МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-12 Информационные системы и технологии (в игровой индустрии)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Разработка игровой концепции»

на тему: «Разработка игровой документации к компьютерной игре в жанре *Action*-*Adventure*»

Исполнитель: студент группы ИТИ-31

Ляпин В.А.

Руководитель: преподаватель-стажёр

Малиновский И.Л.

Дата проверки:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссиипо защите курсового проекта:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2021

**СОДЕРЖАНИЕ**

[Введение……………………………………………………………………………...4](#_Toc89262288)[1 Анализ игровых жанров и средств разработки компьютерных игр……………5](#_Toc89262289)

[1.1 История развития компьютерных игр как индустрии на различных игровых платформах…………………………………………………………………...5](#_Toc89262290)

[1.2 История развития «*Action*-*Adventure*» как самостоятельного игрового жанра……. .6](#_Toc89262291)

[1.3 Игровые движки как основное средство разработки игрового приложения………………………………………………………………..…………….……..7](#_Toc89262292)

[2 Определение основных этапов в разработке игрового приложения………….11](#_Toc89262294)

[2.1 Анализ рынка игровых приложений……………………………………11](#_Toc89262295)

2.2 Анализ целевой аудитории жанра «*Action-Adventure*»…………..…….13

2.3 Цикл работы игрового приложения…………………………………….14

[3 Программная реализация игрового приложегния……………………………...16](#_Toc89262302)

[3.1 Разработка графической составляющей игры 16](#_Toc89262303)

[3.2 Разработка программного составляющего игры 19](#_Toc89262304)

[3.3 Эксплуатация игры 22](#_Toc89262306)

[3.4 Интерфейс игры 24](#_Toc89262307)

[Заключение 27](#_Toc89262308)

[Список](#_Toc89262309) используемых источников………………………………………………...28

[Приложение А Листинг программы 29](#_Toc89262310)

[Приложение Б Руководство пользователя 38](#_Toc89262311)

[Приложение В Руководство программиста 40](#_Toc89262312)

[Приложение Г Руководство системного программиста 41](#_Toc89262313)

[Приложение Д Схема навигации по меню игрового интерфейса и оболочки](#_Toc89262314) игры………………………………………………………………………………….42

**ВВЕДЕНИЕ**

В 21 веке, в свете непрерывного развития высоких технологий, именно игровая индустрия стала проводником идей творчества и новаторства. В досуге современного человека компьютерные игры занимают достаточно важное место и давно перестали позиционироваться как поверхностное развлечение. Для многих игры – это вообще повседневное развлечение. 2021 год показал насколько игры важны для людей и перспективны для развития. Для сравнения, объем рынка игр уже обогнал другие традиционные развлечения, такие как музыка и фильмы вместе взятые.

Поэтому далее рассмотрим более сложную и не всем знакомую часть игр, это их разработка. В процессе выполнения работы необходимо разработать дизайн документ и разработать графические материалы для игры, который будет отражать все основные элементы игрового жанр *«Action-Adventure»*. *«Action-Adventure»*, он же – приключенческий боевик называют гибридным жанр компьютерных игр, где сочетаются квест и боевик. В таких играх игроку предстоит не только ломать голову над головоломками и сюжетными задачами, но и проверить свою реакцию или выносливость.

При разработке игрового приложения необходимо использовать программное обеспечение *Unity*. *Unity* является современным кросс-платформенным движком для создания игр и приложений, разработанный *Unity* *Technologies*.

Исходя из вышеперечисленного сделан вывод, что работа над данным проектом является весьма актуальной в нынешнее время. В результате разработки игрового приложения будут получены востребованные в наше время знания и освоены и применены на практике различные технологии и средства разработки игровых приложений.

В результате выполнения работы будет разработано игровое приложение на *Unity*, которое будет отражать все основные элементы разрабатываемого игрового приложения в жанре *«Action-Adventure»*. Игровое приложение будет содержать игровое меню, различные сцены, уникальные графические материалы и собственный сюжет.

**1 АНАЛИЗ ИГРОВЫХ ЖАНРОВ И СРЕДСТВ РАЗРАБОТКИ КОМПЬЮТЕРНЫХ ИГР**

**1.1 История развития компьютерных игр как индустрии на**

**различных игровых платформах**

В далеком 1978 году коммерчески успешная игра *Space* *Invaders* сильно мотивировала крупных инвесторов обратить внимание на игровую индустрию. Эта видеоигра была выпущена в Японии на платформе, которую мы знаем под названием аркадные аппараты. Игра стала настолько популярной, что вызвала в стране дефицит монет, которые требовались для того, чтобы поиграть в нее. Благодаря *Space* *Invaders* начался бум развития аркадных аппаратов, ставших первой ступенью эволюции индустрии игр.

Вместе с игровыми аппаратами игры развивались на платформе игровых приставок. И хотя история приставок началась еще в 1972 году вместе с релизом первой в мире домашней игровой приставки *Magnavox* *Odyssey*, прорыв в этом направлении начинается в 1977 году, когда в продажу поступает игровая приставка *Atari* 2600. Благодаря инвестициям *Atari* популяризация компьютерных и видеоигр выходит на новый уровень. Появляются новые крупные игроки, желающие занять теплое место на рынке видеоигр. В качестве крупнейших кроме *Atari* в историю вошли *Sega*, *Nintendo*, *Sony* и *Microsoft*.

Другой крупной игровой платформой «с древнейших времен» стали персональные компьютеры. Первые игры на них появились еще в 1960-х годах. Это были текстовые приключенческие игры, в которых общение между игроком и компьютером осуществлялось посредством ввода команд через клавиатуру.

С развитием компьютерных сетей, вылившихся в итоге в появление интернета на смену привычным всем однопользовательским клиентским играм приходит новое поколение игр, дающих возможность играть совместно с другими игроками. Самая первая онлайн игра *Multi-user Dungeon* была сделана в текстовом виде и запущена в сети *TelNet* в 1978 году. Разработчиками этой игры были Ричард Бартл и Рой Трабшоу.

С появлением интернета в дополнение к привычным всем однопользовательским играм приходят онлайн проекты: клиентские и браузерные. В конце 90-х растущие скорости доступа в интернет делают возможным становление браузера как самостоятельной игровой платформы. Он выступает в роли операционной оболочки для игр, позволяя играть без установки дополнительных программ.

Следующий виток развития игровой индустрии произошел благодаря колоссальному росту популярности мобильных телефонов, которые стали новой платформой общения людей и новой платформой распространения игр. Игры на Java, которые были на мобильных телефонах не составляли серьезной конкуренции ни браузерным, ни клиентским, ни социальным играм. Однако начиная с конца первого десятилетия 2000-х годов появление смартфонов, под которые можно было разрабатывать мощные проекты, открыло для индустрии рынок мобильных игр во всей красе.

История становления игровых платформ показывает, что с развитием технологий ускоряется процесс появления инновационных платформ для игр. Некоторые из них «взрывают» рынок и вытесняют старые, некоторые не обретают взрывного роста но занимают свою нишу. Вполне вероятно, что уже в ближайшие годы рынок мобильных игр будет вытеснен из тренда игровой индустрии более совершенной технологией взаимодействия человек-компьютер. Это может стать как виртуальная или дополненная реальность в текущей реализации, так и виртуальная реальность с полным погружением.

**1.2 История развития «*Action*-*Adventure*» как самостоятельного**

**игрового жанра**

Так уж вышло, что в компьютерной индустрии практически не осталось чистых жанров. Практически каждая современная игра включает в себя, как минимум, два-три жанра, нередко – и больше. Одним из таких гибридных жанров стал «*Action*-*Adventure*», он же – приключенческий боевик. И в нашей сегодняшней статье мы немного подробнее ознакомимся с его особенностями.

«*Action*-*Adventure*» один из самых интересных жанров видеоигр: он дает возможность почувствовать себя внутри насыщенного действиями и событиями боевика с харизматичными персонажами, динамичными сражениями и в меру сложными головоломками.

Терминами «*Action*-*Adventure*» или «Приключенческий боевик» называют гибридный жанр компьютерных игр, где сочетаются квест и боевик. В таких играх игроку предстоит не только ломать голову над головоломками и сюжетными задачами, но и проверить свою реакцию или выносливость.

Примечательно то, что до сих пор никто так и не смог договориться о том, когда «*Action*-*Adventure*» становится просто «*Action*» и лишается квестовой составляющей.

Говоря о компонентах жанра, их можно разделить на две категории:

– элементы квеста характеризуются наличием головоломок, сюжета, большого количества действующих лиц, инвентаря и других атрибутов;

– элементы боевика характеризуются перемещением персонажа, сражения, сюжет, изменение темпа игры и контекста, где действует персонаж.

Еще одним аспектом, который можно считать наследием квестов, является необходимость в более гибком устройстве точек зрения и сложных уровнях. Отсюда – непременное наличие сюжета и возможности управлять персонажем напрямую.

В свою очередь, Марк Уолв в своей книге первой игрой в жанре «*Action*-*Adventure*» называл игру «*Adventure*», вышедшая в том же году, где появились сражения, больше присущие action-играм.

На рисунке 1.1 демонстрируется игра первая игра жанра «*Action*-*Adventure*» под названием «*Adventure*».

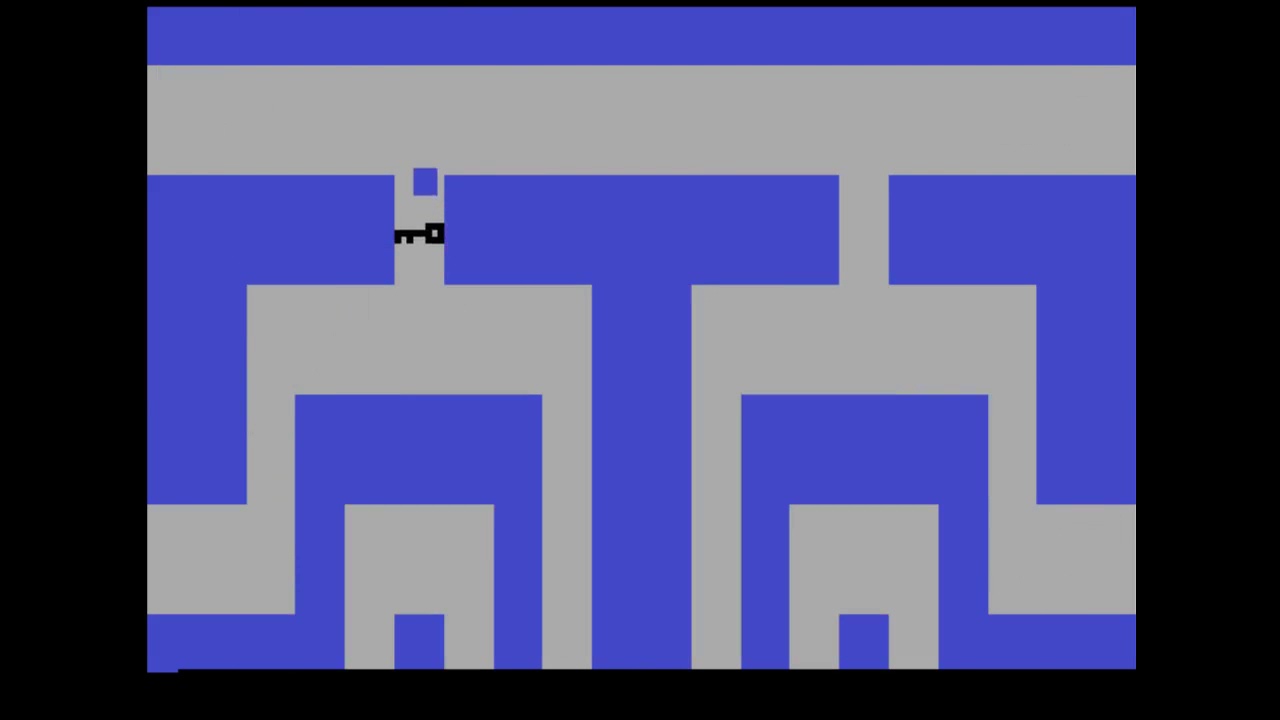


Рисунок 1.1 – Игра *Adventure*

Еще одной игрой, которая заложила основы жанра, стала «*Castle* *Wolfenstein*» 1981 года. В 1986 году вышло сразу несколько игр, которые стали считаться классикой жанра. Это – «*Castlevania*», «*Trojan*», «*The* *Legend* *of* *Zelda*».

**1.3 Игровые движки как основное средство разработки игровых приложений**

Вместе с созданием первых игр программисты пришли к тому, что каждая игра содержит общие компоненты, даже несмотря на различие аппаратных платформ. Общая для игр функциональность – графические решения, игровые механики, расчет физики и другое – стала выделяться в отдельные библиотеки.

Игровой движок в первую очередь создается в целях упрощения и ускорения разработки. Игровой движок представляет своеобразную узкоспециализированную операционную систему, поскольку включает все модули последней. В него входят: система управления памятью, графическая подсистема, система ввода, аудио подсистема, искусственный интеллект, физическая подсистема, сетевая подсистема, редактор игровых уровней и другое. Кроме того, ядро движка может предоставлять особый подход к работе с файлами – файловую систему, а так же отличающиеся от основной операционной системы средства работы с многопоточностью. Современный игровые движки вдобавок включают интерпретатор скриптового языка, заточенного для описания игровой логики, а нередко и полностью визуальный ее редактор. Его использование позволяет абстрагироваться от описания низкоуровневых команд и инструкций, а сконцентрироваться на геймплее. На этом составляющие движок компоненты не ограничиваются, их может быть как больше, так и меньше.

На данный момент лучшими в своем сегменте являются движки *Unity* и *Unreal* *Engine* 4 являются они таковыми в среде разработчиков из-за их удобства, детальной проработки и большого количества дополнительных библиотек, что позволяет настраивать и реализовывать практически любые идеи, приходящие в голову дизайнерам и игроделам.

*Unreal*, вышедший в 1998 году, стал первой игрой, принесший компании большой успех, но не только из-за того, что это был отличный шутер от первого лица. Первая часть серии имела абсолютно бескомпромиссные по меркам того времени системные требования для ее запуска, что вызывало негодование аудитории и сказалось на популярности проекта. Хотелось бы отметить, что большие требования к железу стали отличительной особенностью каждого следущего релиза новой версии *Unreal* *Engine*.

Технологии первой версии впечатляли своими возможностями, в него входили не только сильные графические свойства, но также физический движок и искусственный интеллект, который, кстати, был разработан одним из разработчиков первого *Quake* Стивеном Полжем.

Конечно, визуальный аспект первого Unreal был безусловным прорывом, но все эти возможности не имели бы никакого отражения в самом игровом процессе, если бы не огромный труд над дизайном. Нагнетающая атмосфера инопланетных просторов, замечательная электронная музыка, богатый арсенал оружия и, конечно же, культовый *Headshot* – заслуга гейм-дизайнера Клиффа Блжински.

Несмотря на то, что в первую версию движка был встроен свой редактор карт, он не нашел значительной популярности у энтузиастов. Являясь основным конкурентом *Quake* II, *Unreal* на тот момент имел не самый лучший сетевой режим в отличие от своего брата по жанру, что подстегнуло разработчиков выпустить отдельную версию игры, коей стал *Unreal* *Tournament*.

На выставке *E*3, прошедшей в 2014 году, были представлены первые технические возможности четвертой версии движка, которые показывали именно то, что хотели видеть – прогресса и эволюции в сфере компьютерной графики.

Была проделана масштабная работа над освещением, а физический рендеринг позволял обрабатывать настолько колоссальные пространства, что хотелось забыть о существовании всех предыдущих достижений *Epic* *Games* в этой сфере.

На данный момент над движком ведется постоянная работа, он обновлется практически каждую неделю, а его дополнениями занимается Тим Суини с небольшой командой программистов. Приняв во внимание опыт данного мастодонта, репутацию, заработанную потом и кровью, а также умение создавать действительно оптимизированные технологии, можно предположить, что в ближайшем будущем нас ждет еще одна революция.

На данный момент актуальным остается *Unreal* *Engine* 4, который был разработан в 2014 году. В нем у вас есть возможность создавать игры либо на языке *C*++, либо пользуясь визуальным инструментом программирования – так называемыми «блюпринтами», они же «чертежи». Теоретически они позволяют вообще обойтись без кода.

Крайней на данный момент версией движка является *Unreal* *Engine* 5, показ которой состоялся в мае 2020 года. В числе двух главных реализаций движка *Epic* заявила *Nanite* и *Lumen*. Из обеих выгоду извлекут и разработчики, и геймеры.

Описывая *Nanite*, *Epic* пускает в дело определения «виртуализированная геометрия» и «микрополигональный рендеринг». Что это значит? Разработчики теперь смогут загружать в движок модели высокого, кинематографического качества из миллионов полигонов. Впоследствии их не придется искусственно «портить», ухудшать детализацию, чтобы движок смог вывести всю эту красоту на экран с приемлемой кадровой частотой.

Напротив, благодаря *Nanite* не нужно соглашаться на компромиссы. Не нужно думать, среди прочего, о настройке *LOD* (*level of detail*) – уровней детализации моделей для разных ситуаций. Раньше, чтобы сэкономить ресурсы, разработчики делали, допустим, один вариант модели для близкого расстояния, второй – для среднего, третий – для дальнего. Но *Nanite* в *UE*5 рендерит изображение из расчета полигон на пиксель дисплея, то есть чем разрешение выше, тем картинка четче. Без разницы, лицом к лицу вы с персонажем или в сотне метров от него, количество полигонов в модели не уменьшится. Есть одно «но»: иногда число объектов в кадре зашкаливает, и тогда приходится использовать уловку с динамическим (изменяемым в зависимости от нагрузки) разрешением.

Далее поговорим о *Unity*. *Unity3d* является современным кросс-платфоменным движком для создания игр и приложений, разработанный *Unity* *Technologies*. С помощью данного движка можно разрабатывать не только приложения для компьютеров, но и для мобильных устройств (например, на базе *Android*), игровых приставок и других девайсов. Поговорим немного о характеристиках движка. Во-первых, стоит отметить то, что в среду разработки *Unity* интегрирован игровой движок, иными словами, вы можете протестировать свою игру не выходя из редактора. Во-вторых, *Unity* поддерживает импорт огромного количества различных форматов, что позволяет разработчику игры конструировать сами модели в более удобном приложении, а *Unity* использовать по прямому назначению — разработки продукта. В-третьих, написание сценариев (скриптов) осуществляется на наиболее популярных языках программирования – *C*# и *JavaScript*. Таким образом, *Unity3d* является актуальной платформой, с помощью которой вы можете создавать свои собственные приложения и экспортировать их на различные устройства, будь то мобильный телефон или приставка *Nintendo* *Wii*.

Плюсами движка можно отметить:

– В *Unity* очень просто войти. Проще, чем в другие движки. Он не ощу-щается сложным. По нему есть больше учебных материалов и информации. Также у *Unity* хорошая документация, поэтому с ним легче разобраться.

– *Unity* хорош тем, что он переносной – работает и на *iOS*, и на *Android*, и на огромном количестве платформ. Если есть платформа, то вполне вероятно, что *Unity* на ней работает.

– Огромный плюс движка это – *С*#. На сегодняшний момент язык программирования *C*# один из самых мощных, быстро развивающихся и востребованных языков в *IT*-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. C# уже не молодой язык и как и вся платформа .*NET* уже прошел большой путь. Первая версия языка вышла вместе с релизом *Microsoft Visual Studio* .*NET* в феврале 2002 года. *C*# является объектно-ориентированным и в этом плане много перенял у *Java* и *С*++. Например, *C*# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И *C*# продолжает активно развиваться, и с каждой новой версией появляется все больше интересного функционала. На *С*# намного проще дебажить и писать игровую логику.

*С*# не такой сложный, в нем есть встроенная *Garbage* *Colletor*, система рефлексии и другие приятности современных языков, такие технические вещи, сильно облегчают жизнь при разработке. *Unity* сам по себе визуально простой. После его открытия не придется искать нужное окно с настройками физики или чего-либо подобного, все инструменты находятся прямо под рукой.

**2 ОПРЕДЕЛЕНИЕ ОСНОВНЫХ ЭТАПОВ В РАЗРАБОТКЕ ИГРОВОГО ПРИЛОЖЕНИЯ**

## 2.1 Анализ рынка игровых приложений

Рынок видеоигр – один из самых молодых, перспективных и высокодоходных прямо сейчас. В 2020 году он достиг $175 млрд (рост на 20%), а общая сумма инвестиционных сделок составила $33,6 млрд.

Катализатором роста популярности видеоигр в 2020 году стала пандемия, во время которой люди остались без возможности проводить досуг вне дома. В результате самым доступным способом развлечься оказались компьютеры и консоли.

Эксперты предполагают, что объём отрасли преодолеет порог в 200 миллиардов долларов к 2023 году и достигнет 218,7 миллиарда долларов к 2024 году при среднегодовом темпе роста 5,2%. На рисунке 2.1 демонстрируется предполагаемый рост рынка игровой индустрии.



Рисунок 2.1 – Предполагаемый рост рынка игровой индустрии

Рынок видеоигр активно меняется и эволюционирует, этому способствует выход консолей нового поколения, приход в индустрию абсолютно новой аудитории, популярность *free*-*to*-*play* сегмента. Все больше появляется фондов и компаний, заинтересованных в инвестициях в игровые студии. Вид инвестиций и инвесторов также продолжает увеличиваться. Сейчас студии, как с уже запущенным проектом, так и новые, способны найти финансирование не только на локальном рынке, но и международном. Отчасти такому изменению поспособствовала пандемия, несмотря на влияние которой, количество сделок не только не сократилось, но и получило дополнительный рост. В будущем объём рынка наверняка продолжит увеличиваться, а инвестиции в игровые компании и проекты возрастут.

География рынка игровой индустрии в порядке убывания распределяется следующим образом: Азия, Северная Америка, Европа. Крупнейшими странами на этих рынках, соответственно, выступают Китай, США и Германия. На рисунке 2.2 демонстрируется география рынка игровой индустрии.

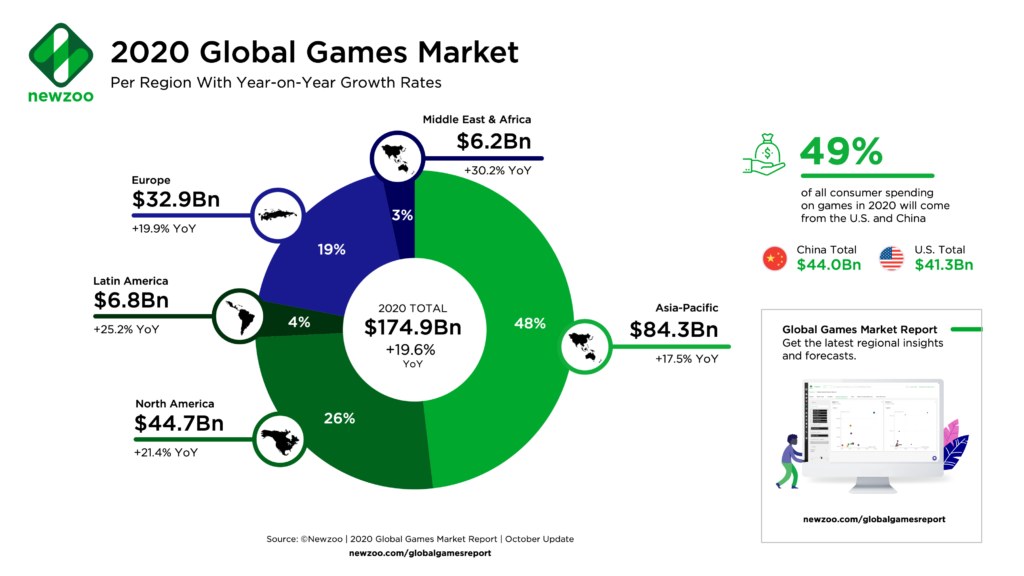


Рисунок 2.2 – География рынка игровой индустрии

В каждом регионе есть свои основные игроки. Рынок видеоигр больше всего сконцентрирован в сегменте мобильных игр. Топ-25 компаний занимают почти половину рынка, в сегменте консольных игр доля таких компаний – 40%, в сегменте игр для ПК – около 20%. Вдобавок на рынке видеоигр, особенно для мобильных платформ, пока еще много непубличных компаний.

За первый квартал 2021 года уже было закрыто 249 сделок на общую сумму превышающую $25 млрд. Из них нельзя не отметить такие крупные сделки как: *Electronic* *Arts* объявила о покупке компании *Glu* *Mobile* за $2,4 млрд, *ByteDance* приобрела игровую студию *Moonton* за $4 млрд, *Saber* *Mobile* купила три новые студии (*Gearbox* *Software*, *Aspyr*, *Easybrain*) за $2,6 млрд, *Enad* *Global* 7 приобрела российского паблишера онлайновых игр *Innova* за $132 млн. С уверенностью можно говорить, что тренд продолжится – в будущем можно ожидать крупные сделки от таких компаний, как *Tencent*, *Zynga*, *Activision* *Blizzard*, *Stillfront*.

При этом, несмотря на позитивные тренды, неизвестно, как справится игровая индустрия с существующими реалиями в виде активно растущих цен на акции игровых компаний и мультипликаторов по сделкам. Возможно, рынок скорректируется на ближайшие несколько лет или же возможны изменения со стороны регуляторов.

**2.2 Анализ целевой аудитории жанра «*Action*-*Adventure*»**

Разработчики видеоигр зарабатывают на играх несколькими способами. Основными моделями монетизации являются:

– премиум-модель;

– *free-to-play*;

– *pay-to-play*.

По премиум-модели игрок сразу приобретает видеоигру на внешнем носителе или ссылку для скачивания дистрибутива купленной игры.

Модель *free-to-play* не требует оплаты игры перед использованием. Игрок может бесплатно скачать игру и начать в нее играть. Но за определенные функции или предметы в игре придется заплатить.

По модели *pay-to-play*, как и по премиальной, игрок сразу покупает игру, но разработчик продолжает добавлять в нее новый контент, за доступ к которому пользователю нужно платить отдельно.

Модель *free-to-play* – основной способ монетизации мобильных игр. Абсолютное большинство проектов использует именно ее. Основная модель для заработка на консольных играх

Определить целевую аудиторию игрового проекта до того, как он выйдет на рынок, можно опираясь на похожие игры выбранного жанра.

В настоящее время игровая индустрия постоянно развивается и захватывает все большую и большую долю рынку. Из этого можно сделать вывод, что интерес людей к компьютерным играм продолжает расти, постоянно появляются новые пользователи. Увеличиваются продажи цифровой дистрибуции, открываются новые площадки для продажи компьютерных игр. На рисунке 2.3 демонстрируется статистика популярности игровых жанров.

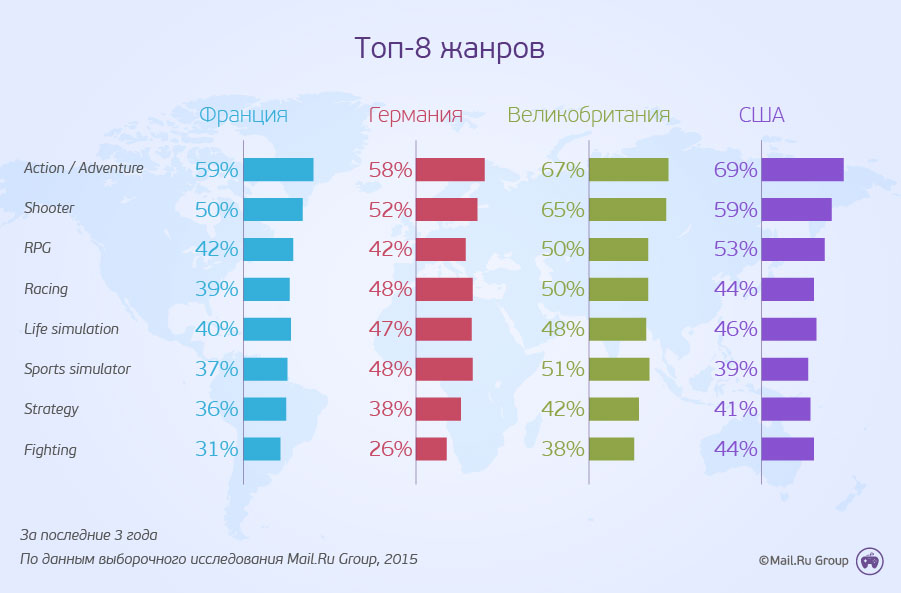


Рисунок 2.3 – Статистика популярности игровых жанров

На сегодняшний день игры в жанре «*Action*-*Adventure*» очень распространённые и любимы игроками. «*Action*-*Adventure*» самый интересный и разнообразный жанр. Жанр позволяет исследовать огромные открытые миры, общаться с различными персонажами, решать головоломки, а также вести захватывающие бои.

Происходит не только увеличение количества увлеченных пользователей, но и расширяется целевая аудитория, как мужская, так и женская аудиторию.

## 2.3 Цикл работы игрового приложения

Игровой цикл – это система или набор механик, которые будет использовать игрок. Будь то строительство, сражения, выращивание урожая, будь то мгновенный геймплей, который создает микрослой проекта.

Хороший игровой цикл создает «спринты» геймплея, которые приносят удовольствие при их повторении. Мало кто запускает игру в ожидании, что будет играть в нее сотни часов. Но если пользователь получает удовольствие, то общее проведенное время в игре начнет заметно увеличиваться.

Один из лучших примеров – соревновательные игры. Катки в *Mortal* *Kombat*, *Counter*-*Strike* и даже в *League of Legends* изначально не рассчитаны на 10 часов. Это короткие сессии, которые повторяются с каждым новым матчем.

Если вы понимаете ваш игровой цикл, и сможете разделить его так, чтобы каждый раз игроку было интересно – это сильно улучшит игру.

Опишем игровой цикл для будущей игры «*Action*-*Adventure*». На рисунке 2.4 демонстрируется разработанный игровой цикл.

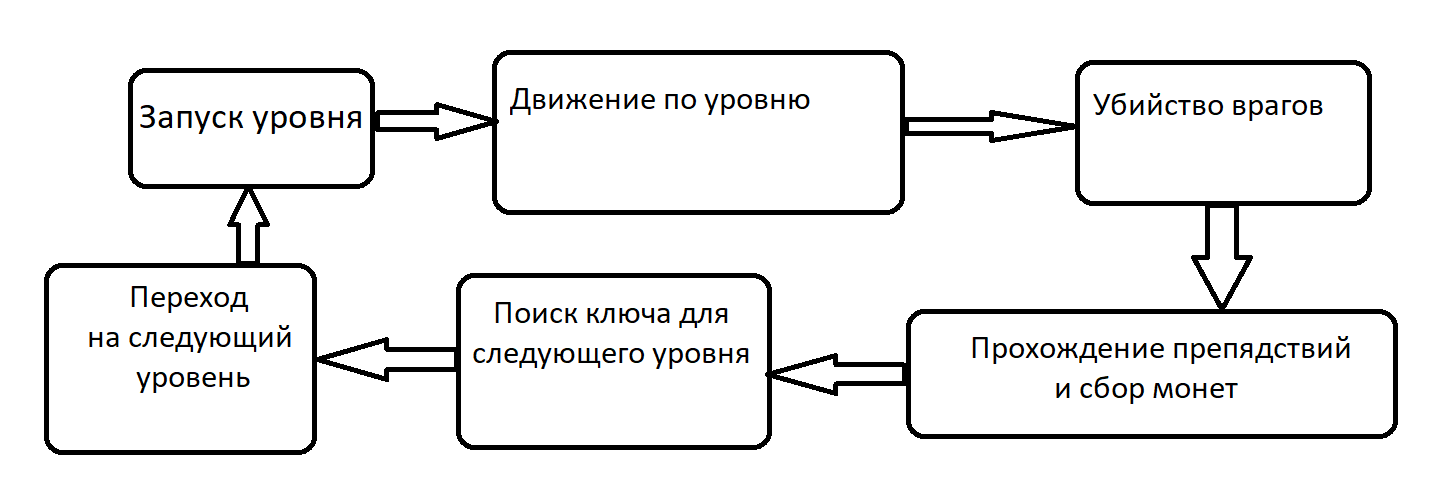


Рисунок 2.4 – Игровой цикл

В данном цикле представлены основные элементы игровой механики, которые пользователю предстоит совершать во время каждой игровой сессии. Кратко опишем каждый из элементов цикла:

* В начале игры игрок находится на первом уровне. С него и начинается игра.
* Каждый игровой уровень содержит свою уникальную стилистику и различные элементы окружения. Игрок может передвигаться в рамках дозволенной сцены.
* Во время прохождения уровней игрок встречает вражеских персонажей. Чтобы пройти дальше игроку необходимо искать путь обхода или уничтожить врага. Уничтожать врагов игрок может с помощью пистолета, но для этого игрок должен иметь патроны.
* Помимо врагов игроку необходимо проходить различные препятствия. На каждом уровне препятствия различаются. Так игрок может собирать монеты и покупать за них боеприпасы.
* Для того чтобы игрок мог перейти на следующий уровень необходимо найти ключ. Ключи находятся в труднодоступных местах.

После того как игрок нашёл ключ он может перейти на следующий уровень. Каждый игровой цикл начинается с нового уровня и заканчивается переходом на новый уровень.

**3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ**

**3.1 Разработка графической составляющей игры**

Этап создания игрового приложения начался с нахождения и создания спрайтов для игры. В них входят спрайты главного персонажа, второстепенных персонажей и врагов, так же различные элементы участвующие в окружении сцен. Игра содержит пять сцен, поэтому для каждой сцены стилистика элементов окружения отличается. При разработке графических материалов использовалось программное обеспечение *Adobe* *Photoshop* 2020 в которых производилось изменение игровых текстур. Так же множество спрайтов были взяты из интернета.

Начнём с главного героя. Спрайты были поделены на пять наборов спрайтов для того чтобы позже создать покадровую анимацию для каждого состояния героя. Главный персонаж имеет шесть наборов спрайтов:

– состояние покоя;

– бег;

– выстрел;

– прыжок;

– подкат;

– смерть.

На рисунках 3.1 демонстрируются наборы спрайтов для бега.

**

Рисунок3.1– Набор спрайтов для бега

На рисунках 3.2 демонстрируются наборы спрайтов для прыжка.

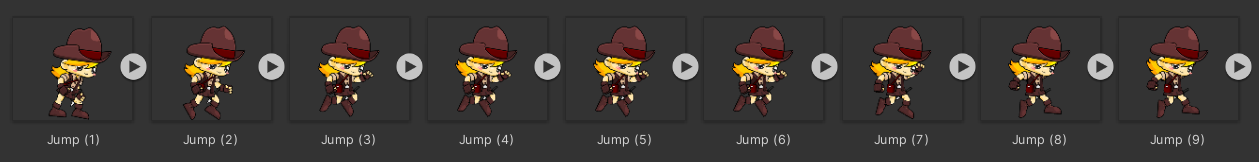
**

Рисунок3.2– Набор спрайтов для прыжка

На рисунках 3.3 демонстрируются наборы спрайтов для состояния покоя.

**

Рисунок 3.3 – Набор спрайтов для состояния покоя

Для реализации анимации смерти персонажа при столкновении с врагом и попадания в ловушки был подобран набор спрайтов состояния смерти. На рисунках 3.4 демонстрируются наборы спрайтов для смерти персонажа.

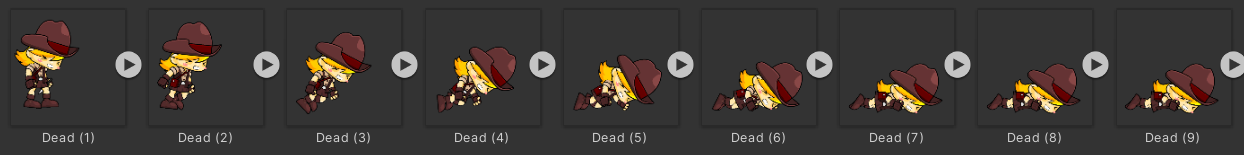
**

Рисунок 3.4 – Набор спрайтов для смерти персонажа

Далее были созданы спрайты для вражеских персонажей. В игре есть девять видов вражеских персонажей. Вид вражеского персонажа зависит от сцены на которой он находится. Вражеские персонажи находятся на определённых труднопроходимых местах сцены. На рисунке 3.5 демонстрируются спрайты для вражеских персонажей.

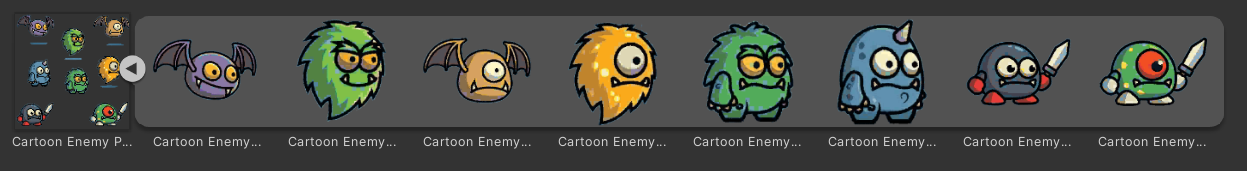
**

Рисунок 3.5 – Набор спрайтов вражеских персонажей

После окончания работы над спрайтами главного персонажа и вражеских персонажей была начата работа над поиском и созданием спрайтов различных элементов окружения сцен. Как было указано ранее игра имеет пять сцен различной стилистики. Это пустынная, лесная, зимняя, ночная и подземная сцена.

Стилистика элементов окружения отличается для каждой сцены.На рисунке 3.6 демонстрируются элементы окружения для лесной сцены.



Рисунок 3.6 – Элементы окружения для лесной сцены

Так же каждая сцена имеет свой задний фон, подходящий по стилистике сцены.На рисунке 3.7 представлен задний фон сцен.

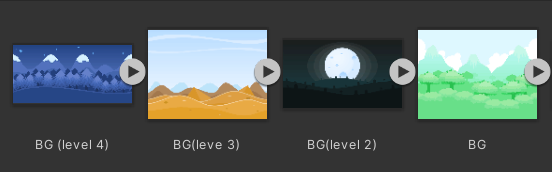


Рисунок 3.7 – Задний фон сцен

Далее были создан набор спрайтов для игровой валюты – монет, который игрок может собирать во время всей игры. После каждого уровня на собранные монеты игрок может купить дополнительные патроны. Без патронов сложность уровней возрастает в разы, так как игроку приходится побеждать ловкостью. На рисунке 3.8 демонстрируется набор спрайтов для монет.

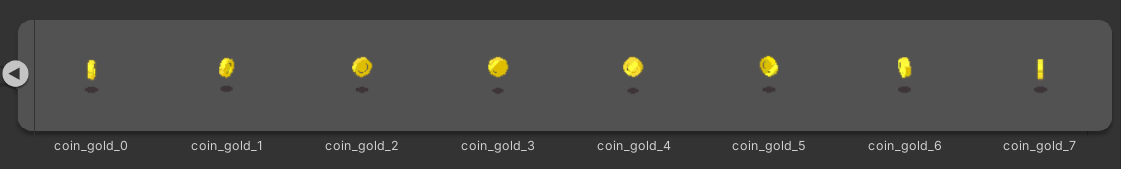


Рисунок 3.8 – Набор спрайтов для монет

Так же были найдены спрайты дополнительных важных элементов игры, такие как пуля и ключ. Для уничтожения вражеских персонажей главный персонаж имеет пистолет, который стреляет одним видом пуль. На начальном уровне количество патронов ограниченно. На рисунке 3.9 демонстрируется спрайт пули.

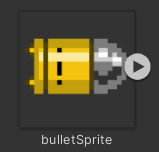
**

Рисунок 3.9 – Спрайт пули

На рисунке 3.10 демонстрируется спрайт ключа.

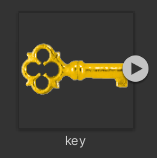
**

Рисунок 3.10 – Спрайт ключа

Ключ нужен для того чтобы проходить на следующие сцены, на каждой сцене в труднодоступном месте находится ключ, без которого игрок не сможет пройти на следующий уровень.

## 

**3.2 Разработка программного составляющего игры**

Для начала определим какой функционал должны будут обеспечивать игровые скрипты. Необходимый функционал игрового приложения:

– управлением главного персонажем, перемещение, прыжок, подкат и так далее;

– движение камеры за главным персонажем;

– уничтожение вражеских персонажей посредством стрельбы из оружия;

– получение урона от врагов;

– сбор монет главным персонажем;

– взаимодействие с различными окружающими элементами сцены;

– переход между сценами;

– реализация меню пользователя.

Для реализации перечисленного выше функционала был создан ряд основных классов:

– *Bullet* – класс отвечающий за логику пули;

– *CameraCollider* – класс отвечающий за коллайдер камеры;

– *CoinCollector* – класс отвечающий за сбор монет;

– *EnemyMovement* – класс отвечающий за движение вражеских персонажей;

– *EnemyAttack* – класс отвечающий за движение вражеских персонажей;

– *KeyCollector* – класс отвечающий за сбор ключей;

– *PlayerMovement* – класс отвечающий за движение главного персонажа;

– *Weapon* – класс отвечающий за логику стрельбы главного персонажа;

– Equipment – класс отвечающий за снаряжение главного персонажа;

– *CanGetDamage* – класс проверяющий получение урона;

– *StartScreenUIManager* – класс отвечающий за логику меню игры;

– *UIRestartButton*– класс отвечающий за логику кнопки обновления уровня;

– *UICoinsUpdater* – класс обновляющий элемент управления, отвечающий за количество монет;

– *UIBulletsUpdater* – класс обновляющий элемент управления, отвечающий за количество патронов.

На рисунке 3.11 представлена библиотека с перечисленными выше классами.

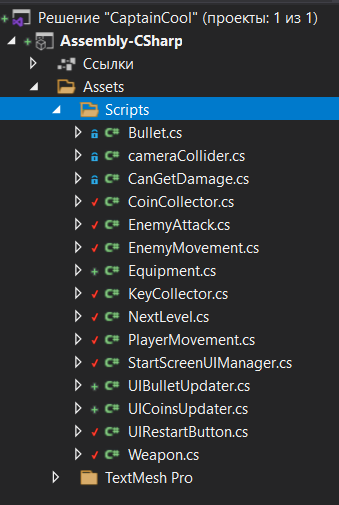


Рисунок 3.11 – Библиотека классов

Далее будет описана последовательность создания скриптов и их взаимодействие. Так же будут описаны основные поля и методы классов.

Для начала создадим класс *PlayerMovement* отвечающий за движение главного персонажа. Данный класс считывает клавиши нажатые игроком и управляет персонажем.

Поля класса *PlayerMovement*:

– *MoveSpeed*– скорость передвижения персонажа;

– *JumpForce* – сила прыжка персонажа;

– *SlideForce* – скорость подката персонажа;

– *Animator* – анимация персонажа;

– *MainCollider* – коллайдер персонажа;

– *SlideCollider* – коллайдер подката персонажа.

Методы класса *PlayerMovement*:

– *Slide* – реализация подката;

– *StopSlide* – реализация завершения подката;

– *Jump* – реализация прыжка;

– *Flip* – реализация поворота перонажа;

– *Dead* – реализация смерти перонажа.

После создания класса отвечающего за движение персонажа создадим классы для реализации стрельбы. Класс *Weapon* отвечает за стрельбу главного персонажа.

Поля класса *Weapon*:

– *Bullet*– игровой объект пуля;

– *FirePosition* – начальная позиция выстрела пули;

– *Animator* – анимация выстрела.

Методы класса *Weapon*:

– *Fire* – реализация выстрела.

Класс *Bullet* реализует функционал пули.

Поля класса *Bullet*:

– *Speed*– скорость пули;

– *BulletSelfDestroyTime* – время уничтожения пули;

– *Rb* – физика пули.

Методы класса *Bullet*:

– *OnTriggerEnter2D* – метод считывающий столкновения пули.

Далее были созданы классы отвечающие за поведение вражеских персонажей. Это классы *EnemyAttack* и *EnemyMovement*. Класс *EnemyAttack* реализует атаку и смерть самих вражеских героев, метод *OnCollisionEnter2D* считывает столкновения с главным персонажем. Класс *EnemyMovement* реализует движение вражеских персонажей, в методе *FixedUpdate* реализуется движение вражеского персонажа.

Поля класса *EnemyMovement*:

– *Speed*– скорость вражеского персонажа;

– *Raycast* – свойство *Transform* игрового объекта вражеского персонажа.

Далее создадим класс *CoinCollector* отвечающий за сбор монет и класс *KeyCollector* отвечающий за сбор ключей.

Методы класса *CoinCollector*:

– *OnTriggerEnter2D* – реализация сбора монет;

– *GetNoOfCoins*– получение количества собранных монет.

Методы класса *KeyCollector*:

– *OnTriggerEnter2D* – реализация сбора ключей;

– *HaveRightKey* – получение информации о собранных ключах.

Последний шаг – это написание скриптов для работы с элементами управления интерфейсом. Данные скрипты необходимы для реализации меню пользователя, возобновление игры и вывода информации о количестве монет и патронов. Класс *StartScreenUIManager* отвечает за логику пользовательского меню. Методы класса *StartScreenUIManager*:

– *StartButtonClicked* – логика кнопки «*Start*», начать игру;

– *SettingsButtonClicked* – логика кнопки «*Settings*», настройки игры;

– *ExitButtonClicked* – логика кнопки «*Exit*», выйти из игры.

Класс *UIRestartButton* отвечает за логику кнопки обновления уровня и имеет метод *RestartLevel*, который перезапускает уровень.

Класс *UICoinsUpdater* обновляет элемент управления, отвечающий за количество монет. Класс имеет поле *СoinsCountUI* которое ссылается на элемент управления для вывода количества монет. В методе *Update* обновляется значение монет.

Класс *UIBulletsUpdater* обновляет элемент управления, отвечающий за количество патронов. Класс имеет поле *BulletsCountUI* которое ссылается на элемент управления для вывода количества патронов. В методе *Update* обновляется значение патронов.

Нужно отметить что библиотека классов не имеет скриптов реализующих управление камерой, так как вся логика камеры была реализована с помощью возможностей библиотеки *Cinemachine.*

**3.3 Эксплуатация игры**

После запуска игры игрок находится в начальном меню. В нём игрок может настроить игру и начать играть. В начале игры игрок находится на первом уровне. С него и начинается игра.

Цель игры состоит в том, чтобы пересечь все уровни и собрать все ключи, только после этого игрок будет свободен и сможет начать свой собственный путь.

Нужно отметить то, что не собрав ключи игрок не может перемещаться между уровнями. Ключи находятся в труднодоступных местах. На рисунке 3.12 демонстрируется ключ одного из уровней.

**

Рисунок 3.12 – Ключ одного из уровней

Во время прохождения уровней игрок встречает вражеских персонажей. Чтобы пройти дальше игроку необходимо искать путь обхода или уничтожить врага. Уничтожать врагов игрок может с помощью пистолета, но для этого игрок должен иметь патроны. Нужно отметить что в начале первого уровня игрок имеет 20 патронов. Если же патроны закончились игрок может приобрести их в магазине за собранные монеты. Стоимость одного патрона равняется десяти монетам. На рисунке 3.13 демонстрируется ситуация встречи с вражескими персонажами.

**

Рисунок 3.13 – Встреча с вражескими персонажами

Помимо врагов игроку необходимо проходить различные препятствия. На каждом уровне они отличаются. В пустынном уровне это может быть яма с трупами, а в подземном уровне это может быть железная пила. На рисунке 3.14 представлен один из видов препятствия.

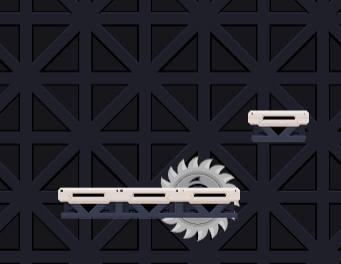


Рисунок 3.14 – Препятствие в виде железной пилы

Далее будет приведена раскладка клавиатуры и мыши для управления интерфейсом игры и главным персонажем. Раскладка клавиатуры демонстрируется в таблице 3.1.

Таблица 3.1 – Раскладка клавиатуры и мыши

|  |  |
| --- | --- |
| Действие | Клавиша |
| Движение влево | A |
| Движение влево | Left |
| Движение вправо | D |
| Движение вправо | Right |
| Прыжок | Space |
| Выстрел | Ctrl |
| Выстрел | Левая кнопка мыши |
| Меню | Escape |

При желании игрок может поменять раскладку клавиатуры в начале игры в начальном меню или же в процессе игры в игровом меню.

3.4 Интерфейс игры

После запуска игры запускается начальное меню. Начальное меню содержит такие кнопки как:

– *Start* – начать игру;

– *Settings* – настройки игры;

– *Start* – выйти из игры.

На рисунке 3.15 демонстрируется начальное меню.



Рисунок 3.15 – Начальное меню игры

Так же во время игры может войти в игровое меню. Игровое меню содержит функционал аналогичный начальному меню и возможность зайти в магазин. Игровое меню содержит такие кнопки как:

– Re*start* – перезапустить уровень;

– *Shop* – игровой магазин;

– *Settings* – настройки игры;

– *Start* – выйти из игры.

На рисунке 3.16 демонстрируется игровое меню.

**

Рисунок 3.16 – Игровое меню

Так же было создано подменю с магазином, для того чтобы игрок мог пополнять боезапас во время игры. Чтобы зайти в магазин необходимо перейти с игрового меню в раздел *Shop*. Магазин содержит такие кнопки как:

– *Buy* – приобрести патроны;

– *Back* – вернуться в игровое меню.

На рисунке 3.17 демонстрируется раздел *Shop*.



Рисунок 3.17 – Игровое меню

После нажатия кнопки «*Start*» в начальном меню, игрока начинает игру с первого уровня. Как видно выше во время игры игроку демонстрируется количество собранный монет и количество патронов. На рисунке 3.18 демонстрируется один из уровней игры.

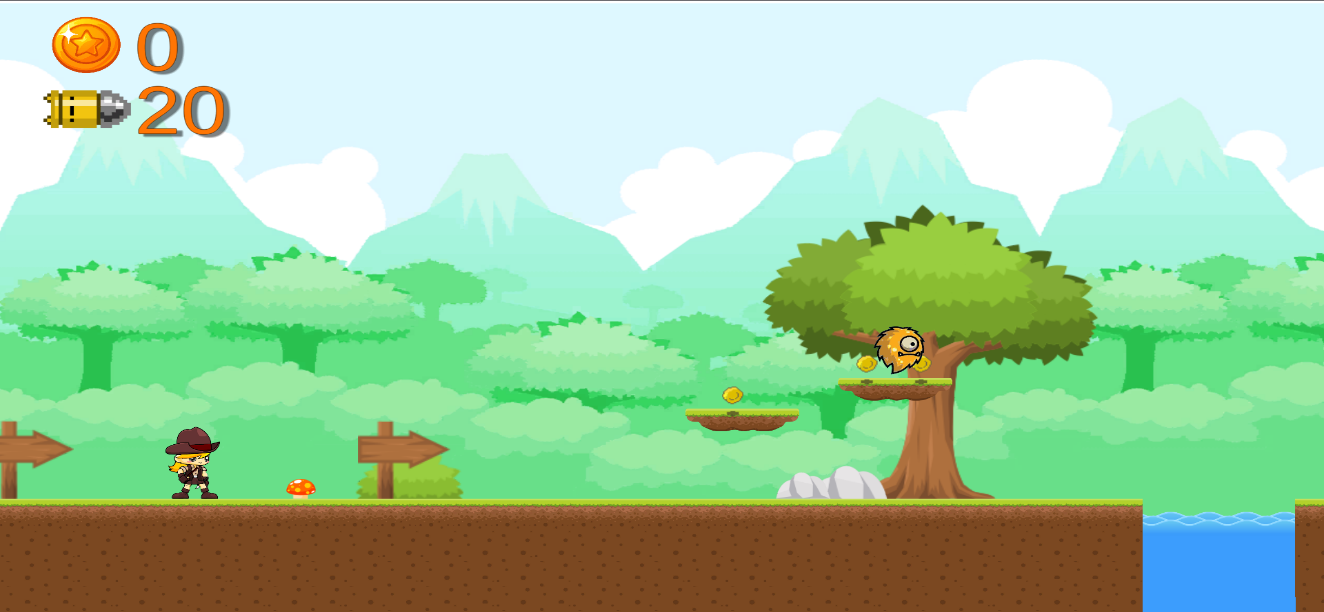


Рисунок 3.18 – Первый уровень игры

Если проходя уровень игрока убивают вражеские персонажи или игрок попадает в ловушки, то игра проигрывает. Игрок может выйти с игры или перезапустить уровень на котором он умер. На рисунке 3.19 демонстрируется реализация проигрыша игрока.



Рисунок 3.19 – Проигрыш игрока

Необходимо отметить что игровые меню созданы с помощью компонент *Canvas*. *Canvas* представляет собой абстрактное пространство, в котором производится настройка и отрисовка *UI*. Для реализации логики игровых меню был создан ряд классов, которые описаны в главе «Разработка программного составляющего игры».

**ЗАКЛЮЧЕНИЕ**

За время работы над проектом было изучено и реализовано на практике множество методов и подходов создания игр с помощью *Unity*.

Для того чтобы изучит все нужные технология была подобрана литература в области игровой литературы и разработки игр.

При реализации игры были использованы средства кросс-платформенной среды Unity и возможности языка программирования C#.

В ходе разработки игрового приложения решены следующие задачи:

– произведён обзор литературы и средств по программно-техническому обеспечению для разработки игровых приложения;

– разработаны графические материалы и программное обеспечение реализующие весь функционал игры;

– предоставлена вся информация о подходах и методах создания игр жанра «*Action*-*Adventure*»;

– предоставлены данные для эксплуатации игры.

В результате работы была разработана компьютерная 2*D* игра жанра «*Action*-*Adventure*» под названием «*CapitanCool*». Цель игры состоит в том, чтобы пересечь все уровни и собрать все ключи, только после этого игрок будет свободен и сможет начать свой собственный путь.

Данная игра подойдет игрокам любой возрастной категории. Игра имеет понятное управление и простую, но достаточно приятную спрайтовую графику.

Больше всего игра понравится людям, любящим поиграть как в игры с элементами боевика так и с квестовой составляющей.

Из интересных особенностей игры можно выделить динамический и интересный сюжет игры насыщенные сцены с приятной графикой.

Дизайн-документ игрового приложения содержит 92,66% уникальной информации.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. *Action*-*adventure* – что же скрывается за этими словами? [Электронный ресурс]. – Режим доступа: https://zen.yandex.ru/media/gamescore/actionadventure--chto-je-skryvaetsia-za-etimi-slovami-604fafcbd3283e6cff6f094f. – Дата доступа: 14.11.2021.
2. Руководство *Unity* [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/ru/530/Manual/UnityManual.html. – Дата доступа: 21.11.2021.
3. *CLR* *via* *C*#. Программирование на платформе *Microsoft* .*NET* *Framework* 4.5 на языке *C*#. – СПб: Питер, 2020. – 896c.
4. Искусство создания сценариев в *Unity* / Алан Торн. – СПб: Питер, 2016. – 360с.
5. *Unity* и *С*#. Геймдев от идеи до реализации / Бонд Джереми Гибсон. –СПб: Питер, 2019. – 928 с.
6. *Unity* в действии. Мультиплатформенная разработка на *C*# / Джозеф Хокинг. – СПб: Питер, 2018. – 608 c.
7. Знакомство с *Timeline* в *Unity* [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/418035/. – Дата доступа: 29.11.2021.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы «*CapitanCool*»**

Листинг класса Bullet.cs:

using UnityEngine;

public class Bullet : MonoBehaviour

{

public float speed;

public float bulletSelfDestroyTime = 3f;

public Rigidbody2D rb;

private void Start()

{

rb.velocity = transform.right \* speed;

Destroy(gameObject, bulletSelfDestroyTime);

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision != cameraCollider.getCollider())

{

Destroy(gameObject);

//Debug.Log(collision.gameObject.name);

if (collision.gameObject.layer == 9)

{

Destroy(collision.gameObject);

}

}

}

}

Листинг класса CameraCollider.cs:

using UnityEngine;

public class CameraCollider : MonoBehaviour

{

static Collider2D camCollider;

private void Awake()

{

camCollider = GetComponent<Collider2D>();

}

public static Collider2D getCollider()

{

return camCollider;

}

}

Листинг класса CanGetDamage.cs:

using UnityEngine;

public class CanGetDamage : MonoBehaviour

{

Animator animator;

private void Start()

{

animator = GetComponent<Animator>();

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.layer == 11)

{

animator.SetBool("IsDead", true);

}

}

}

Листинг класса CoinCollector.cs:

using UnityEngine;

public class CoinCollector : MonoBehaviour

{

private static int coinsCount;

private void Start()

{

coinsCount = 0;

}

private void OnTriggerEnter2D(Collider2D collision)

{

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

{

Destroy(gameObject);

coinsCount++;

}

}

public static int GetCoinsCount()

{

return coinsCount;

}

}

Листинг класса EnemyAttack.cs:

using UnityEngine;

public class EnemyAttack : MonoBehaviour

{

private Animator animator;

private void OnCollisionEnter2D(Collision2D collision)

{

gameObject.GetComponent<Collider2D>().enabled = false;

animator = collision.gameObject.GetComponent<Animator>();

animator.SetBool("IsDead", true);

}

}

Листинг класса EnemyMovement.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyMovement : MonoBehaviour

{

public float speed;

public Transform raycast;

[SerializeField] float raycastDistanceDown = 2f;

[SerializeField] float raycastDistanceFront = 1f;

private bool isFacingRight = true;

private LayerMask ignoreRay;

private Vector2 frontRayDirection = Vector2.right;

private void Start()

{

ignoreRay = LayerMask.GetMask("Enemy", "Collectable", "Ignore Raycast");

}

private void FixedUpdate()

{

transform.Translate(Vector2.right \* speed \* Time.deltaTime);

// Down ray (enemy not drop from top).

RaycastHit2D raycastHit2DDown = Physics2D.Raycast(raycast.position, Vector2.down, raycastDistanceDown, ~ignoreRay);

// Front ray (for checking frong collider)

RaycastHit2D raycastHit2DFront = Physics2D.Raycast(raycast.position, frontRayDirection, raycastDistanceFront, ~ignoreRay);

// Checking for collider (player and coins set to ignoreRaycast).

if (raycastHit2DDown.collider == false || raycastHit2DFront.collider == true)

{

if (isFacingRight == true)

{

frontRayDirection = Vector2.left;

transform.eulerAngles = new Vector2(0, -180);

isFacingRight = false;

}

else

{

frontRayDirection = Vector2.right;

transform.eulerAngles = new Vector2(0, 0);

isFacingRight = true;

}

}

}

}

Листинг класса Equipment.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using UnityEngine;

public class Equipment

{

private int bulletsCount;

public Equipment(int bulletsCount)

{

this.bulletsCount = bulletsCount;

}

public int BulletsCount

{

get { return bulletsCount; }

set { bulletsCount = value; }

}

}

Листинг класса KeyCollector.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class KeyCollector : MonoBehaviour

{

private static bool haveRightKey;

private void Start()

{

haveRightKey = false;

}

public static bool HaveRightKey ()

{

return haveRightKey;

}

private void OnTriggerEnter2D(Collider2D collision)

{

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

{

Destroy(gameObject);

haveRightKey = true;

}

}

}

Листинг класса NextLevel.cs:

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using UnityEngine;

using UnityEngine.SceneManagement;

public class NextLevel : KeyCollector

{

private void OnCollisionEnter2D(Collision2D collision)

{

if (HaveRightKey() && collision.gameObject.tag == "Player")

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

}

}

Листинг класса PlayerMovement.cs:

using System.Collections;

using TMPro;

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

public float moveSpeed = 5f;

public float jumpForce = 5f;

public float slideForce = 20f;

public Animator animator;

public Collider2D mainColider, slideColider;

public TextMeshProUGUI gameOverText;

public GameObject restartLevel;

bool isGrounded;

bool P\_facingRight = true;

bool jumpKeyDown;

bool isSliding = false;

bool slideKeyDown;

float movement;

Rigidbody2D rb;

private void Start()

{

rb = GetComponent<Rigidbody2D>(); //getting the rigidBody.

slideColider.enabled = false;

}

private void FixedUpdate()

{

//all physics related work done here.

moveHorizontal();

jump();

slide();

}

private void Update()

{

//all input related work done here.

movement = Input.GetAxisRaw("Horizontal"); //geting the input on horizontal axis.

if (!jumpKeyDown && isGrounded)

{

jumpKeyDown = Input.GetButtonDown("Jump"); //checking for jump key down.

}

if (!slideKeyDown && isGrounded && Mathf.Abs(rb.velocity.x) > 0.05)

{

slideKeyDown = Input.GetKeyDown(KeyCode.S); //checking for slide key down.

}

if (movement > 0 && !P\_facingRight) //checking for player faceing direction.

{

Flip();

}

else if (movement < 0 && P\_facingRight)

{

Flip();

}

if (!isGrounded && !isSliding) //checking if the player is drooping

{

animator.SetBool("IsJumping", true);

}

}

private void slide()

{

if (slideKeyDown)

{

isSliding = true;

animator.SetBool("IsSliding", true);

if (P\_facingRight)

{

rb.velocity = new Vector2(slideForce, 0f);

}

else if (!P\_facingRight)

{

rb.velocity = new Vector2(-slideForce, 0f);

}

slideColider.enabled = true;

mainColider.enabled = false;

StartCoroutine("stopSliding");

}

}

private IEnumerator stopSliding()

{

yield return new WaitForSeconds(0.5f);

animator.SetBool("IsSliding", false);

slideKeyDown = false;

mainColider.enabled = true;

slideColider.enabled = false;

isSliding = false;

}

private void jump()

{

if (jumpKeyDown && isGrounded)

{

// Adding force to rigidBody.

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

// Enabling the jump animation.

animator.SetBool("IsJumping", true);

isGrounded = false;

jumpKeyDown = false;

}

}

private void moveHorizontal()

{

// Adding the velocity to the rigidBody

rb.velocity = new Vector2(movement \* moveSpeed, rb.velocity.y);

// Enabling or disabling run animation.

animator.SetFloat("Speed", Mathf.Abs(movement));

}

/// <summary>

/// Checking the player it is grounded or not.

/// </summary>

/// <param name="collision"> Collider2D. </param>

private void OnTriggerStay2D(Collider2D collision)

{

if (collision != null && collision != cameraCollider.getCollider())

{

isGrounded = true;

animator.SetBool("IsJumping", false); //if player is grounded disabling the jump animation.

}

}

private void OnTriggerExit2D(Collider2D collision) //checking for player to exit the ground.

{

isGrounded = false;

}

/// <summary>

/// Fliping the player direction.

/// </summary>

private void Flip()

{

// Switch the way the player is labelled as facing.

P\_facingRight = !P\_facingRight;

transform.Rotate(0f, 180f, 0f);

}

/// <summary>

/// Calling through animation event(at end frame)

/// </summary>

private void dead()

{

Destroy(gameObject);

enablingUI();

}

private void preDeath() //calling through animation event(at end frame)

{

Destroy(gameObject, 1f);

enablingUI();

}

private void enablingUI()

{

gameOverText.enabled = true; //enabling the game over text

restartLevel.SetActive(true); //enabling restart button

}

}

Листинг класса StartScreenUIManager.cs:

using UnityEngine;

using UnityEngine.SceneManagement;

public class StartScreenUIManager : MonoBehaviour

{

public void StartButtonClicked()

{

SceneManager.LoadScene(4);

}

public void SettingsButtonClicked()

{

// TODO: настройки

}

public void ExitButtonClicked()

{

Application.Quit();

Debug.Log("Quit!");

}

}

Листинг класса UIBulletUpdater.cs:

using TMPro;

using UnityEngine;

public class UIBulletUpdater : MonoBehaviour

{

public TextMeshProUGUI BulletsCountUI;

private void Update()

{

BulletsCountUI.text = Weapon.GetBulletsCount().ToString();

}

}

Листинг класса UICoinsUpdater.cs:

using TMPro;

using UnityEngine;

public class UICoinsUpdater : MonoBehaviour

{

public TextMeshProUGUI СoinsCountUI;

private void Update()

{

СoinsCountUI.text = CoinCollector.GetCoinsCount().ToString();

}

}

Листинг класса UIRestartButton.cs:

using UnityEngine;

using UnityEngine.SceneManagement;

public class UIRestartButton : MonoBehaviour

{

public void RestartLevel()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

}

Листинг класса Weapon.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Weapon : MonoBehaviour

{

public GameObject bullet;

public Transform firePosition;

public Animator animator;

private static int bulletsCount;

private void Start()

{

bulletsCount = 20;

}

private void Update()

{

if (Input.GetButtonDown("Fire1") && bulletsCount > 0)

{

animator.Play("Player\_Fire");

Fire();

}

}

private void Fire()

{

Instantiate(bullet, firePosition.position, firePosition.rotation);

bulletsCount -= 1;

}

public static int GetBulletsCount()

{

return bulletsCount;

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Руководство пользователя**

1. Введение.

Руководство пользователя обеспечивает получение пользователем базовых навыков по эксплуатации игрового приложения.

Разработана компьютерная игра жанра «*Action*-*Adventure*» под названием «*CapitanCool*». Цель игры состоит в том, чтобы пересечь все уровни и собрать все ключи, только после этого игрок будет свободен и сможет начать свой собственный путь. Управление персонажем осуществляется с помощью клавиатуры и мышью.

Программное средство обладает следующим функционалом:

* наличие счёта;
* игровые сцены;
* игровое меню
* управление персонажем при помощи клавиатуры;
* наличие игровой валюты.

Ход использования приложения.

Для использования программного средство пользователь должен быть ознакомлен с:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

1. Назначение и условия применения.

Для корректной работы программного средства необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows* *XP* и выше;
* наличие стандартной клавиатуры;
* наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

1. Подготовка к работе.

Для установки приложения необходимо загрузить на компьютер каталог *CapitanCool*. После этого в папке с программой и запустить файл *Capitan* *cool*.*exe*.

Если все инструкции соблюдены и приложение не выдает никаких сообщений об ошибках, значит, программа работает исправно.

1. Описание операций.

В ходе разработки игрового приложения были реализовано окно игрового поля, на котором происходит основное действие игры.

Игровой процесс начинается сразу после запуска игры и нажатия кнопки «*Start*».

Игровое пространство реализовано в виде сцены.

Раскладка клавиатуры и мыши, отвечающая за управление персонажем и интерфейсом игры демонстрируется в таблице 1.

Таблица 1 – Раскладка клавиатуры и мыши

|  |  |
| --- | --- |
| Действие | Клавиша |
| Движение влево | A |
| Движение влево | Left |
| Движение вправо | D |
| Движение вправо | Right |
| Прыжок | Space |
| Выстрел | Ctrl, левая кнопка мыши |
| Меню | Escape |

Аварийные ситуации.

Чтобы избежать ошибок при использовании данного игрового приложения, необходимо соблюдать порядок действий и условия использования, описанные в главе 3 данного руководства пользователя.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова. Или можно подождать некоторое время, так как существует возможность зависания программы за счет наличия библиотек, используемых в игровом приложении.

1. Рекомендации по освоению.

Для правильного функционирования игровое приложение рекомендуется запускать при наличии одного игрока и клавиатуры.

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Руководство программиста**

1. Назначение и условия применения.

Разработана компьютерная игра жанра «*Action*-*Adventure*» под названием «*CapitanCool*». Цель игры состоит в том, чтобы пересечь все уровни и собрать все ключи, только после этого игрок будет свободен и сможет начать свой собственный путь.

Для корректной работы программного средства необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие стандартной клавиатуры;
* наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

1. Характеристики программы.

Игровой процесс реализуется в одном окне.

1. Обращение к программе.

Приложение запускается путем открытия файла *Capitan* *cool*.*exe*.

1. Выходные данные.

Выходной информацией является экран с результатом игрового процесса.

1. Сообщения.

По завершению игры на экране появляется сообщение о завершении игры.

**ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Руководство системного программиста**

1. Общие сведения о программе.

Разработана компьютерная игра жанра «*Action*-*Adventure*» под названием «*CapitanCool*». Цель игры состоит в том, чтобы пересечь все уровни и собрать все ключи, только после этого игрок будет свободен и сможет начать свой собственный путь. Для корректной работы программного средства необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие стандартной клавиатуры;
* наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

1. Структура программы.

Игровой процесс происходит в одном окне, где отображаются все игровые объекты.

1. Настройка программы

Настройка программы осуществляется в игровом меню путём перехода во вкладку «*Settings*».

1. Проверка программы.

Отсутствие сообщения об ошибках свидетельствует, о корректной работе приложения.

1. Дополнительные возможности.

Дополнительных возможностей программа не имеет.

1. Сообщения системному программисту.

Игровой процесс происходит в одном окне сразу после запуска приложения.

**ПРИЛОЖЕНИЕ Д**

(обязательное)

**Схема навигации по меню игрового интерфейса и оболочки игры**