield return StartCoroutine(Utils.WaitFor(CanIWait, delay));
            if (eventCount > hintOrder) {
                hintOrder = eventCount;
                ShowHint();
            }
        }
    }
        // меняет 2 элемента местами
    IEnumerator SwapByPlayerRoutine(Chip a, Chip b, bool onlyForMatching, bool byAI =
false) {
        System.Action breaker = () => {
            if (a && !a.destroying)
                a.busy = false;
            if (b && !b.destroying)
                b.busy = false;
            swaping = false;
        };
```

```
            if (!isPlaying)
                yield break;
            if (!iteraction && !byAI)
                yield break;

            if (swaping)
                yield break;
            if (!a || !b)
                yield break;
            if (a.destroying || b.destroying)
                yield break;
            if (Chip.busyList.Count > 0)
                yield break;

            if (a.slot.block || b.slot.block)
                yield break;
            if (movesCount <= 0 ||eat <= 0 || energy <= 0 || hygiene <= 0 || coffee <= 0)
                yield break;
            Mix mix = mixes.Find(x => x.Compare(a.chipType, b.chipType));
            int move = 0;
            swaping = true;
            Vector3 posA = a.slot.transform.position;
            Vector3 posB = b.slot.transform.position;
            float progress = 0;
            Vector3 normal = a.slot.x == b.slot.x ? Vector3.right : Vector3.up;
            float time = 0;
            a.busy = true;
            b.busy = true;

            while (progress < ProjectParameters.main.swap_duration) {
                if (!a || !b) {
                    breaker.Invoke();
                    yield break;
                }
                time = EasingFunctions.easeInOutQuad(progress / ProjectParameters.main.swap_duration);
                a.transform.position = Vector3.Lerp(posA, posB, time) + normal * Mathf.Sin(3.14f * time) * 0.2f;
                if (mix == null)
                    b.transform.position = Vector3.Lerp(posB, posA, time) - normal * Mathf.Sin(3.14f * time) * 0.2f;
```

```
    progress += Time.deltaTime;
    yield return 0;
}
a.transform.position = posB;
if (mix == null)
    b.transform.position = posA;
a.movementID = main.GetMovementID();
b.movementID = main.GetMovementID();
if (mix != null) {
    swaping = false;
    a.busy = false;
    b.busy = false;
    MixChips(a, b);
    yield return new WaitForSeconds(0.3f);
    MovesFunction(1);
    swapEvent++;
    yield break;
}
Slot slotA = a.slot;
Slot slotB = b.slot;
if (!slotB || !slotB || !a || !b) {
    breaker.Invoke();
    yield break;
}
slotB.chip = a;
slotA.chip = b;
move++;
int count = 0;
Solution solution;
solution = MatchAnaliz(slotA);
if (solution != null)
    count += solution.count;
solution = MatchSquareAnaliz(slotA);
if (solution != null)
    count += solution.count;
solution = MatchAnaliz(slotB);
if (solution != null)
    count += solution.count;
solution = MatchSquareAnaliz(slotB);
if (solution != null)
    count += solution.count;
```

```
            if (count == 0 && !onlyForMatching) {
                AudioAssistant.Shot("SwapFailed");
                while (progress > 0) {
                    if (!a || !b) {
                        breaker.Invoke();
                        yield break;
                    }
                    time = EasingFunctions.easeInOutQuad(progress / ProjectParameters.main.swap_duration);
                    a.transform.position = Vector3.Lerp(posA, posB, time) - normal * Mathf.Sin(3.14f * time) * 0.2f;
                    b.transform.position = Vector3.Lerp(posB, posA, time) + normal * Mathf.Sin(3.14f * time) * 0.2f;
                    progress -= Time.deltaTime;
                    yield return 0;
                }
                a.transform.position = posA;
                b.transform.position = posB;
                a.movementID = GetMovementID();
                b.movementID = GetMovementID();
                slotB.chip = b;
                slotA.chip = a;
                move--;
            } else {
                AudioAssistant.Shot("SwapSuccess");
                swapEvent++;
            }
            firstChipGeneration = false;
            if (!byAI)
            {
                Debug.Log("move = " + move);
                MovesFunction(move);
            }
            EventCounter();
            a.busy = false;
            b.busy = false;
            swaping = false;
        }
```