МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«МАГНИТОГОРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМ. Г.И. НОСОВА»

Многопрофильный колледж

Отделение №1 «Информационные технологии и документоведение»

ПЦК «Информатика и вычислительная техника»

Допустить к защите

Заведующий отделением

\_\_\_\_\_\_\_\_\_\_/ Н.В. Сидорова

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**ДИПЛОМНЫЙ ПРОЕКТ**

**Д.09.02.03.18.ДП.ПЗ**

Студента Клюшина Михаила Алексеевича

(фамилия имя отчество)

На тему: Разработка мобильной игры на основе 3D Unity в жанре «Платформер»

(полное наименование темы)

Состав дипломного проекта:

1. Пояснительная записка на \_\_ страницах

2. Графическая часть на \_\_\_\_\_\_ листах

Руководитель преподаватель, к.т.н., доцент Тутарова В.Д.

(подпись, дата, должность, ученая степень, звание, Ф.И.О.)

Консультанты преподаватель, Котельникова Ю.М.

(подпись, дата, должность, ученая степень, звание, Ф.И.О.)

Рецензент Белявский А.Б.

(подпись, дата, должность, ученая степень, звание, Ф.И.О.)

Председатель ПК/ПЦК преподаватель Зорина И.Г.

(подпись, дата, должность, ученая степень, звание, Ф.И.О.)

|  |  |
| --- | --- |
| Отметка нормоконтролера  \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_Турбина О.А./  (подпись, дата) (ФИО)  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г. | Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г. |

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Магнитогорский государственный технический университет им. Г.И. Носова»

Многопрофильный колледж

УТВЕРЖДАЮ:

Заведующий отделением

№1 «Информационные технологии   
и документоведение»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Н.В. Сидорова

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

|  |  |
| --- | --- |
| **ДИПЛОМНЫЙ ПРОЕКТ** | |
| **задание** | |
| **Тема:** Разработка мобильной игры на основе 3D Unity в жанре «Платформер»  (полное наименование темы)  Студенту Клюшину Михаилу Алексеевичу  (фамилия, имя, отчество) | |
| Тема утверждена приказом №\_\_\_\_\_\_\_от\_\_\_\_\_\_\_\_\_2018г. | | | |
| Срок выполнения «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г. | | | |
| Исходные данные к работе: | | | |
| данные для наполнения приложения:  отчетные документы;  прайс-листы;  рекламные материалы;  литературные и научные источники. | | | |
|  | | | |
|  | | | |

Перечень вопросов, подлежащих разработке в выпускной квалификационной работе:

1. Аналитическая часть: описание и анализ предметной области, выделение недостатков существующей системы, анализ существующих средств автоматизации, построение информационной модели – диаграммы потоков данных соответствующей предметной области, обоснование актуальности разработки приложения, указание возможных ограничений, назначение разработки с указанием категорий пользователей, цель и задачи проекта.
2. Проектная часть:
3. Экономическая часть:

* расчет трудоемкости разработки программного обеспечения;
* расчет экономической эффективности

Графическая часть *(при наличии)*:

презентационные материалы;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

листинг программного кода.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Консультанты по работе (с указанием относящихся к ним разделов) *(при наличии)*:\_\_\_\_\_

Котельникова Ю.М. раздел «Экономическая часть»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |
|  |
| Руководитель:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_В.Д.Тутарова\_\_\_\_\_\_\_\_\_/  (подпись, дата) |
| Задание получил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_М.А. Клюшин\_\_\_\_\_\_\_\_/  (подпись, дата) |

# Содержание

ВВЕДЕНИЕ

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Краткая характеристика средств общения в сети Интернет

1.2 Анализ существующих средств общения в сети Интернет

1.3 Выбор средств разработки программного обеспечения

1.4 Модели системы общения в сети

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Постановка задачи

2.2 Разработка физической и логической структуры Веб-сайта

2.3 Разработка базы

2.3.1 Проектирование структуры базы данных

2.3.2 Создание таблиц, связей между ними

2.4 Проектирование интерфейса

2.5 Руководство пользователя

2.6 Руководство программиста

2.7 Информационная безопасность

2.7.1 Виды возможных угроз веб-приложения

2.7.2 Информационная безопасность приложения

3 ЭКОНОМИЧЕСКАЯ ЧАСТЬ

3.1 Расчет трудоемкости разработки программного продукта3.2 Расчет экономической эффективности программного продукта

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЕ А

# Введение

Научно-технический прогресс, набравший к концу ХХ в. головокружительную скорость, послужил причиной появления такого чуда современности как компьютер и компьютерные технологии.

С совершенствованием компьютеров совершенствовались и игры, привлекая все больше и больше людей. Позже появились мобильные телефоны, на которые с большим успехом стали создаваться игры различных жанров, ничем не уступающие своим компьютерным аналогам.

Мобильные телефоны давно уже превратились из средств связи в многофункциональные мультимедийные устройства, которым по силам удовлетворить очень многие запросы своих владельцев - касается ли это музыки, видео, доступа в интернет или игр.

На сегодняшний день компьютерная техника достигла такого уровня развития, что позволяет программистам разрабатывать очень реалистичные игры с хорошим графическим и звуковым оформлением не только для компьютеров, но и для мобильных устройств.

Современная мобильная игровая индустрия является важным направлением в развитии сотовых телефонов, о чем свидетельствуют хотя бы цифры многочисленных исследований. На многих рынках мобильные развлечения занимают, вслед за компьютерными играми, ведущие позиции в игровой индустрии.

На данный момент мобильный рынок предоставляет большой выбор различных смартфонов, среди которых на 2017г. 73,54% рынка составляли устройства под операционной системой Android [1]. Факт популярности Android устройств обусловлен тем, что они охватывают большинство ценовых категорий и это дает огромный выбор для покупателя.

В связи с популярностью мобильных устройств среди покупателей, играют в игры не только дети, но и взрослые. Сейчас мобильные устройства распространены повсеместно. Наверное, нет ни одного ученика, который бы не любил играть в игры, некоторые же готовы играть день и ночь. Родители, напротив, запрещают играть в компьютерные, хотя и сами порой не прочь поиграть.

Целью данного дипломного проекта является разработка мобильной игры на основе 3D Unity в жанре «Платформер».

Для выполнения дипломного проекта поставлены следующие задачи:

* описание предметной области, выбранной для изучения;
* определение проблем исследования и возможные варианты решения этих проблем;
* оценка существующих решений, выявленных проблем;
* анализ и оценка существующих программных и технических средств, используемых для решения проблемы;
* обоснование и формулировка цели и задач исследования.
* разработка модели (концептуальная, информационная и др.);
* декомпозиция проблемы;
* построение структуры, схемы данных;
* построение схемы функционирования программного продукта;
* проектирование пользовательского интерфейса (выбор диалога, структура справочной системы);
* разработка автоматизированной информационной системы « »;
* оценка информационной безопасности;
* расчет трудоемкости разработки программного продукта и определение его цены;
* оценка экономической эффективности.

# 1 Анализ области мобильных игр в жанре «Платформер»

## 1.1 Основные жанры мобильных игр

На данный момент в игровой индустрии присутствует огромное количество различных игровых жанров, это обусловлено тем, что индустрия не стоит на месте и постоянно развивается. Ниже приведены основные жанры игр, появившиеся одними из первых и до сих пор остающимися популярными.

Аркада (“Arcade”) – жанр игр, характеризующийся коротким по времени, но интенсивным игровым процессом [2].

Некоторые игровые обозреватели, журналы или сайты даже выделяют подобные игры в отдельный жанр, причисляя к ним и различные «Платформер»ы. Изначально данные игры выпускались для игровых автоматов и имели упрощенный игровой процесс, графику или иную составляющую.

Экшен (“Action”) – жанр игр, в которых успех игрока в большой степени зависит от его скорости реакции и способности быстро принимать тактические решения [3].

Обозначение жанра игры как “action” относительно редко используется без дополнения, так как это понятие очень широкое и им можно охарактеризовать более половины всех когда-либо вышедших игр. В эту категорию попадают шутеры, файтинги, «Платформер»ы и т. п. Гораздо чаще слово “action” подставляется к основному жанру игры. Например, Action-adventure – приключенческая игра, включающая в себя боевые или акробатические элементы, нехарактерные для традиционных квестов.

Гонки – жанр игр с видом от первого или от третьего лица, где игрок принимает участие в гоночном соревновании на каком-либо типе наземных, водных, воздушных или космических транспортных средств [4]. Гонки могут быть основанными как каких-то реально существующих гоночных сериях, либо могут происходить в рамках полностью вымышленных сеттингов. В общем случае гоночной игрой может быть любая игра из полного спектра от автосимуляторов высокой степени реализма до относительно простых аркадных гоночных игр. Также гоночные игры могут относиться к категории спортивных игр.

Головоломки – жанр игр, целью которых является решение логических задач, требующих от игрока задействования логики, стратегии и интуиции [5]. Предтечей жанра являлись настольные, графические и механические головоломки – от кроссвордов до кубика Рубика. Эти головоломки требовали от игрока логики и ловкости в решении, которые также стали играть важную роль в прототипах жанра, таких, как Q\*bert и Boulder Dash. Эталоном же жанра стала игра Тетрис, появившаяся в 1985 году и сочетавшая в себе простой и захватывающий игровой процесс.

Стратегии – игры данного жанра характеризуются тем, что игроку для достижения цели необходимо применять стратегическое мышление, и оно противопоставляется быстрым действиям и реакцией, которые, как правило, не обязательны для успеха в таких играх [6]. Стратегические игры бывают абстрактные, настольные (шахматы, шашки, «Монополия»), с симуляцией менеджмента и другие.

Симуляторы – игры, предоставляющие возможность симуляции и управления тем или иным процессом из реальной жизни.

Существует два основных подкласса: симулятор техники и спортивные игры. Первые предполагают высокую сложность физических расчетов. Их задача – максимально приблизить поведение прототипа к реальному. Вторые - попытка имитации спортивных соревнований. Игрок точно так же, как и в “action”, управляет человеком (или даже несколькими). С первыми этот жанр роднит максимально реалистичное поведение персонажей и их взаимодействие.

## 1.2 «Платформер» как жанр в мобильных играх

«Платформер» (англ. “platformer”) – жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня [7]. Некоторые предметы, называемые пауэр-апами (англ. “power-up”), наделяют управляемого игроком персонажа особой силой, которая обычно иссякает со временем (к примеру: силовое поле, ускорение, увеличение высоты прыжков). Коллекционные предметы, оружие и «пауэер-ап» собираются обычно простым прикосновением персонажа и для применения не требуют специальных действий со стороны игрока. Реже предметы собираются в «инвентарь» героя и применяются специальной командой (такое поведение более характерно для аркадных головоломок).

Противники (называемые «врагами»), всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Иногда противник может быть нейтрализован либо прыжком ему на голову, либо из оружия, если им обладает герой. Смерть живых существ обычно изображается упрощённо или символически (существо исчезает или проваливается вниз за пределы экрана).

Уровни, как правило, изобилуют секретами (скрытые проходы в стенах, высокие или труднодоступные места), нахождение которых существенно облегчает прохождение и подогревает интерес игрока.

Игры подобного жанра характеризуются нереалистичностью, рисованной мультяшной графикой. Героями таких игр обычно бывают мифические существа (к примеру: драконы, гоблины) или антропоморфные животные.

Для того чтобы игроку было интересно играть, нужно обеспечить удобное управление и разнообразие контента. Это достигается путем добавления в игру различных уровней и достижений за выполнение конкретных задач.

Чтобы обеспечить работоспособность игры на платформе Android необходимо использовать такие наборы инструментов как Android NDK (Native Development Kit), JDK (Java Development Kit) и Android SDK (Software Development Kit). Данные наборы предоставляют все необходимые инструменты для того, чтобы производить сборку приложения под Android.

## 1.3 Анализ и оценка существующих мобильных игр в жанре ««Платформер»»

Был выполнен анализ существующих мобильных игр в жанре «Платформер». Исследование интернет - источников выявило следующие актуальные на данный момент игры:

* Sword of Xolan [8];
* Meganoid [9];
* Dan the Man [10];
* Cuphead Mobile [11];
* Lode Ronner 1 [22]

### 1.3.1 Sword of Xolan

Sword of Xolan (рис. 1.1) является простой и понятной игрой для начинающего пользователя. «Платформер»ы всегда были сложными в прохождении, часто происходят моменты, когда у персонажа остается «одна жизнь» или же, срываясь с платформы, приходиться начинать уровень с начала. Эти проблемы до сих пор преследуют жанр. В мобильных «Платформер»ах к этим проблемам добавляются неотзывчивое сенсорное управление и промахи мимо едва обозначенных кнопок.

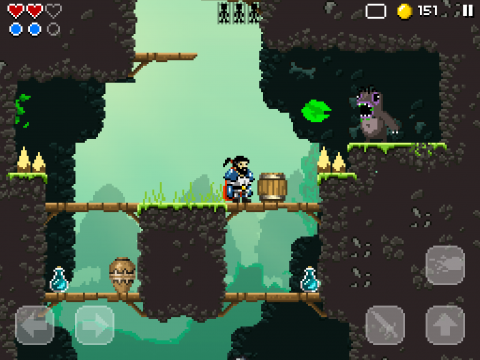
[](http://s.4pda.to/xnPXTz1Emz164ZFosz1wFHX8Y7GQ2n6.png)

Рисунок 1.1 – Игровой процесс в Sword of Xolan

В Sword of Xolan этого нет. Автор постепенно вырезал из своего проекта все раздражающие элементы жанра. Управление – отличное, по большим кнопкам тяжело промахнуться. Смертельных обрывов нет, миллиметровать между оврагами с шипами никто не заставляет. Врагов много, но с ними можно справиться с помощь меча и огненных выстрелов. Уровни разветвлённые, но потеряться очень сложно. В этой игре все продуманно до малейшей детали. Монетки спрятаны в укромных местах, но пройти игру можно и без покупок в магазине.

В Sword of Xolan, сравнивая с другими представителями жанра, нет ничего особенного. Если разобрать её на составляющие, то это – обычный «Платформер».

Но в этом и кроется основная фишка Sword of Xolan. Не предложив ничего нового, она, вбирая в себя лучшие черты жанра, что позволяет выделиться на фоне остальных.

**Достоинства:**

* идеально выверенная игровая механика;
* компактные, но продуманные уровни.

**Недостатки:**

* слишком сложные сражения с боссами;
* нет сюжета.

### 1.3.2 Meganoid

Meganoid для Android представляет собой очень интересный сложный «Платформер» с элементами action от студии OrangePixel, про приключения астронавта. Игра получила олдскульный интерфейс, пиксельное графическое оформление, неплохую анимацию вкупе с кучей спецэффектов и отличное звуковое сопровождение. Игровой процесс построен на принципах Roguelike (рис. 1.2), а это значит, что каждый новый мир генерируется непосредственно во время запуска игры и смерть нашего подопечного ждет на каждом шагу.



Рисунок 1.2 – Игровой процесс в Meganoid

Судя по официальной предыстории, смысл игрового процесса заключается в бесконечном исследовании бесконечного мира в поисках космического корабля. Где приземлился главный герой, не сообщается. Во время исследования планеты астронавт забрался так далеко, что заблудился. Собственно, теперь он пытается найти звездолет или челнок, чтобы отправиться в другое место. Но замыслам главного героя сбыться не суждено. Он так и будет блуждать из одной локации в другую, встречая различных монстров, попадая в ловушки и собирая многочисленные бонусы. Среди стационарных препятствий можно отметить автоматические лазерные и скорострельные пушки, а также торчащие повсюду шипы. Эти препятствия необходимо обходить стороной, так как они не оставляют шанса на выживание, в отличие от некоторых антагонистов.

Подземелья игры Meganoid населяют гигантские блохи, огненные котята и огромные верзилы. С первыми двумя можно справиться, запрыгнув на них сверху, а от третьего лучше держаться подальше. Найденные бочки можно перемещать по ровному полу и разбивать. Внутри обычно имеется какой-нибудь бонус. Это либо микрочип, активирующий портал в другую локацию, либо связка динамита, разрушающая все без исключения в радиусе поражения, либо кристаллы и монеты. Иногда главный герой находит радио, и игра наполняется оригинальными композициями в ритмичном стиле.

**Достоинства:**

* генерация локаций случайным образом;
* нелинейное прохождение;
* множество предметов коллекционирования.

**Недостатки:**

* сложный игровой процесс.

### 1.3.3 Dan the Man

[Dan The Man](http://0s.m5qw2zlt.nvqws3booj2q.cmle.ru/pc/games/dan_the_man/) – отличный двухмерный «Платформер» с системой развития (рис. 1.3), возможностью поменять облик персонажа и множеством способов уничтожения противников [https://games.mail.ru/pc/articles/review/dan\_the\_man/].

В начале игры игроку показывают, эпизод из середины истории, умолчав об предыдущих событиях из жизни главного героя. Им посвящены семь роликов, которые можно найти на YouTube и на сайте Studio Joho. Эти ролики рассказывают историю до начала игры Dan the Man.

В начале игры в арсенале главного героя только удары ногами и руками, но по мере продвижения по уровням он получает самое более совершенное оружие: от стальных сюрикенов до гранатомета. Боезапас каждого орудия ограничен, поэтому расходовать его необходимо аккуратно.



Рисунок 1.3 – Игровой процесс в Dan the Man

Разновидностей врагов в Dan The Man большое количество: боевики с дубинками, дробовиками и автоматами, летающие боевики на джетпаках, гигантские громилы. В конце некоторых уровней происходит битва с боссом. В игре их реализовано немного, но победить их довольно сложно. К каждому ключевому противнику нужен свой подход.

Платформенные задания более щадящие, чем в старых восьмибитных играх. Не придется заново перепроходить один и тот же уровень из-за смерти персонажа, при падении персонаж не умирает, а лишь теряет незначительную часть здоровья. В случае смерти героя всегда есть возможность возобновить игру с последнего сохранения. Для этого нужно посмотреть рекламу или уплатить 200 внутриигровых монет.

Помимо Дэна, играть можно еще и за девушку Джози. После нескольких вступительных уровней появляется возможность создать собственного героя: в редакторе доступно множество причесок, костюмов и прочих косметических предметов. Так же разрешается прокачивать умения и оружие персонажа. Однако параметры и навыки не переходят от одного персонажа к другому. Улучшения покупаются за монеты, которые игрок зарабатывает по ходу прохождения. В игре есть режим боя «Арена», где на каждом уровне нужно отбиваться от трех волн врагов.

Рекламы в Dan the Man очень много. Реклама появляется на точках сохранения, в начале уровня, в конце уровня. Для получения бонуса необходимо просмотреть 30-секундное рекламное видео. От всего этого избавляет единоразовый «Премиум»-абонемент.

**Достоинства:**

* простой игровой процесс;
* интересный сюжет.

**Недостатки:**

* неполная история сюжета;
* большое количество рекламы.

### 1.3.4 Cuphead mobile

Cuphead (с англ. — «Чашкоголовый») – игра в жанре run and gun и «Платформер», разработанная и изданная канадской компанией StudioMDHR Entertainment (рис. 1.4). Управляя персонажем по имени Капхед, игрок сражается с серией боссов, чтобы вернуть долг дьяволу. Игра была сильно вдохновлена работами аниматоров 1930-х годов, таких как Fleischer Studios Макса Флейшера, и стремилась сохранить их сюрреалистические качества.



Рисунок 1.4 – Игровой процесс в Cuphead

Cuphead – shoot 'em up-игра (с англ. – «перестреляй их всех»), основанная на непрерывных боях с боссами. Персонаж игрока, Капхед, проиграл спор с дьяволом и пытается вернуть долг. Игра имеет вид разветвляющейся последовательности уровней. У Капхеда бесконечное число попыток, и он не теряет оружие после смерти. У главного героя игры есть способность парирования, и парирование различных объектов, закодированных цветом, заполняет специальный счётчик, который позволяет ему выполнять специальную способность. Уровни доступны через внешний мир в стиле action RPG, который имеет собственные секретные локации. В игре есть кооперативный режим, в котором в сражениях с боссами может участвовать еще один игрок, управляя вторым персонажем по имени Магмен.

**Достоинства:**

* уникальный игровой процесс;
* необычный дизайн;
* интересный сюжет.

**Недостатки:**

* сложность игры;
* игра находится в alpha версии.

### 1.3.5 Lode Runner 1

Lode Runner 1 – приключенческий «Платформер» для мобильных устройств на базе ОС андроид, в котором пользователям предстоит управлять персонажем, оказавшимся в настоящем лабиринте с ценными сокровищами и опасными монстрами (рис. 1.5). Приложение распространяется на бесплатной основе. В нем доступно более ста увлекательных уровней, сложность которых постепенно увеличивается.



Рисунок 1.4 – Игровой процесс в Lode Runner 1

«Платформер» включает в себя десятки заданий различной сложности. В качестве главного героя выступает молодой человек с пышными рыжими волосами, одетый в необычный костюм, чем-то напоминающий скафандр. Судя по всему, это космонавт, который оказался на другой планете где-то далеко в космическом пространстве. Как выяснилось, здесь хранятся несметные сокровища – слитки золота можно встретить на каждом шагу, при этом они никому не принадлежат. И лишь жуткие инопланетные монстры охраняют этот клад.

Главного героя ждут многочисленные испытания. На неизведанной планете действуют свои правила. Например, космонавт не может здесь прыгать – слишком сильна сила притяжения. Однако у него имеется эффективное оружие – огонь. На каждом уровне цель главного героя – добраться до двери с надписью «Выход». Однако есть одна проблема – дверь заблокирована. Она автоматически откроется только тогда, когда персонаж отыщет и заберет все спрятанное золото. Никаких особенных тайников в лабиринте нет. Просто иногда не так просто добраться до золотых слитков.

Все уровни представлены в виде помещения из нескольких этажей с многочисленными стенками, лестницами. Время нахождения на уровне не ограничено. Спускаться и подниматься по лестницам персонаж может без проблем. При падении с большой высоты с персонажем ничего не происходит. Иногда кажется, что герой в тупике – невозможно пройти дальше, вокруг одни каменные стены. Однако нужно внимательнее приглядеться. Наверняка под ногами есть хлипкий участок, который можно прожечь огнем. Такие отверстия быстро восстанавливаются.

Достоинства:

* увлекательный игровой процесс;
* элементы головоломки;
* большое количество уровней;
* удобное управление.

Недостатки:

* не обнаружено.

Сравнение игр по общим критериям приведено в таблице 1.1.

Таблица 1.1 – Сравнение игр.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерии | Sword of Xolan | Meganoid | Dan the Man | Cuphead Mobile |
| Наличие сюжета | + | – | + | + |
| Сложность игрового процесса | Средняя | Высокая | Низкая | Высокая |
| Наличие рекламы | + | – | + | – |

## 1.4 Вывод по главе 1

В главе «Анализ области мобильных игр в жанре «Платформер» рассмотрены основные жанры мобильных игр, имеющие одну из главных ролей в игровой индустрии. Благодаря проведенному анализу жанров, сделан вывод что жанр «Платформер» имеет в себе черты других жанров, таких как “arcade” и “action” но при этом, привносит и свои индивидуальные элементы, что позволяет выделится среди остальных.

При анализе существующих мобильных игр в жанре «Платформер» были оценены наиболее популярные на данный момент игры, такие как “Sword of Holan”, “Meganoid”, “Dan the Man”, “Cuphead Mobile” и ”Lode Runner 1”. Учитывая все достоинства и недостатки этих продуктов была поставлена цель – разработать собственную мобильную игру на основе 3D Unity в жанре «Платформер».

# 2 Проектные решения при создании мобильной игры в жанре «Платформер»

## 2.1 Постановка задачи

В результате проведенного анализа мобильных игр в жанре «Платформер» было выявлено, что данный жанр является одним из популярных на данный момент и ориентирован на различные возрастные категории.

Цель – разработать мобильную игру на основе 3D Unity в жанре «Платформер».

Исходя из поставленной цели, были сформулированные следующие задачи:

1. проанализировать данную предметную область;
2. проанализировать существующие мобильные игры в жанре «Платформер»;
3. выбрать средства разработки;
4. спроектировать основные игровые аспекты;
5. разработать клиентскую часть приложения, интерфейс приложения
6. администрировать приложение;
7. организовать информационную безопасность;
8. рассчитать трудоемкость разработки программного обеспечения;
9. рассчитать экономическую эффективность системы.

Игры этого жанра отличаются простотой сюжета, имеют многоуровневое прохождение и простое управление.

Простота игрового процесса не является решающим фактором в популярности игры, из этого можно сделать вывод что игра может быть как простой, так и сложной.

За различные достижения игрок должен получать поощрения в виде очков, игровых предметов или бонусов.

Уровни должны содержать NPC препятствующие прохождению игроку или головоломки, при уничтожении или выполнении которых игрок переходит на следующий уровень или получает бонусы.

Игра должна быть ориентирована на пользователей различных возрастных категорий, что позволит добиться более широкого охвата аудитории. Игра будет развивать мышление, логику, мелкую моторику рук и концентрировать внимание.

## 2.2 Выбор средств разработки программного обеспечения

### 2.2.1 Визуальная часть проекта

В данном дипломном проекте присутствует большое количество графических элементов, это обусловлено тем, что в игре присутствует множество различных объектов, имеющих индивидуальное отображение в едином для общей картины стиле. В связи с поставленной задачей и моим опытом работы в этой программе, для создания и редактирования элементов графики был выбран редактор Adobe Photoshop [12].

Adobe Photoshop – многофункциональный графический редактор, разработанный и распространяемый фирмой Adobe Systems (рис. 2.1). В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известным продуктом фирмы Adobe.

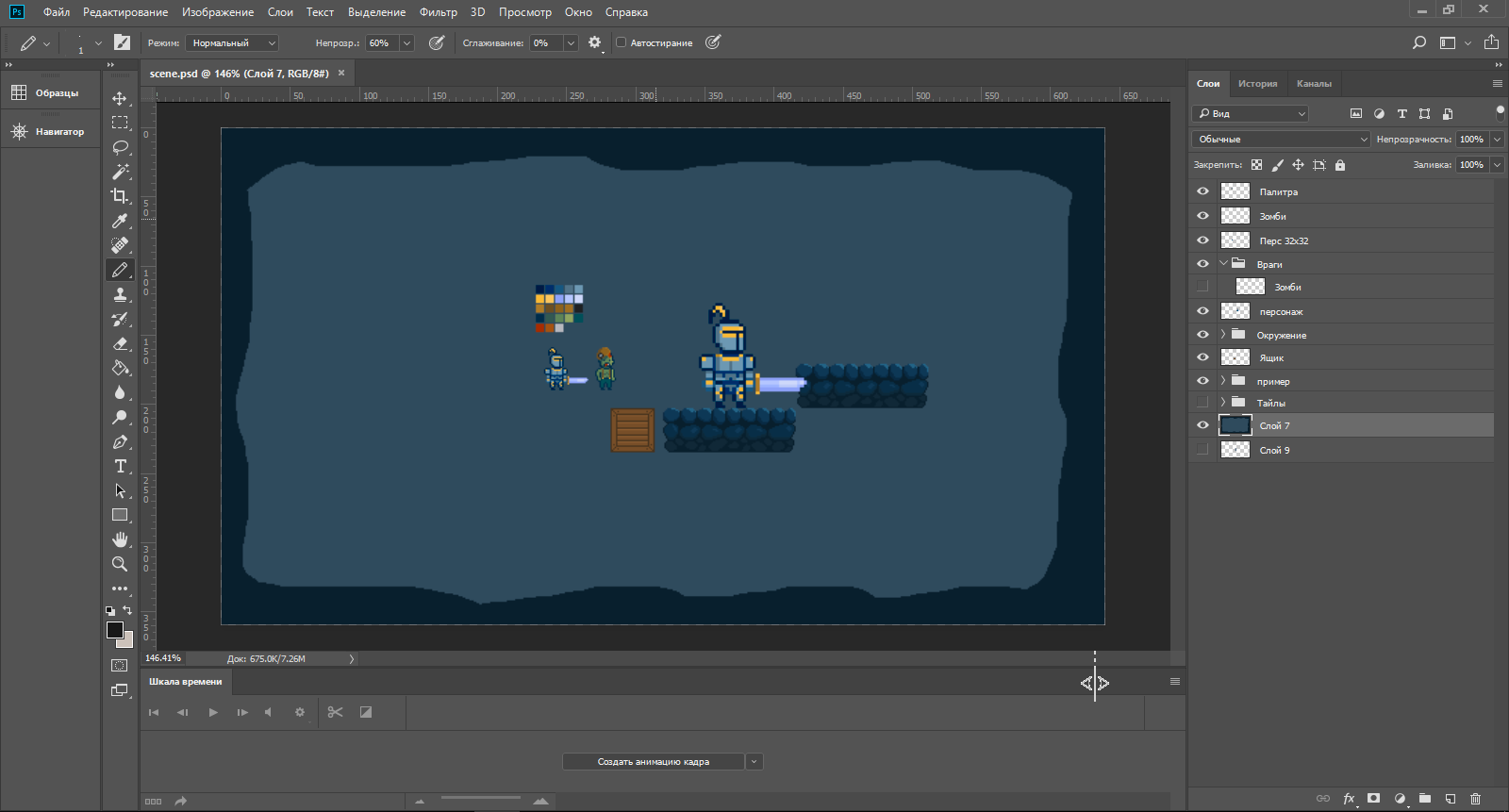


Рисунок 2.1 – Интерфейс программы Adobe Photoshop CC 2017

Данный редактор предоставляет широкий набор инструментов для создания и редактирования изображений в необходимом для вас стиле.

### 2.2.2 Программная часть проекта

Для создания логической и программной части дипломного проекта, выбор был сделан в пользу игрового движка Unity [13].

Unity – межплатформенная среда разработки компьютерных игр. Unity позволяет создавать приложения, работающие под более чем 20 различными операционными системами, включающими персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

Редактор Unity имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Движок поддерживает два скриптовых языка: C#, JavaScript (модификация). Ранее была поддержка Boo (диалект Python), но его убрали в 5-ой версии. Расчёты физики производит физический движок PhysX от NVIDIA.

Выбор обусловлен тем, что данный движок имеет следующие преимущества:

* Простота и удобство использования;
* Наличие обширной документации и уроков на русском языке;
* Наличие бесплатной версии;
* Неограниченные возможности применения.

Для написания и редактирования программного кода был выбран редактор Visual Studio 2017.

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms [14].

Visual Studio включает в себя редактор исходного кода и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения.

### 2.2.3 Язык программирования

Для реализации программного кода был выбран язык программирования C# [15].

C# – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников – языков C++, Pascal, Модула, Smalltalk и, в особенности, Java – С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Выбор был сделан исходя из того, что в игровом движке Unity, С# является основным поддерживаемым языком и большая часть документации написана именно под него.

## 2.3 Проектирование приложения

### 2.3.1 Алгоритм приложения

Алгоритм – набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата.

Общий алгоритм для игры можно представить в виде блок-схемы, изображенной на рисунке 2.2.

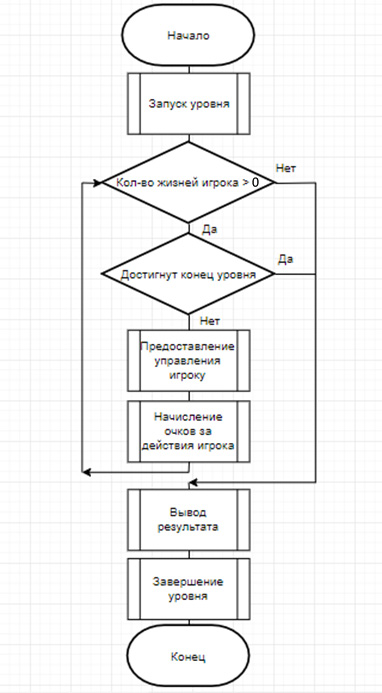


Рисунок 2.2 – Общая блок-схема игры

Словесно алгоритм можно описать следующим образом:

1. начало;
2. производим запуск уровня с генерацией мира и добавлением основных объектов;
3. выполняем проверку «жив ли персонаж», при выполнении условия (игрок жив) продолжаем выполнение. При невыполнении условия, завершаем уровень.
4. проверяем, достиг ли игрок конца уровня. При выполнении условия переходим к завершению уровня, при невыполнении переходим дальше;
5. передаем управление игровым персонажем игроку;
6. производим начисление очков за выполнение определенных действий.
7. выводим результат набранных очков игроку;
8. выполняем завершение уровня;

### 2.3.2 Диаграмма состояний

Диаграмма состояний – это диаграмма состояний из теории автоматов (раздел дискретной математики, изучающий абстрактные автоматы – вычислительные машины, представленные в виде математических моделей – и задачи, которые они могут решать) со стандартизированными условными обозначениями (рисунок 2.3) [19]:

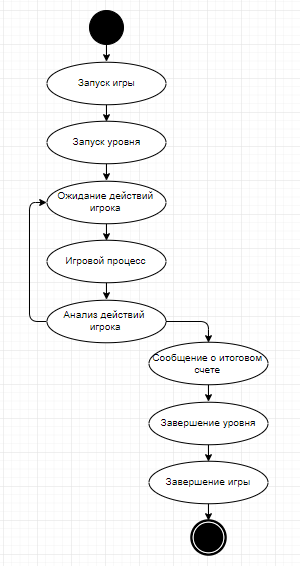


Рисунок 2.3 – Диаграмма состояний

### 2.3.3 Диаграмма последовательности

Диаграмма последовательности (англ. sequence diagram) – диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актёров (действующих лиц) ИС в рамках какого-либо определённого прецедента (отправка запросов и получение ответов). Используется в языке UML.

Для дипломного проекта была создана своя UML-диаграмма последовательности (рис. 2.4), показывающая последовательность действий в игре и линию жизни противника.

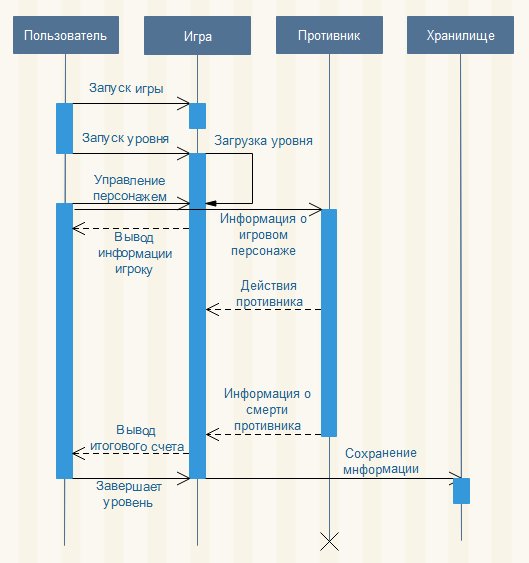


Рисунок 2.4 – Диаграмма последовательностей

### 2.3.4 Функциональная диаграмма

Функциональная диаграмма – отражает взаимные связи функций разрабатываемого программного обеспечения. Они создаются на ранних стадиях проектирования систем для того, чтобы помочь проектировщику выявить основные функции и составные части проектируемой программной системы и, по возможности, обнаружить и устранить существенные ошибки.

На рисунке 2.5 приведена функциональная диаграмма, отображающая взаимодействие между программными средствами, используемыми при разработке мобильной игры.

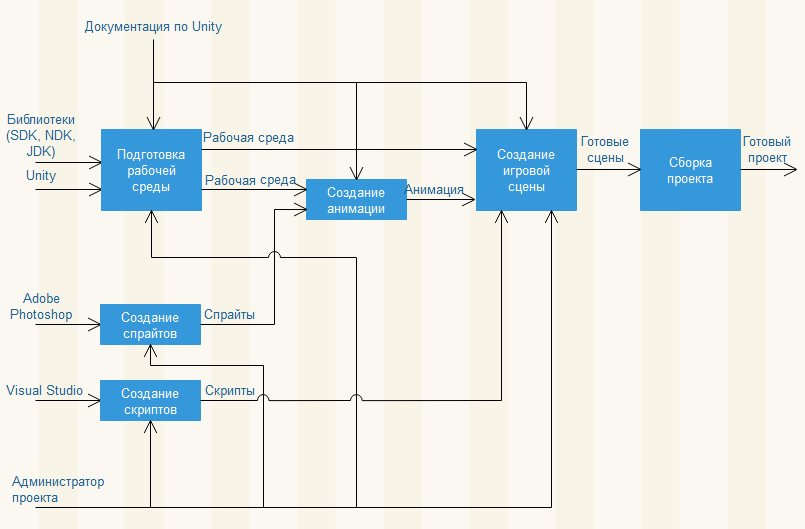


Рисунок 2.5 – Функциональная диаграмма

## 2.4 Моделирование проекта

Логическое 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 реализации системы [16].

Логическое проектирование не учитывает особенностей среды, в которой будет функционировать программа. При физическом проектировании все эти факторы (технические и программные средства компьютера) наоборот должны быть учтены.

### 2.4.1 Визуальная часть

Визуальная часть игры – совокупность изображений и анимации, отображающие действия происходящие в игре. Визуальная часть необходима для того, чтобы игрок наглядно видел свой прогресс в игре.

Графика в игре должна быть выполнена в едином стиле и передавать обстановку происходящую в игре. Для создания спрайтов персонажа, врагов и игровых объектов используется техника рисования “Pixel Art” (рис. 2.2).

Pixel Art – форма цифрового изображения, созданного на компьютере с помощью растрового графического редактора, где изображение редактируется на уровне пикселей (точек), а разрешение изображения настолько мало, что отдельные пиксели чётко видны [17].

Данная техника позволяет сохранить единую стилизацию игры и ускоряет процесс создания спрайтов.



Рисунок 2.2 – Пример графики в стиле Pixel Art из игры Starbound

Анимация в игре должна быть спрайтовой и храниться в спрайт листах (spritesheet), чтобы увеличить быстродействие игры (рис. 2.3).

Анимация – последовательный показ (слайд-шоу) заранее подготовленных графических файлов, а также компьютерная имитация движения с помощью изменения (и перерисовки) формы объектов или показа последовательных изображений с фазами движения.

Анимация должна соответствовать происходящим на сцене событиям и отражать реальное положение объектов, что приведет к большей отзывчивости управления, а значит, к лучшим ощущениям от игры.

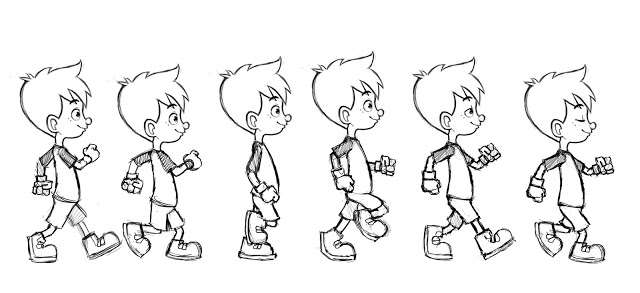


Рисунок 2.3 – Пример спрайтовой анимации

### 2.4.2 Игровой персонаж

Игровой персонаж – главный объект игры, обладающий некоторым набором индивидуальных параметров, контроль и управление которым осуществляет игрок.

Управление и взаимодействие персонажа с игровым миром будет осуществляться игроком с помощью специальных кнопок управления, отображаемых на экране смартфона. Логика работы кнопок заключается в том, чтобы при нажатии вызывать необходимый для движения персонажа метод. Каждый метод отвечает за отдельное действие или же за некоторый набор действий. Например, при нажатии кнопки бега должна проигрываться анимация бега.

Построенная логическая модель данных (рис. 2.4) в дальнейшем будет востребована на этапе физического проектирования, а также на этапе эксплуатации и сопровождении уже готового проекта, позволяя наглядно представить принцип управления персонажем.

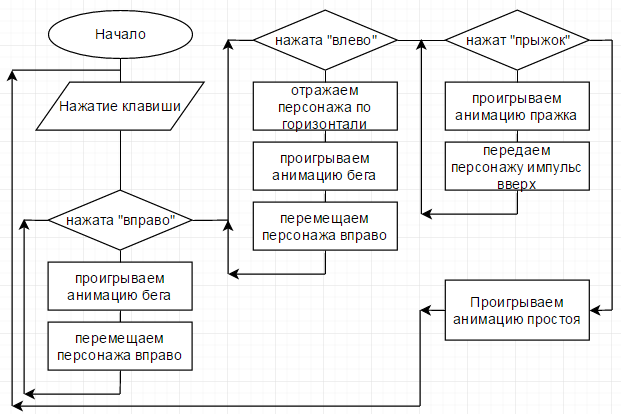


Рисунок 2.4 – Алгоритм управления персонажем

Игровой персонаж должен уметь выполнять такие действия как прыжок, бег и удар. При выполнении удара должно быть реализовано нанесение урона врагам и отнимание у врагов жизней. Все действия персонажа сопровождаются соответствующей анимацией.

На рисунке 2.5 представлена схема взаимодействия между анимациями персонажа.

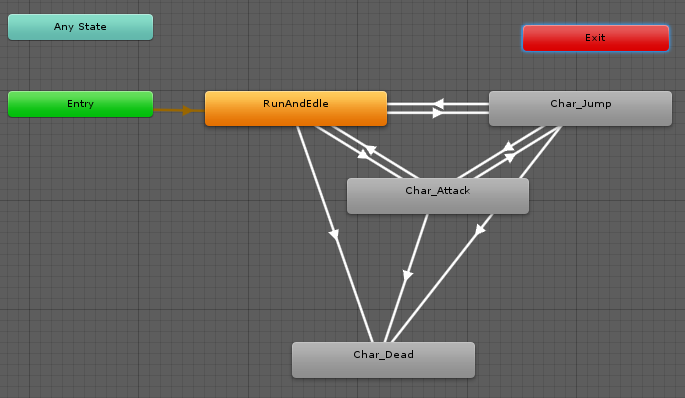


Рисунок 2.5 – Схема взаимодействия между анимациями персонажа.

Данная схема отображает все варианты переходов между одной анимацией к другой. Переходы осуществляются в зависимости от различных параметров изменяемых в зависимости от ситуации через программный код.

У игрока в начале уровня имеется максимально доступное количество здоровья. При получении атаки от врага у персонажа должно отниматься здоровье. Как только здоровье персонажа равно нулю игра завершается.

За уничтожение врагов игроку начисляются очки, которые в дальнейшем можно будет потратить на улучшение способностей персонажа.

Для корректного взаимодействия персонажа с игровыми объектами (ящиками, платформами и др.) у каждого объекта включая персонажа должен быть свой коллайдер отражающий физические размеры. Коллайдер (Collider) – компонент, определяющий форму объекта для физических столкновений [18].

Главному объекту на сцене (игроку) соответствует тег «Player», который будет необходим для определения игрока как противника для ботов игры.

При старте уровня игровой персонаж появляется в точке старта, если игрок достигает конца уровня, осуществляется подсчет набранных очков и предоставляется доступ к следующему уровню.

### 2.4.3 Игровые боты

В игре враги реализованы в виде ботов.

Бот – специальная программа, выполняющая автоматически и/или по заданному расписанию какие-либо действия через интерфейсы, предназначенные для людей [20].

Интеллект бота должен отвечать за своевременное обнаружение и атаку игрока. При потере игрока из поля видимости бот должен преследовать его еще какое-то время. При выполнении бота каких-либо действий должна проигрываться соответствующая анимация.

В основе противника лежит объект, к которому прикреплены 2 области:

* первая область, необходима для обнаружения игрока спереди;
* вторая область, отвечает за обнаружение игрока сзади.

Данные области реагирую на событие соприкосновения с другими объектами. Если объект с тегом «Player» взаимодействует с одной из областей, то происходит анализ того, с какой областью взаимодействует игрок и исходя из этого отдаются команды боту на конкретные действия (атака или преследование).

Бот имеет набор собственных параметров, отличающих его от персонажа игрока. Основным параметром является присвоенный ему тег «Monster», по данному параметру можно легко определить кем является персонаж в игре.

### 2.4.4 Игровой мир

Одной из основных задач, поставленных при разработке игры, была разработка механики случайной генерации игрового мира. Данная механика при новом запуске уровня должна генерировать новый ландшафт мира, что позволит разнообразить прохождение игры

За основу генерации мира был выбран псевдослучайный Шум Перлина.

Perlin noise (Шум Перлина, также иногда Классический шум Перлина) – математический алгоритм по генерированию процедурной текстуры псевдослучайным методом. Используется в компьютерной графике для увеличения реализма или графической сложности поверхности геометрических объектов. Также может использоваться для генерации эффектов дыма, тумана и т.д.

Шум Перлина – это градиентный шум, состоящий из набора псевдослучайных единичных векторов (направлений градиента), расположенных в определенных точках пространства и интерполированных функцией сглаживания между этими точками. Для генерации шума Перлина в одномерном пространстве необходимо для каждой точки этого пространства вычислить значение шумовой функции, используя направление градиента (или наклон) в указанной точке.

Функция «Perlin noise» проводит генерацию текстур методом генерации псевдослучайных чисел, однако все визуальные детали текстуры имеют одинаковый размер. Это свойство делает шум Перлина легко управляемым; множество масштабированных копий шума Перлина могут быть вставленными в математические выражения для создания самых разнообразных процедурных текстур.

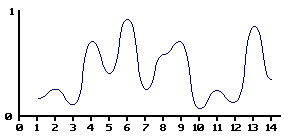


Рисунок 2.6 – Пример результата работы Шума Перлина для одномерного пространства

Алгоритм шума Перлина можно масштабировать одно-, двух- и трёхмерного вида. Более того, в алгоритм можно ввести четвёртое временное измерение, позволяя алгоритму динамически изменять текстуры во времени.

Шум Перлина был создан Кеном Перлином (англ.) в 1983 году и впоследствии был назван в честь своего создателя.

В игре использовалась функция Шума Перлина для одномерного пространства, но график результата ее работы сильно колеблется. Чтобы это исправить используется интерполяция.

Интерполяция, интерполирование (от лат. Inter-polis – «разглаженный, подновлённый, обновлённый; преобразованный») – в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений [21].

В игре использовался один из видов интерполяции – линейная интерполяция.

Линейная интерполяция – интерполяция алгебраическим двучленом (2.1) функции f, заданной в двух точках x0 и x1 отрезка [a, b]. В случае, если заданы значения в нескольких точках, функция заменяется кусочно-линейной функцией.

|  |  |  |
| --- | --- | --- |
|  | *P1(x) = ax + b* | (2.1) |

где a, b – значения между которыми производится интерполяция;

x – значение от 0 до 1.

Функция интерполяции возвращает значение между a и b в зависимости от значения х. Если х равен 0, то возвращается a, и при х = 1, то возвращается b. Если х находится между 0 и 1, то возвращает некоторое значение между a и b.

Процесс генерации мира можно разделить на следующие этапы:

1. Получение начальных значений (длина и высота мира);
2. Генерация начального ландшафта на основе Шума Перлина;
3. Добавление стартовой и финишной зоны;
4. Разделение ландшафта на платформы путем добавления обрывов;
5. Добавление точек появления монстров.
6. Производим заполнение платформ спрайтами.

Подробное описание каждого этапа генерации мира.

Получение начальных значений – перед началом генерации производится передача параметров мира (длины и высоты), это сделано с целью многократного использования алгоритма для построения различных уровней путем изменения данных параметров. Передача параметров происходит после выбора игроком конкретного уровня.

Генерация начального ландшафта на основе Шума Перлина – выполнение функции Перлина для одномерного массива равного длине уровня и запись полученных значений в двумерный массив. Выполнение интерполяции.

Добавление стартовой и финишной зоны – находим значения высот начальной и конечной точки массива и достраиваем до плоскости. Определяем центр у каждой зоны и добавляем точки начала и конца уровня.

Разделение ландшафта на платформы путем добавления обрывов – для получения количества обрывов, длину мира делим на платформы по 16 блоков. Далее с помощью функции рандомных чисел от 8 до 16 определяем длину платформы, а после определяем длину обрыва от 2 до 4 блоков. Повторяем до тех пор, пока не заполним весь ландшафт.

Добавление точек появления монстров – определяем центр каждой платформы и в полученном месте добавляем точку появления монстров.

Производим заполнение платформ спрайтами – производим проверку массива блоков мира и в зависимости от его положения блока присваиваем ему соответствующий спрайт.

### 2.4.5 Игровой счет

Для поощрения действий игрока была введена механика игрового счета. Данная механика позволяет накапливать очки за уничтожение врагов и тратить их на улучшение способностей персонажа.

### 2.4.6 Сохранение в игре

## 2.5 Проектирование пользовательского интерфейса

Интерфейс – общая граница между двумя функциональными объектами, совокупность средств, методов и правил взаимодействия (управления, контроля и т.д.) между элементами системы.

Для создания пользовательского интерфейса использовался инструмент межплатформенной среды Unity – UI System.

В основе работы UI в Unity лежат три компонента: Canvas, Rect Transform и Event Trigger.

Canvas – является контейнером для всех элементов UI и определяет режим рендера. Таких контейнеров на сцене может быть более одного. Главным параметром для отображения элементов игры и положения камеры является Render Mode. Значение данного параметра было выставлено в Screen Space – Camera. В данном режиме полотно рисуется на плоскости перпендикулярной взгляду камеры, на некотором расстоянии от точки взгляда. Размер полотна не меняется с изменением расстояния, оно всегда растягивается, чтобы заполнять разрез пирамиды видимости у камеры (camera frustrum view). Интерфейс будет заслоняться любыми 3D элементами, которые находятся перед плоскостью интерфейса.

Дополнительные параметры компонента Canvas:

* Pixel perfect – использовать ли antialiasing.
* Render camera – камера с помощью которой будет отображен интерфейс.
* Plane distance – дистанция плоскости нашего интерфейса от камеры.

Rect Transform – этот компонент позволяет задать положение и размер игрового объекта, используя удобные визуальные настройки. Он вводит понятия ширины и высоты и масштаба.

Unity Event – доработанная система событий, а конкретно компонент Event –> Event Trigger который включает компоненты визуального управления вызовом событий.

Интерфейс игры должен быть прост и понятен для пользователя.

Главным окном игры является главное меню (рис. 2.7), на котором пользователь может выполнить следующие действия:

* Начать игру – открыть окно выбора уровня (рис. 2.8);
* Выход – выход из игры.

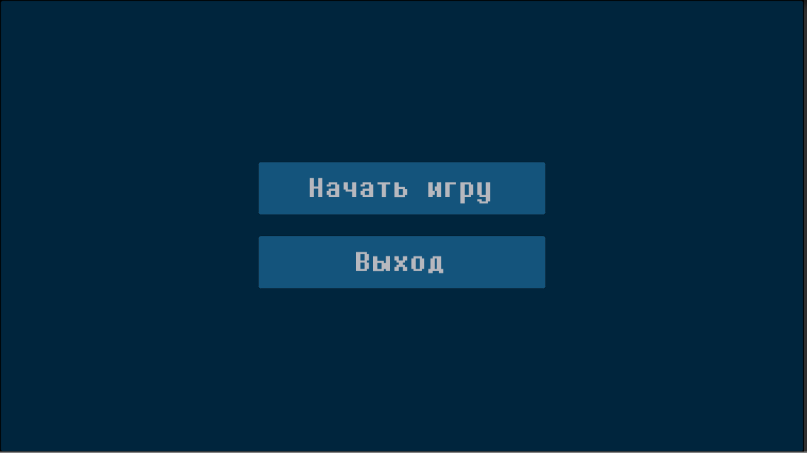


Рисунок 2.7 – Начальное окно

Ниже представлено окно выбора уровня (рис. 2.8), оно отображает доступные на данный момент для игры уровни. После выбора одного из уровней производится его загрузка.

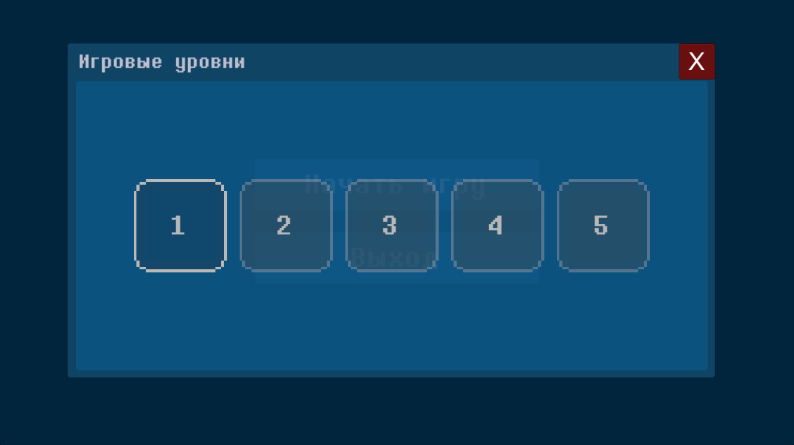


Рисунок 2.8 – Окно выбора уровня

Для отображения процесса загрузки используется Load Bar (рис. 2.9). Данное окно будет отображаться в момент перехода с одной сцены на другую.

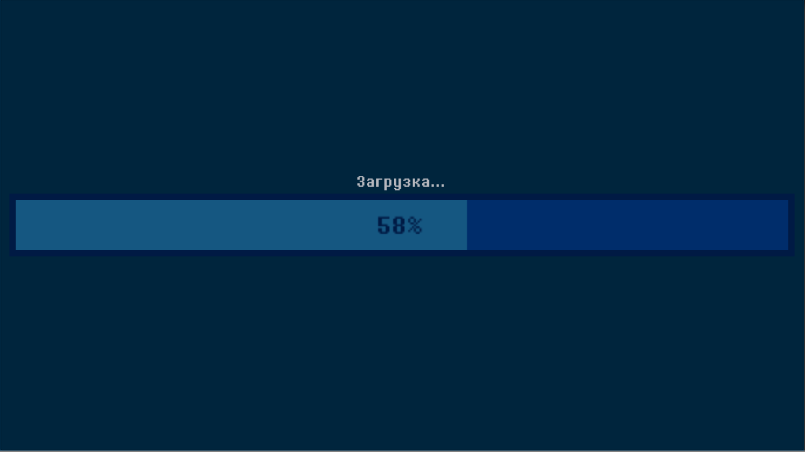


Рисунок 2.9 – Окно загрузки

Все элементы управления, основная информация и игровой процесс отображаются в основном окне (рис. 2.10).



Рисунок 2.10 – Основное игровое окно

## 2.6 Руководство пользователя

### 2.6.1 Общие сведения о программном продукте

Мобильная игра на основе 3D Unity в жанре «Платформер», представляет собой «Платформер» с элементами RPG игр, такими как развитием персонажа, система инвентаря и система очков. Основной особенностью игры является процедурно генерируемые уровни, что позволяет разнообразить прохождение.

Основной целью игры является прохождение уровней с уничтожением всех противников.

Предполагается, что программа будет использоваться пользователями в возрасте старше шести лет для хорошего времяпрепровождения, а также для развития тактических способностей.

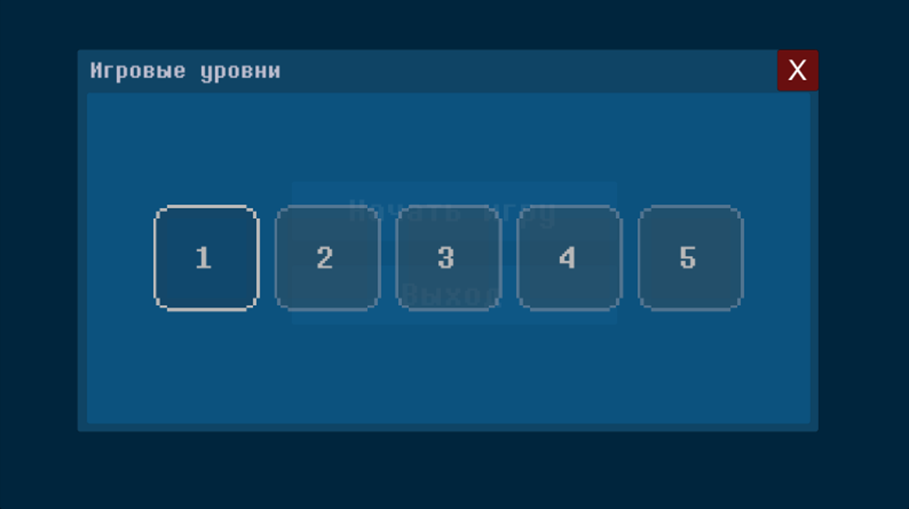
Для того чтобы установить игру на ваше устройство, необходимо запустить файл установки в формате .apk, подтвердить начало установки и после завершения игра будет готова к запуску.

Минимальные системные требования, наличие версии операционной системы Android 4.1 или выше.

### 2.6.2 Инструкция по игре

При входе в игру вам становится доступно главное меню (рис. 2.7), на котором вы можете перейти к панели доступных для прохождения уровней, нажав на кнопку «Начать игру» или же выключить приложение нажав на кнопку «Выход».

Элементы элементы окна выбора уровня представленны на рисунке 2.11.



Доступный уровень

Недоступные уровени

Рисунок 2.11 – Элементы элементы окна выбора уровня

Элементы игрового поля представленны на рисунке 2.12.



Live Bar отображающий кол-во жизней игрока

Кнопка игровой паузы

Кол-во набранных очков

Кнопка удара

Кнопка прыжка

Кнопки движения влево и вправо

Игровой персонаж

Противник

Рисунок 2.12 – Элементы игрового поля

## 2.7 Руководство программиста

При разработке данной версии программного продукта использовался игровой движок Unity 2017.3.1f1 Personal.

Основной игровой процесс завязан на объекте игрока, при удалении которого происходит множество ошибок скриптов, вызывающих параметры игрового персонажа. Данные ошибки не являются критическими, а лишь информируют нас о том, что параметры или объекты небыли найдены. Таким образом можно не боясь сломать проект удалять или редактировать компоненты не использующие параметры игрока.

В игровом мире большинство элементов реализованы как отдельные объекты со своими параметрами. Что позволяем манипулировать ими независимо друг от друга.

Изначально на сцене находятся не все объекты, а лишь их часть, ниже приведен список объектов, изначально находящихся на сцене.

* Игровой персонаж;
* Генератор мира;
* Счетчик игрового счета;
* Начальная и конечная точки уровня;
* Элементы интерфейса;
* Камера, отображающая сцену.

После запуска уровня изначально генерируется мир и точки появления монстров, корректируется положение игрока, начальной и конечной точки уровня.

Для добавления собственных спрайтов, тайлов и изображений, необходимо просто поместить их в папку Assets проекта. Для создания изображений можно использовать любой графический редактор.

Для добавления анимации необходимо также, как и изображение поместить ее в папку Assets проекта и настроить переходы через аниматор.

В настройках самой анимации можно добавить или редактировать флаги вызывающие различные методы в конкретный момент времени.

## 2.8 Информационная безопасность

### 2.8.1 Виды информационных атак на приложение

При разработке мобильного приложения следует учитывать, что данные, которыми оперирует это приложение, могут представлять определенный интерес для третьих лиц. Степень ценности этих данных варьируется в широких пределах, тем не менее, даже наиболее простая приватная информация, например, пароль входа в приложение, требует проработки ее защиты. Особенно это важно в свете распространения мобильных приложений на все сферы электронных услуг, включая финансовые, банковские операции, хранение и передачу личных данных и так далее [23].

Основные виды атак на мобильное приложение:

* Декомпиляция файла приложения (.apk-файлы для ОС Android) и разбор локально сохраненных данных. Защита этого, наиболее важного в настоящее время, уровня целиком лежит на плечах мобильного разработчика.
* Перехват данных, передаваемых по сети (MITM-атаки). Большинство мобильных приложений являются клиент-серверными, следовательно, постоянно передают и принимают большие объемы информации. И хотя современная мобильная и веб-разработка активно завершают переход на HTTPS-протокол общения, тем не менее, не стоит полагаться на единственный рубеж защиты в виде защищенного канала связи.
* Получение root-доступа к устройству и атака на приложение и применяемые в нем алгоритмы через внешние отладочные инструменты.

Перечень основных уязвимостей приложений

Рассмотрим уязвимости общего характера, без привязки к конкретной платформе. Здесь и далее используется аббревиатура КВД – критически важные данные пользователей. К КВД относятся любые данные, которые не должны быть доступны третьей стороне, это касается как персональных данных пользователя (дата рождения, адрес проживания, личная переписка), так и его приватных данных (пароли, данные кредитных карт, номера банковских счетов, номера заказов и так далее).

В перечень основных уязвимостей входят:

1. Использование незащищенных локальных хранилищ

Опасность: очень высокая.

Комментарий: встречается повсеместно, выражается в хранении КВД в незащищенных или слабо защищенных локальных хранилищах, специфических для конкретной платформы. Вскрытие третьей стороной – элементарное, и, как правило, не требуется наличие специальных навыков у атакующего.

Защита: хранить КВД можно только в защищенных хранилищах платформы.

1. Хранение КВД в коде.

Опасность: высокая.

Комментарий: уязвимость касается хранения КВД внутри кода (в статических константных строках, в ресурсах приложения и т.п.). Яркие примеры: хранение соли для пароля (password salt) в константе или макросе, которая применяется по всему коду для шифрования паролей; хранение приватного ключа для асимметричных алгоритмов; хранение паролей и логинов для серверных узлов или баз данных. Легко вскрывается третьей стороной при наличии базовых навыков декомпиляции.

Защита: не хранить никакие КВД в коде или ресурсах приложения.

1. Применение алгоритмов с хранением приватного ключа

Опасность: высокая.

Комментарий: уязвимость актуальна в случае, если приватная информация алгоритма (приватный ключ) вынужденно сохраняется в коде или ресурсах мобильного приложения (чаще всего так и бывает). Легко вскрывается методом декомпиляции.

Защита: в мобильной разработке желательно применять только современные симметричные алгоритмы с генерируемым случайным одноразовым ключом, обладающие высокой стойкостью к взлому методом грубой силы, либо выводить асимметричный приватный ключ за пределы приложения, либо персонализировать этот ключ (как пример – приватным ключом может выступать пользовательский код входа, сохраненный в зашифрованном виде в защищенном хранилище операционной системы).

1. Использование асимметричного алгоритма с приватным ключом, известным серверу.

Опасность: зависит от степени защищенности сервера.

Комментарий: уязвимость носит двойной характер. Хранение приватного ключа допускает возможность расшифровки пользовательских данных на стороне сервера. Во-первых, это некорректно с точки зрения безопасности (если сервер будет взломан – атакующий также получит доступ к приватным данным пользователей), а во-вторых, это нарушает приватность персональных данных. Пользователь всегда должен быть уверен, что его персональная информация не известна никому, кроме него самого (только если он явно не дал разрешение на ее публикацию). Часто приложения позиционируют себя как защищенные, но на деле таковыми не являются, так как содержат внутри себя средства для расшифровки персональной информации.

Защита: без явной необходимости и явного разрешения пользователя (чаще всего через лицензионное соглашение) ни приложение, ни сервер не должны иметь никакой возможности расшифровать приватные данные пользователя. Простейший пример – пароль пользователя должен уходить на сервер уже в виде хеша, и проверяться должен хеш, а не исходный пароль (серверу абсолютно незачем знать пользовательский пароль; если же пользователь его забыл – для такой ситуации существует давно отлаженный механизм восстановления пароля, в том числе с двухфакторной авторизацией клиента для повышенной безопасности процедуры восстановления).

1. Использование самописных алгоритмов шифрования и защиты.

Опасность: средняя.

Комментарий: это прямое нарушение принципа Керкгоффса. Выражается в попытке разработчика изобрести "свой личный, не известный никому, а поэтому супер-защищенный алгоритм шифрования". Любое отклонение от существующих, многократно проверенных и изученных, математически доказанных алгоритмов шифрования в 99% случаев оборачивается быстрым взломом подобной "защиты". Требует наличия средне-высоких навыков у атакующего.

Защита: следует подбирать подходящий алгоритм только из отлаженных и актуальных общеизвестных криптографических алгоритмов.

1. Передача КВД во внешнюю среду в открытом виде.

Опасность: средняя.

Комментарий: выражается в передаче КВД без применения шифрования по любому доступному каналу связи с внешней средой, будь то передача данных стороннему приложению или передача в сеть. Может быть вскрыто опосредованно путем вскрытия не приложения, а его хранилища, или целевого приложения. Взлом требователен к наличию навыков у атакующего, при условии, что хранилище является защищенным.

Защита: любые КВД перед выходом за пределы приложения должны быть зашифрованы. Локальные хранилища платформы не являются областью приложения, они тоже должны получать на вход только зашифрованные данные.

1. Игнорирование факта наличия рутованных или зараженных устройств.

Опасность: средняя.

Комментарий: рутованные устройства – это девайсы, где выполнена модификация для получения прав суперпользователя на любые операции, изначально запрещенные производителем операционной системы. Выполняется пользователем на своем устройстве самостоятельно, и не обязательно добровольно (клиент может быть не в курсе, что устройство взломано). Установка приложения на рутованный девайс нивелирует все штатные средства защиты операционной системы.

Защита: если это технически возможно для платформы – то желательно запрещать работу приложения, если удалось понять, что запуск производится на рутованном устройстве, или хотя бы предупреждать об этом пользователя (спасибо за дополнение DjPhoeniX).

1. Хранение КВД в защищенных хранилищах, но в открытом виде.

Опасность: средняя.

Комментарий: разработчики зачастую склонны сохранять КВД в защищенные системные хранилища без дополнительной защиты, поскольку системные механизмы хорошо сопротивляются взлому. Однако уровень их стойкости падает до минимума в случае, если устройство рутованное.

Защита: КВД не должны использоваться в приложении без дополнительного шифрования. Как только надобность в "открытых" КВД отпала – они немедленно должны быть либо зашифрованы, либо уничтожены.

1. Перевод части функционала во встроенные веб-движки.

Опасность: средняя.

Комментарий: чаще всего выглядит как передача КВД во встроенный браузер, где загружается внешняя веб-страница, выполняющая свою часть функционала. Уровень защиты в этом случае резко снижается, особенно для рутованных устройств.

Защита: не использовать встроенный браузер и встроенный веб-движок в операциях с КВД. На крайний случай – шифровать КВД перед передачей.

1. Реверсивная инженерия алгоритмов, представляющих интеллектуальную ценность.

Опасность: низкая, зависит от ценности алгоритма.

Комментарий: если при разработке приложения внутри компании используются некие собственные алгоритмы, которые могут представлять высокую ценность для потенциальных конкурентов или взломщиков, то эти алгоритмы должны быть защищены от постороннего доступа.

Защита: автоматическая или ручная обфускация кода.

Обфускация (от лат. obfuscare – затенять, затемнять; и англ. obfuscate – делать неочевидным, запутанным, сбивать с толку) или запутывание кода – приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

### 2.8.2 Информационная безопасность приложения

Для защиты своего умственного труда от мошенников, пытающихся украсть код, графические элементы или создать себе преимущества в игре нечестным путем, были произведены следующие меры:

* Защита сохраненных данных игры;
* Защита исходного кода игры;
* Защита памяти игры от программ (Cheat Engine, Art Money, Game Guardian, Game Killer и т.д).

Защита сохраненных данных игры – для обеспечения защиты сохраненных данных игры был введен метод шестнадцатеричного шифрования. Данный метод позволяет преобразовывать данные в шестнадцатеричные значения, что делает их бесполезными без расшифровки.

Защита исходного кода игры – для защиты исходного кода игры от посторонних глаз использовался плагин для Unity под названием Obfuscator.

Плагин Obfuscator был разработан для повышения безопасности программного обеспечения и игр, созданных с помощью Unity.

Плагин выполняет функция обфускации встроенных сборок .Net, таких как assembly-csharp и assembly-unityscript для автономных игр Windows/Mac/Linux и встроенных систем, таких как Android и IPhone. Obfuscator рассматривает специфические функции Unity, такие как MonoBehaviour, NetworkBehaviours, Serialization, Reflection и т.д., чтобы обеспечить простую и запутанную обфускацию.

Защита памяти игры от программ –

## 2.9 Вывод по главе 2

В главе «Проектные решения при создании мобильной игры в жанре «Платформер»» был произведен выбор основных программных средств, используемых для разработки дипломного проекта. Также был произведен этап проектирования приложения, в котором был разработан алгоритм работы игры и построены основные UML диаграммы.

В результате проведенного этапа моделирования были определены основные концепции игры, такие как генерация мира, управление персонажем и др. В последствии был спроектирован пользовательский интерфейс и определена степень защиты приложения от информационных атак.

# Экономическая часть

## Расчет трудоемкости разработки программного продукта

Трудоемкость – количество рабочего времени человека, затрачиваемого на производство единицы продукции.

Трудоемкость программного продукта – это зависимость количества массовых операций (сравнения, обмены, сдвиги, повторения цикла и т.п.) от объема обрабатываемых данных, отражает общее количество затрат на разработку в зависимости от сложности проекта.

Перечень стадий, этапов и видов работ необходимых для расчета трудоемкости программного продукта:

* Техническое задание – изучение и анализ предметной области;
* Эскизный и технический проект – выбор проектных решений, определение перечня входных и выходных документов, разработка структуры входных и выходных данных;
* Рабочий проект – разработка структуры БД и программирование, комплексная отладка программы;
* Расходы на отладку программного продукта – изменение программы с учётом текущих пожеланий пользователей
* Внедрение – подготовка документации.

Трудоемкость разработки программного обеспечения в чел.час определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Т = То+Ти+Та+Тп+Тотл+Тд | (3.1) |

где Т – общие затраты труда, чел. час.

То – затраты труда на описание задачи

Ти – затраты на исследование предметной области

Та – затраты на разработку алгоритма решения задачи

Тп – затраты на программирование

Тотл – затраты на отладку программы

Тд – затраты на подготовку документации

Все составляющие определяются через условное число операторов Q:

|  |  |  |
| --- | --- | --- |
|  | Q=q\*c\*(1+p) | (3.2) |

где q – число операторов (оператор – предписание о выполнении конкретных действий. Например, в языке программирования Паскаль содержится 11 операторов.)

c – коэффициент сложности задачи (принимается от 1,25 …. 2)

p – коэффициент коррекции программы, учитывающий новизну проекта (для совершенно новой программы равен 0,1)

Q = 850\*1,6\*(1+0,1) = 1496

То – принимаем в диапазоне от 30 до 40 чел.час (в зависимости от сложности продукта)

Ти – определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Ти = (Q\*B) / (Sи\*k) | (3.3) |

где В – коэффициент увеличения затрат труда в следствие недостаточного описания задачи (1,2 – 1,5)

Sи – количество операторов, приходящихся на 1 чел. час ( 75-85 )

k – коэффициент квалификации работника (табл. 3.1)

Таблица 3.1. -Квалификация работников

|  |  |
| --- | --- |
| Стаж | Коэффициент |
| до 2-х лет | 0,8 |
| 2- 3 года | 1,0 |
| 3-5 лет | 1,1 – 1,2 |
| 5 – 7 лет | 1,3 – 1,4 |
| свыше 7 лет | 1,5 – 1,6 |

Ти = (1496\*1,2) / (75\*1,2) = 19,95 чел. час

Та рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  | Та = Q / (Sа\*k) | (3.4) |

Sа – количество операторов на разработку алгоритма решения задачи, приходящихся на 1 чел.час ( 20 – 25)

Та = 1496 / (20\*1,2) = 62 чел.час

Затраты труда на программирование найдем по формуле:

|  |  |  |
| --- | --- | --- |
|  | Тп = Q / (Sп\*k) | (3.5) |

Sп  – количество операторов на программирование, приходящихся на 1 чел.час (20 – 25)

Тп = 1496 / (20\*1,2) = 62 чел.час

Затраты труда на отладку программы находим по формуле:

|  |  |  |
| --- | --- | --- |
|  | Тотл = Q / (Sотл\*k) | (3.6) |

Sотл  – количество операторов на отладку программы, приходящихся на 1 чел.час (4 – 5)

Тотл = 1496 / (4\*1,2) = 311,66 чел.час

Затраты на подготовку документации по задаче определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  | Тд = Тдр + Тдо | (3.7) |

где Тдр – затраты труда на подготовку материала рукописи;

Тдо – затраты труда на редактирование, печать и оформление документов

|  |  |  |
| --- | --- | --- |
|  | Тдр = Q / (Sдр\*k) | (3.8) |

Sдр  – количество операторов на подготовку материала рукописи, приходящихся на 1 чел.час (15 – 20)

Тдр = 1496 / (17\*1,2) = 73,33 чел.час

Тдо определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Тдо = 0,75 \* Тдр | (3.9) |

Тдо = 0,75 \* 73,33 = 55 чел.час

Тд = 73,33 + 55 = 128,33 чел.час

Т = 35 + 21,6+ 62 + 62+ 311,66 + 128,33 = 620,59 чел. час.

Полученное значение общей трудоемкости необходимо скорректировать с учетом уровня языка программирования:

|  |  |  |
| --- | --- | --- |
|  | Ткор = Т\* kкор | (3.10) |

где kкор – коэффициент учитывающий уровень языка программирования (0,8 – 1)

Ткор = 620,59\* 0,8 = 496,47 чел.час.

Заработная плата – это совокупность вознаграждений в денежной или натуральной форме, полученных работниками за фактически выполненную работу, а также за периоды, включаемые в рабочее время.

Виды заработной платы:

* Номинальная – количество денег в номинальном размере, которое получает работник в виде вознаграждения за труд.
* Реальная – это количество товаров и услуг, которое можно приобрести на номинальную заработную плату.
* Основная зарплата – это оплата отработанных часов, или выполненного задания, а величина выплат соответствует установленным нормам труда (например, тарифные ставки или оклады).
* Дополнительная зарплата – это оплата за сверхурочный труд, различные трудовые успехи и надбавки за особые условия труда.
* Формы и системы заработной платы:
* Повременная – это форма заработной платы, при которой заработная плата зависит от количества фактически отработанного затраченного времени, учитывая квалификации работника и условий труда.
* Сдельная – форма заработной платы, при которой оплата зависит от количества произведенных единиц продукции, учитывая их качества, сложности и условий труда.

Величина заработной платы для отдельного сотрудника рассчитывается в конце месяца и на прямую зависит от величины тарифной ставки.

Тарифная ставка – это размер денежной выплаты в составе заработной платы, который выплачивается работнику за выполнение нормы труда определённой сложности за установленное время без учёта компенсационных, стимулирующих и социальных выплат.

Часовая тарифная ставка инженера технолога определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Тст = МРОТ/Вф | (3.11) |

где МРОТ – минимальный размер оплаты труда на 21.05.2018г. (оклад сотрудника), равный 11163 руб./мес.

Вф – фактически отработанное время (168 час. в месяц, т.е 21 смена \*8 часов)

Тст = 12837 / 168 = 76,41 руб.

Время работы персонального компьютера при создание программного продукта определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  | Фв = 1,15 \* (Тп + Тд + Тотл)\*kкор | (3.12) |

Фв = 1,15 \* (45,83 + 99,09 + 240,63) \* 0,8 = 354,71час.

Себестоимость продукции (работ, услуг) – это стоимостная оценка используемых в производстве продукции (работ, услуг) природных ресурсов, сырья, материалов, топлива, энергии, основных фондов, трудовых ресурсов и других затрат на её производство и сбыт.

Затраты, образующие себестоимость продукции (работ, услуг) группируются в соответствии с их экономическим содержанием по следующим элементам:

* Материальные затраты (за вычетом стоимости возвратных отходов);
* Затраты на оплату труда;
* Отчисления на социальные нужды;
* Амортизация основных средств;
* Прочие затраты.

Калькулирование – это система расчетов, с помощью которых определяется себестоимость всей товарной продукции и её частей, себестоимость конкретных видов изделий, сумма затрат отдельных подразделений предприятия на производство и реализацию продукции.

Калькуляция себестоимости необходима для определения цены единицы продукции и уровня эффективности работы предприятия.

Таблица 3.2 – Затраты на разработку программного продукта

|  |  |  |
| --- | --- | --- |
| Наименование статьи | Формула для расчета | Сумма затрат |
| Основная ЗП с учетом коэффициента корректировки | ЗПо = (Тотл + Тп) \* kкор \*Тст + (Тдр +Та) kкор \*Тст + (Тп +Ти +То) kкор \*Тст = (311,66 + 62) \* 0,8 \* 76,41 + (73,33 + 62) \* 0,8 \* 76,41 + (62 + 19,95 + 35) \*0,8 \* 76,41=38262,46 руб. | 38262,46 руб. |
| Дополнительная заработная плата | ЗПд = 0,2 \* ЗПо = 0,2 \* 38262,46 = 7652,49 руб. | 7652,49 руб. |
| Отчисления на социальные нужды | ЕСН = (ЗПо+ЗПд)\*0,3 | 13774,49 руб. |
| Расходы на материалы и запчасти | Зм от 650 руб. до 1000 руб  . | 1000 руб. |
| Наименование статьи | Формула для расчета | Сумма затрат |
| Затраты на ТО и ТР | 2% от балансовой стоимости оборудования  34500 \* 0,02 = 690 руб. | 690 руб. |
| Затраты на потребляемые энергоресурсы | Зэ = 0,05 кВт\ч \* 3,03 руб. \* 461,83 ч. = 50 руб. | 70 руб. |
| Итого (Зи): | – | 61449 руб. |

Стоимость услуг за разработку программного продукта по договору с покупателем 76811 руб.

Прибыль – положительная разница между суммарными доходами (в которые входит выручка от реализации товаров и услуг, полученные штрафы и компенсации, процентные доходы и т. п.) и затратами на производство или приобретение, хранение, транспортировку, сбыт этих товаров и услуг.

Определим прибыль от реализации программного продукта заказчику:

Пр = Вр – Зи, (13)

где Вр – выручка от реализации программного продукта (деньги от заказчика)

Зи – итоговая сумма затрат на разработку

Пр = 76811 – 61449 = 15362 руб.

# Заключение

# Список используемых источников

1. <http://appcraft.pro/blog/kak-ios-i-android-razdelili-mobilnyi-rynok/>
2. <http://fb.ru/article/249494/chto-takoe-arkada-podrobnyiy-razbor>
3. <https://ru.wikipedia.org/wiki/Action>
4. <https://ru.wikipedia.org/wiki/Гоночная_игра>
5. <https://ru.wikipedia.org/wiki/Компьютерная_стратегическая_игра>
6. <https://ru.wikipedia.org/wiki/Классификация_компьютерных_игр#Симуляторы/Менеджеры>
7. [https://ru.wikipedia.org/wiki/»Платформер»](https://ru.wikipedia.org/wiki/Платформер)
8. <http://4pda.ru/2015/06/10/226223/>
9. <https://gamer-info.com/game/meganoid/>
10. <https://games.mail.ru/pc/articles/review/dan_the_man/>
11. <https://ru.wikipedia.org/wiki/Cuphead>
12. <https://ru.wikipedia.org/wiki/Adobe_Photoshop>
13. <https://ru.wikipedia.org/wiki/Unity_(игровой_движок)>
14. <https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio>
15. <https://ru.wikipedia.org/wiki/C_Sharp>
16. <http://upr-proektom.ru/proektirovanie>
17. <https://ru.wikipedia.org/wiki/Пиксельная_графика>
18. <https://docs.unity3d.com/ru/530/Manual/CollidersOverview.html>
19. <https://ru.wikipedia.org/wiki/Диаграмма_состояний_(UML)>
20. <https://ru.wikipedia.org/wiki/Бот_(программа)>
21. <https://ru.wikipedia.org/wiki/Интерполяция>
22. <http://appwarm.com/ru/10126-lode-runner-1.html>
23. <https://habr.com/post/327760/>

# Приложение А

**(обязательное)**

**C# код**

Листинг 1 скрипт игрового персонажа

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/// <summary>

/// Главный скрипт игрока, овечает за действия и управление

/// </summary>

public class Player : PlayerMotor {

public float maxSpeed = 10f; // макс. скорость персонажа

public float jumpForce = 5f; // сила прыжка персонажа

public float backValue = 1.2f; // компенсирование смещения игрока при смене направления движения

public bool isFacingRight = true; // направление персонажа вправо/влево

public float attackDamage = 1; // урон при атаке

public float timeEffect = 0.3f; // время эффекта повреждения

private bool isGrounded; // находится ли персонаж на земле

private bool AttackAnimation;

private float speedX; // переменная запоминающая направление движения

private Rigidbody2D thisRigidbody;

public Transform thisTransform;

private Animator thisAnimator;

private SpriteRenderer thisSpriteRender;

public CharacterHp playerHp; // ссылка на компонент Hp игрока

public bool playerIsLive = true; // показывает жив ли игрок

public CharactersMotor charactersMotor; // ссылка на компонент действий игрока

private AttackArea attackArea; // область атаки игрока

private bool canAttack = false;

void Start()

{

thisSpriteRender = GetComponent<SpriteRenderer>();

thisRigidbody = GetComponent<Rigidbody2D>();

thisTransform = transform;

thisAnimator = GetComponent<Animator>();

playerHp = GetComponent<CharacterHp>();

attackArea = FindObjectOfType<AttackArea>();

}

void Update()

{

Dead();

}

void FixedUpdate()

{

Run(thisTransform, speedX);

playerIsLive = playerHp.CharIsLive();

thisSpriteRender.color = Color.red;

canAttack = attackArea.OnColisionTrue;

charactersMotor = attackArea.childrelCharMotor; // получаем компонент монстра если он находится а зоне атаки

}

public void LeftButtonDown() // движение влево

{

speedX = -maxSpeed;

isFacingRight = ModelTurn(thisTransform, isFacingRight, speedX, backValue);

thisAnimator.SetFloat("Speed", Mathf.Abs(speedX)); //Вкл. анимацию бега

}

public void RightButtonDown() // движение вправо

{

speedX = maxSpeed;

isFacingRight = ModelTurn(thisTransform, isFacingRight, speedX, backValue);

thisAnimator.SetFloat("Speed", Mathf.Abs(speedX));//Вкл. анимацию бега

}

public void JumpButtonDown() // прыжок

{

if (isGrounded == true)

{

thisRigidbody.AddForce(new Vector2(0, jumpForce), ForceMode2D.Impulse);

thisAnimator.SetBool("Jump", true);

}

}

public void StopPlayer()

{

speedX = Stop();

thisAnimator.SetFloat("Speed", Mathf.Abs(speedX)); // Выкл. анимацию бега

}

private void OnCollisionStay2D(Collision2D collision) // Проверяем находится ли игрок на земле, если да, то разрешаем прыжок и выкл. анимацию прыжка

{

if (collision.gameObject.tag == "Ground")

{

isGrounded = true;

thisAnimator.SetBool("Jump", false);

}

}

private void OnCollisionExit2D(Collision2D collision) // Если игрок перестает касаться земли то запрещаем прыжок

{

if (collision.gameObject.tag == "Ground")

{

isGrounded = false;

}

}

public override void Attack()

{

speedX = Stop();

thisAnimator.SetFloat("Speed", 0);

thisAnimator.SetBool("Attack", true);

Invoke("AnimationAttackFalse", 0.45f); // Выкл. анимации удара

}

// Выключение анимации удара, вызов из др. метода

public void AnimationAttackFalse()

{

thisAnimator.SetBool("Attack", false);

}

// Нанесение урона монстру, вызывается из анимации

public override void SetDamage()

{

if (canAttack) // если враг в зоне атаки

{

charactersMotor.GetDamage(attackDamage);

}

}

// Получение урона

public override void GetDamage(float damage)

{

if (playerIsLive)

{

playerHp.GetDamage(damage);

DamageEffect(thisSpriteRender, thisRigidbody); // Эффект получения урона

Invoke("DamageOff", timeEffect); // Выкл. эффекта

}

}

// Вызываем выключение эффекта,

private void DamageOff()

{

OffDamageEffect(thisSpriteRender);

}

// Смерть персонажа

public override void Dead()

{

if (!playerIsLive)

{

speedX = 0;

thisAnimator.SetBool("Dead", true);

}

}

//Удаляем персонажа из сцены, вызывается из анимации смерти

public void CharDestroy()

{

Destroy(gameObject);

}

}

Листинг 2 скрипт генерации мира

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/// <summary>

/// Скрипт генерации мира

/// </summary>

public class GeneratorWorld : MonoBehaviour

{

#pragma warning disable 78

public GameObject dirtPrefab; // Земля

public GameObject surfacePrefab; // Поверхность (камни)

public GameObject rightSurfacePrefab; // Поверхность справа

public GameObject leftSurfacePrefab; // Поверхность слева

public GameObject rightUpSurfacePrefab; // Угловоя поверхность справа

public GameObject leftUpSurfacePrefab; // Угловоя поверхность слева

public Transform startPoint; // Место появления игрока

public Transform endPoint; // Конец уровня

public GameObject enemySpawnPoint;

public GameObject savePoint;

public GameObject deadBreakage;

public int StartingAreaLenght = 1;

public int EndAreaLenght = 1;

// Размеры мира

int minX = 0;

int maxX = 128;

int minY = 0;

int maxY = 20;

public PerlinNoise noise;

GameObject[,] arrayBlocks;

public LevelController levelController;

public GameObject Settings;

private float width = 0.245f;

private float height = 0.245f;

void Start()

{

levelController = FindObjectOfType<LevelController>();

//maxX = (int)levelController.lenghtLevel;

arrayBlocks = new GameObject[maxX - minX, maxY - minY];

noise = new PerlinNoise(Random.Range(1000000, 10000000)); //кол-во иттераций

Regenerate();

StartingArea();

EndArea();

Green1Block();

BreakageGeneration();

FillingSprites();

AddSavePoint();

DeadBreakage();

}

// Генератор мира с помощью шума Перлина

private void Regenerate()

{

for (int i = minX; i < maxX; i++) //Колонка (x values)

{

int columnHeight = 5 + noise.getNoise(i - minX, maxY - minY - 5); //Высота мира (колонки Y)

for (int j = minY; j < minY + columnHeight; j++) //Значение высоты колонки (y values)

{

GameObject block = (j == minY + columnHeight - 1) ? surfacePrefab : dirtPrefab;

arrayBlocks[i, j] = Instantiate(block, new Vector2(i \* width, j \* height), Quaternion.identity);

}

}

}

// Тестовый

private void Green1Block()

{

for (int i = minX; i < maxX; i++) //Колонка (x values)

{

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if (arrayBlocks[i, j] != null)

{

SpriteRenderer colores = arrayBlocks[i, j].GetComponent<SpriteRenderer>();

colores.color = Color.white;

}

}

}

}

// Стартовая зона

private void StartingArea()

{

// Определяем высоту первого блока мира

int FirstColumnHeight = 0;

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if (arrayBlocks[minX, j] == null)

{

FirstColumnHeight = j;

break;

}

}

for (int i = 0; i < StartingAreaLenght; i++) //Колонка (x values)

{

for (int j = 0; j < FirstColumnHeight; j++) //Значение высоты колонки (y values)

{

GameObject block = (j == minY + FirstColumnHeight - 1) ? surfacePrefab : dirtPrefab;

Instantiate(block, new Vector2(i \* -width, j \* height), Quaternion.identity);

}

if (i == Mathf.Floor(StartingAreaLenght / 4))

{

Transform block = startPoint;

block.position = new Vector3(i \* -width, FirstColumnHeight \* height, 0);

}

}

}

// Конечная зона

private void EndArea()

{

// Определяем высоту последнего блока мира

int LastColumnHeight = 0;

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if (arrayBlocks[maxX - 1, j] == null)

{

LastColumnHeight = j;

break;

}

}

for (int i = 0; i < EndAreaLenght; i++) //Колонка (x values)

{

for (int j = 0; j < LastColumnHeight; j++) //Значение высоты колонки (y values)

{

GameObject block = (j == minY + LastColumnHeight - 1) ? surfacePrefab : dirtPrefab;

Instantiate(block, new Vector2((i + maxX) \* width, j \* height), Quaternion.identity);

}

if (i == Mathf.Floor(EndAreaLenght / 2))

{

Transform block = endPoint;

block.position = new Vector3((i + maxX) \* width, LastColumnHeight \* height, 0);

}

}

}

// Генерация обрывов в мире

private void BreakageGeneration()

{

int numberIneration = (int)Mathf.Floor(maxX / 16); // Кол-во разрывов в мире

int column = 0; // Размеченное растояние

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

{

int randomLenght = Random.Range(8, 16); // Длина поверхности между обрывами

column += randomLenght;

int randomPit = Random.Range(2, 4); // Длина обрывами

for (int i = 0; i < randomPit; i++) // Колонка (x values)

{

for (int j = minY; j < maxY; j++) // Значение высоты колонки (y values)

{

if (arrayBlocks[i + column, j] != null)

{

Destroy(arrayBlocks[i + column, j]);

arrayBlocks[i + column, j] = null;

}

}

}

// Добавляем спавнеры монстров

if (k != 1)

{

int positionEnemySpawnPoint = (int)Mathf.Floor(randomLenght / 2);

int p = column - positionEnemySpawnPoint;

for (int j = minY; j < maxY; j++) // Значение высоты колонки (y values)

{

if ((arrayBlocks[p, j] == null) && (arrayBlocks[p, j - 1] != null))

{

GameObject block = enemySpawnPoint;

Instantiate(block, new Vector2(p \* width, j \* height + 1f), Quaternion.identity);

}

}

}

column += randomPit;

}

}

// Заполнение спрайтами

private void FillingSprites()

{

// Если пустое пространство справа

for (int i = minX; i < maxX - 1; i++) //Колонка (x values)

{

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if ((arrayBlocks[i, j] != null) && (arrayBlocks[i + 1, j] == null))

{

Destroy(arrayBlocks[i, j]);

GameObject block = (arrayBlocks[i, j + 1] == null) ? rightUpSurfacePrefab : rightSurfacePrefab;

arrayBlocks[i, j] = Instantiate(block, new Vector2(i \* width, j \* height), Quaternion.identity);

}

}

}

// Если пустое пространство слева

for (int i = minX + 1; i < maxX; i++) //Колонка (x values)

{

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if ((arrayBlocks[i, j] != null) && (arrayBlocks[i - 1, j] == null))

{

Destroy(arrayBlocks[i, j]);

GameObject block = (arrayBlocks[i, j + 1] == null) ? leftUpSurfacePrefab : leftSurfacePrefab;

arrayBlocks[i, j] = Instantiate(block, new Vector2(i \* width, j \* height), Quaternion.identity);

}

}

}

}

// Добавляем точки сохранения

private void AddSavePoint()

{

int countSavePoint = (int)Mathf.Floor(maxX / 96); // Кол-во сохранений на уровне

Debug.Log("countSavePoint = " + countSavePoint);

int distanceToSave = maxX / (countSavePoint + 1); // дистанция до точки сохранения

// Размещение точек сохранения на уровне

for (int n = 0; n < countSavePoint; n++)

{

int i = distanceToSave \* (n + 1);

while (arrayBlocks[i, 0] == null) // Выполняем если точка сохранения приходится на обрыв

{

i += 4;

}

// Поиск положения и добавление точки сохранения на сцену

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if ((arrayBlocks[i, j] == null) && (arrayBlocks[i, j - 1] != null))

{

GameObject block = savePoint;

Instantiate(block, new Vector2(i \* width, j \* height - 0.02f), Quaternion.identity);

}

}

}

}

// Область при падении в которую игрок погибает

private void DeadBreakage()

{

for (int i = minX; i < maxX; i++) //Колонка (x values)

{

for (int j = minY; j < maxY; j++) //Значение высоты колонки (y values)

{

if ((arrayBlocks[i, j] == null) && (j == 4))

{

GameObject block = deadBreakage;

Instantiate(block, new Vector2(i \* width, j \* height), Quaternion.identity);

}

}

}

}

}