Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Хакасский технический институт – филиал ФГАОУ ВО
«Сибирский федеральный университет»

Кафедра прикладной информатики, математики и естественно-научных дисциплин

УТВЕРЖДАЮ
Заведующий кафедрой
___ Е. Н. Скуратенко
подпись
«____» _____ 2020 г.

БАКАЛАВРСКАЯ РАБОТА
09.03.03 Прикладная информатика

Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Руководитель _____ ст. преподаватель В. И. Кокова
подпись, дата

Выпускник _____ К. В. Бобонаков
подпись, дата

Консультанты
по разделам:

Экономический _____ Е. Н. Скуратенко
подпись, дата

Нормоконтролер _____ В. И. Кокова
подпись, дата

Абакан 2020

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Хакасский технический институт – филиал ФГАОУ ВО
«Сибирский федеральный университет»

Кафедра прикладной информатики, математики и естественно-научных дисциплин

УТВЕРЖДАЮ
Заведующий кафедрой
___ Е. Н. Скуратенко
подпись
«____» _____ 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту Бобонакову Константину Владимировичу

Группа ХБ 16-03

Направление 09.03.03 Прикладная информатика

Тема выпускной квалификационной работы: Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Утверждена приказом по университету № 216 от 06.04.2020 г.

Руководитель ВКР: В. И. Кокова, ст. преподаватель, ХТИ – филиал СФУ

Исходные данные для ВКР: заказ ХТИ – филиала СФУ.

Перечень разделов ВКР:

1. Анализ предметной области. Выбор средств проектных решений.
2. Описание разработки игры.
3. Расчёт затрат и оценка экономической эффективности реализации проекта «Разработка игры в жанре 2D-платформер для направления «Прикладная информатика».

Перечень графического материала: нет

Руководитель ВКР _____ В. И. Кокова
подпись

Задание принял к исполнению _____ К. В. Бобонаков
подпись

«06» апреля 2020 г.

РЕФЕРАТ

Выпускная квалификационная работа (ВКР) по теме «Разработка игры в жанре 2D-платформер для направления «Прикладная информатика» содержит 55 страниц текстового документа, 7 таблиц, 32 рисунка, 7 формул, 22 использованных источников, 1 приложение.

ПРОЕКТ, ИНФОРМАЦИОННАЯ СИСТЕМА, БАЗЫ ДАННЫХ, UNITY, 2D-ПЛАТФОРМЕР, ИГРА, КВЕСТ, ЭКОНОМИКА, ЭФФЕКТИВНОСТЬ, РИСК, ЭКСПЛУАТАЦИЯ.

Объект выпускной квалификационной работы: процесс разработки компьютерных игр.

Предмет выпускной квалификационной работы: разработка компьютерных игр в жанре 2D-платформер.

Актуальность работы заключается в том, что разработка компьютерных игр в настоящее время является перспективной сферой занятости, так как существует потребность в качественных разработчиках, поэтому это позволит привлечь интерес к направлению «Прикладная информатика», а также повысить популярность ХТИ – филиала СФУ.

Цель ВКР: разработка игрового приложения для привлечения абитуриентов при поступлении в ХТИ – филиал СФУ на направление «Прикладная информатика».

Основными задачами ВКР являются:

– проанализировать существующие практики для привлечения абитуриентов;

– выбрать и обосновать жанр, в котором будет разрабатываться игра;

– проанализировать и обосновать средства разработки игры;

– разработать игру;

– провести оценку экономической эффективности игры.

В результате была разработана игра, которая будет выполнять функцию привлечения абитуриентов на направление «Прикладная информатика».

ABSTRACT

The theme of the graduation thesis is «Development of a 2D-platformer for the Training Program of «Applied Informatics». It contains 55 pages, 7 charts, 32 drawings, 7 formulae, 22 reference items, 1 application.

PROJECT, IT SYSTEM, DATABASES, UNITY, 2D-PLATFORMER, GAME, QUEST, ECONOMY, EFFICIENCY, RISK, OPERATION.

The object of the graduation thesis: the process of creating computer games.

The subject of the graduation thesis: the development of 2D-platformer computer games.

The relevance of the research: development of computer games is currently a promising area of employment, since there is a need for high-quality developers, so this will attract interest in the training program of «Applied Informatics», as well as increase the popularity of KhTI – branch of SibFU.

The purpose of the graduation thesis is the development of a gaming application to get the enrollees interested in the training program of «Applied Informatics» of KhTI – branch of SibFU.

The main objectives of the graduation thesis are:

1. To analyze existing practices to attract enrollees.
2. To give reasons for choosing the game's genre to be developed.
3. To analyze and to give ground for game development tools.
4. To develop a computer game.
5. To evaluate the economic efficiency of a computer game's promotion.

The game has been developed; it will contribute to attracting enrollees to the «Applied Informatics» training program.

English language supervisor _____ N.V. Chezybaeva

Signature, date

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Индустрия видеоигр зародилась в середине 70-х годов прошлого века как движение энтузиастов и за сравнительно небольшое время выросла из небольшого рынка в отрасль с годовой прибылью в более, чем 140 миллиардов долларов [1].

Компьютерные игры – один из главных двигателей прогресса в информационной сфере на сегодняшний день. С каждым годом игры обретают новые возможности, а системные требования возрастают, как следствие, появляется новое техническое оборудование, позволяющее в полной мере раскрыть технический потенциал игры.

Цель ВКР: разработка игрового приложения для привлечения абитуриентов при поступлении в ХТИ – филиал СФУ на направление «Прикладная информатика».

Для достижения этой цели были поставлены следующие задачи:

– проанализировать существующие практики для привлечения абитуриентов;

– выбрать и обосновать жанр, в котором будет разрабатываться игра;

– проанализировать и обосновать средства разработки игры;

– разработать игру;

– провести оценку экономической эффективности игры.

В первом разделе будет проведен анализ предметной области и ее бизнес-процессов, будет построена функциональная модель IDEF0 и обоснован выбор программных средств.

Во втором разделе будет подробно описана разработка игры.

В третьем разделе будут рассчитаны капитальные, эксплуатационные и прямые затраты, а также экономическая эффективность проекта.

1 Анализ предметной области. Выбор средств проектных решений

1.1 Организационно-экономическая характеристика предметной области

Заказчиком проекта является Хакасский технический институт – филиал СФУ. Данному учебному заведению требуется программный продукт, который будет привлекать абитуриентов для поступления на направление «Прикладная информатика».

Краткое наименование: ХТИ – филиал СФУ. Юридический адрес учреждения: Республика Хакасия, г. Абакан, ул. Щетинкина, 27 (корпус "А"), 655017. Контактный телефон: (8 3902) 22-53-55.

Основной задачей ХТИ – филиала СФУ является создание передовой образовательной, научно-исследовательской и инновационной инфраструктуры, продвижение новых знаний и технологий для решения задач социально-экономического развития Сибирского федерального округа, а также формирование кадрового потенциала – конкурентоспособных специалистов по приоритетным направлениям развития Сибири и Российской Федерации, соответствующих современным интеллектуальным требованиям и отвечающих мировым стандартам.

Институт имеет сертификат соответствия системы менеджмента качества требованиям ГОСТ ІЗО 9001-2011 (ІЗО 9001:2008) применительно к деятельности в области образования.

Институт осуществляет подготовку по очной и заочной формам обучения, по 1 направлению специалитета, 6 направлениям бакалавриата и 1 направлению магистратуры. Ведётся подготовка специалистов и бакалавров по направлениям:

– 08.03.01 Строительство;

– 08.05.01 Строительство уникальных зданий и сооружений;

– 09.03.03 Прикладная информатика;

– 13.03.02 Электроэнергетика и электротехника;

– 15.03.05 Конструкторско-технологическое обеспечение машиностроительных производств; по ширине

– 38.03.01 Экономика;

– 23.03.03 Эксплуатация транспортно-технологических машин и комплексов. по ширине

Структурная схема ХТИ – филиала СФУ представлена на рисунке 1.

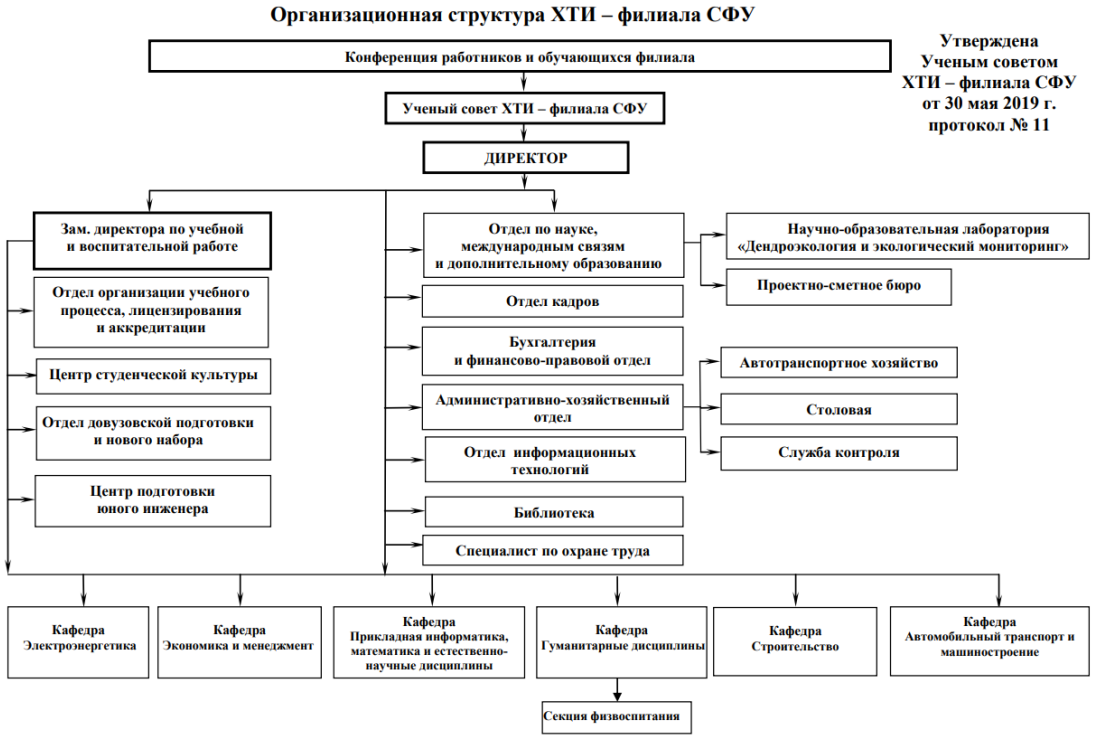

Рисунок 1 – Структурная схема ХТИ – филиала СФУ

Рассмотрим более подробно деятельность кафедры «Прикладная информатика, математика и естественно-научные дисциплины», которая осуществляет подготовку бакалавров по направлению «09.03.03 Прикладная информатика». Выпускник, освоивший данное направление, должен уметь решать следующие профессиональные задачи:

– проведение обследования прикладной области в соответствии с профилем подготовки: сбор детальной информации для формализации требований пользователей заказчика, интервьюирование ключевых сотрудников заказчика;

– разработка проектов автоматизации и информатизации прикладных процессов и создание информационных систем в прикладных областях;

– моделирование прикладных и информационных процессов, описание реализации информационного обеспечения прикладных задач;

– составление технико-экономического обоснования проектных решений и технического задания на разработку информационной системы;

– проектирование информационных систем в соответствии со спецификой профиля подготовки по видам обеспечения (программное, информационное, организационное, техническое);

– программирование приложений, создание прототипа информационной системы, документирование проектов информационной системы на стадиях жизненного цикла, использование функциональных и технологических стандартов;

– применение системного подхода к информатизации и автоматизации решения прикладных задач, к построению информационных систем на основе современных информационно-коммуникационных технологий и математических методов;

– осуществление технического сопровождения информационных систем в процессе ее эксплуатации; информационное обеспечение прикладных процессов [2].

В настоящее время существует острая потребность в специалистах, которые не только обладают знаниями и навыками в информационных технологиях, но и в значительной степени обладают знаниями предметной области. Интенсивно развиваются информационные технологии в государственном и муниципальном секторе, а также в сфере бизнеса. И это в значительной степени определяет востребованность IT-специалистов. В связи с этим совокупность полученных во время обучения знаний и навыков позволяет выпускникам вуза значительно расширить круг выбора вакансий на современном рынке труда.

1.2 Описание бизнес-процессов предметной области

В Хакасском техническом институте – филиале СФУ, как и в любом учебном заведении, существует практика проведения различных мероприятий для привлечения абитуриентов: дни открытых дверей; посещение школ, с целью информирования учащихся об имеющихся направлениях обучения; профориентационные мероприятия, которые позволяют будущим абитуриентам, если и не определиться с желаемым направлением, то хотя бы понять, какие профессиональные области вызывают больший интерес, а также различные конкурсы и олимпиады. Кроме того, за получение призового места, в некоторых из них, могут начислить дополнительные баллы при поступлении, как например, в олимпиадах «Бельчонок», «Ищем Ломоносовых» и других [3].

Рассматривая данные олимпиады, можно сделать вывод, что они могут привлекать абитуриентов к поступлению в данный вуз, так как часто происходят ситуации, что абитуриентам не хватает нескольких баллов для поступления. Тем не менее, они скорее выполняют свою целевую задачу (предоставление возможности получить дополнительные баллы), чем осуществляют функцию привлечения абитуриентов поступить в данный институт. Кроме того, это подразумевает то, что будущий студент заранее знает, куда будет поступать, чтобы получить соответствующие достижения. Так как даже участие в олимпиадах института начинается с осени прошедшего года перед поступлением.

Вообще, подобная система могла быть реализована, например, в виде компьютерного или мобильного аналога олимпиады, однако все же такой вид проведения не является достаточно привлекательным для абитуриентов. И на то есть вполне очевидная причина. Отвечать на тестовые вопросы и решать определенные задачки – это достаточно монотонный и скучный процесс. Иногда встречаются творческие задачи, которые могут разнообразить процесс. Но, опять же, нельзя нагружать данный процесс одними лишь творческими заданиями, так как это будет так же утомительно.

Однако, в связи с отсутствием юридических и технических основ, а также из-за перевода института в режим дистанционного обучения было решено, что игра будет реализовывать только функцию привлечения студентов на направление «Прикладная информатика», так как в связи с такой ситуацией не было возможности присутствовать в институте для создания технической основы, которая была нужна для обеспечения возможности начисления дополнительных баллов.

Адаптация игровых методов к неигровым процессам для большей вовлеченности, называемая геймификацией, имеет большую популярность в сфере образования, а в данном случае для привлечения абитуриентов. Это обосновывается тем, что самый большой процент людей, играющих в компьютерные игры, приходится на категорию лиц около 18 лет и младше [4]. А значит, это поспособствует привлечению абитуриентов в институт, например, у них может появиться желание научиться разрабатывать подобные игры.

1.3 Анализ литературы и других источников информации по функционированию подобных систем

Подобная практика существует в Томском политехническом университете, которая представлена в виде интерактивных обучающих игр. На данный момент их существует три:

– Интерактивный обучающий курс «Экспедиция» представляет собой игру, в которой представлены основные понятия и определения о полезных ископаемых, их химические и физические свойства, особенности. Игра представляется собой 2D-платформер, в котором необходимо пройти контрольные точки, собрав, все предметы, попутно узнавая про специальность геологии;

– Научная игра «Пилигримы» представляет собой интерактивный квест, в котором игрока ждут экспедиции в загадочное прошлое Земли вместе с Институтом времени. Путешествуя с различными заданиями по древним эпохам планеты, начинающий пилигрим не только непосредственно знакомится с

геологическим, биологическим и химическим миром минувшего, но и выполняет ряд поручений, некоторые из которых влияют и на события в настоящем времени;

– Join.TPU представляет собой видеоквест, который проходит в определенный срок. Вам предоставляется возможность почувствовать себя студентом ТПУ: пройти обучение, начиная с поступления в университет и заканчивая получением диплома [5].

Все представленные системы в той или иной степени можно соотнести с жанром квестовых игр. Первые две игры обладают схожими чертами – это типичные 2D-платформеры, геймплей которых заключается в исследовании мира и собирании предметов. Для технической реализации подобной игры требуются несколько углубленные знания разработки, но реализация может быть выполнена за достаточно небольшой срок (1-2 месяца). Последний проект является большим по объему, так как необходимо было отснять (во все времена года) и отредактировать много видеоматериала.

1.4 Обоснование и обзор выбранного жанра игры

Существует различное множество жанров компьютерных игр: активные шутеры, требующие от игрока высокого уровня концентрации и быстрой реакции, логические головоломки и квесты, различные ролевые игры, где игроку необходимо вжиться в роль сюжетного персонажа и так далее.

Каждый из жанров может быть исполнен как в 2D, так и в 3D, однако, есть некоторые особенности, которые делают тот или иной формат предпочтительнее. Конечно, 3D игры более привлекательные для игроков, нежели 2D, так как они предлагают большую реалистичность, как графическую, так и игровую, да и, в целом, 2D графика несколько устарела, однако, 2D-игры все еще популярны среди определенного, да и довольно немалого числа игроков [6, 7, 8].

Преимущества и недостатки двух видов исполнения игр представлены в таблице 1.

Таблица 1 – Сравнение 2D и 3D игр

	2D	3D
Преимущества	Достаточно низкая стоимость разработки. Не требует от разработчика большого опыта работы. Идеально подходит для создания малых и средних проектов за небольшой срок. Максимально подходящие жанры: квесты, платформеры и головоломки, Возможность сочетания необычных вариаций графики и геймплея.	Реалистичная и красивая графика. Более живописный, интересный и проработанный игровой мир. Больше возможностей сделать геймплей захватывающим. Возможность создания проекта любой сложности.
Недостатки	2D игры не так привлекательны для рядового пользователя. Рынок разработки меньше, а значит меньшее количество потребителей. Невозможность создания больших проектов вследствие ограниченных возможностей разработки в двух плоскостях.	Высокая стоимость и сложность разработки. Часто требуется большая команда разработчиков, в которой каждый отвечает за отдельный аспект игры.

Таким образом, можно сделать вывод, что 2D игра будет идеальным выбором, так как не требуется большого опыта разработки видеоигр, проект может быть разработан за небольшое количество времени и нет необходимости в больших затратах на разработку. Также жанром игры станет именно платформер, так как это наиболее популярный жанр 2D игр на данный момент, вдобавок, имеющий достаточно большие возможности организовать увлекательный игровой процесс [9].

2D-платформер – жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам и собирание предметов, обычно необходимых для завершения уровня. Все действия происходят в вертикальной и горизонтальной плоскостях. 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 [10].

1.5 Описание геймплея разрабатываемой игры

Разрабатываемая игра «IT-Adventurer» – это 2D-платформер, который включает в себя элементы квеста. Игрок наблюдает за игровым процессом посредством расположенной сбоку камеры. Он может перемещаться влево или вправо, а также вверх в пределах камеры.

Суть игры заключается в том, чтобы игрок набрал как можно больше очков посредством сбора монет, уничтожения противников, а также путем активации специальных флажков, при взятии которых игроку необходимо ответить на несколько вопросов, касающихся сферы информационных технологий. В игре содержится три уровня, которые объединены в один непрерывный. В зависимости от уровня меняются противники и окружение. Существует три вида врагов: неподвижные, перемещающиеся, неподвижные, но стреляющие. Каждого из них можно нейтрализовать как прыжком сверху, так и при помощи выстрела, однако, во втором случае игрок получит меньше очков. При столкновении с ними с любой другой стороны игрок получит урон, равный одной жизни. Получить жизни можно путем подбора сердечек, расположенных по всей игре, или отвечая на вопросы при взятии флажка. Пять правильных ответов даруют одну жизнь, а пять неправильных ее отнимают. Если число жизней упадет до нуля, то игру придется начать сначала. Если игрок дойдет до финиша, то ему отобразится сообщение с оценкой его результата в сравнении с максимально возможным.

Различные блоки и платформы, из которых будут состоять уровни, представлены на рисунке 3. Некоторые из них могут активироваться игроком. При касании такого блока он может исчезнуть, таким образом, в игре на всех уровнях спрятаны несколько секретных областей, в которых находятся

дополнительные жизни или монеты. Еще есть падающие блоки, которые требуют от игрока скорости реакции, иначе он уже не сможет забраться в некоторые места.

1.6 Моделирование бизнес-процесса «Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Моделирование информационной системы основывалось на методологии функционального моделирования IDEF0 [11]. Особенностью IDEF0 является её акцент на соподчинение объектов. Для этого были определены следующие данные (рисунок 2):

Входы:

– квестлайн – последовательность заданий, которые должен будет выполнить игрок для успешного прохождения игры;

– информация из внешних источников – данный пункт необходим для проведения анализа, который необходим перед началом разработки, например, анализ аналогов;

Механизмы:

– разработчик – выполняет роли аналитика, программиста, дизайнера, а именно, анализирует различные данные, создает программный код и графику, а также, в целом, выполняет все действия, связанные с разработкой проекта.

– Photoshop – программное обеспечение, необходимое для работы с графикой;

– Unity – игровой движок, на котором будет осуществляться разработка игры.

Управление:

– договор с заказчиком – необходим для установления сроков выполнения и сдачи проекта, а также для соблюдения обеими сторонами своих обязательств;

– ГОСТ Р «ИСО 9001-2001 Системы менеджмента качества. Требования»;

– ГОСТ «ISO/IEC/IEEE 29148:2018 Программная и системная инженерия. Процессы жизненного цикла. Разработка требований»;

– сеттинг – стиль, в котором будет выполнена игра. Это касается не только графической составляющей, но и игровых механик, чтобы они подходили игровому миру, а также сам персонаж вписывался в окружение.

Выходы:

– готовая игра – полностью завершенный программный продукт, удовлетворяющий заказчика и готовый к внедрению.

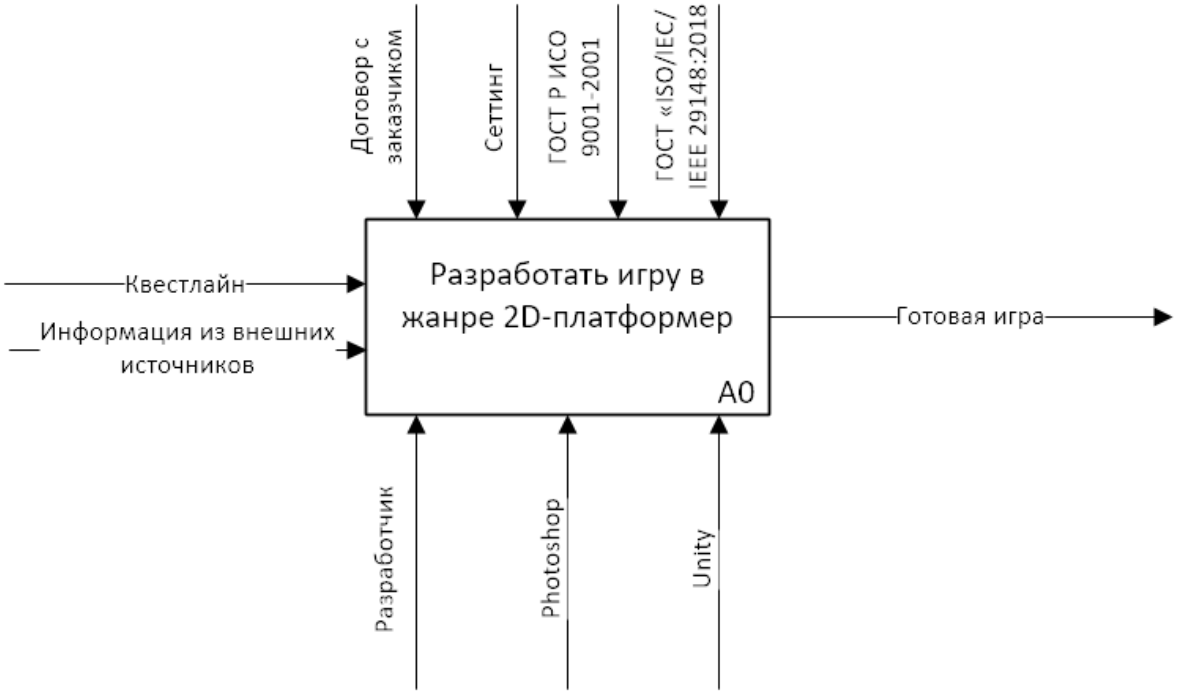

Рисунок 2 – Функциональная модель проекта

Выполним декомпозицию функциональной модели для составления перечня задач (рисунок 3):

– провести анализ аналогичных продуктов;

– создать техническое задание;

– создать прототип игры;

– выполнить тестирование;

– сдать проект заказчику.

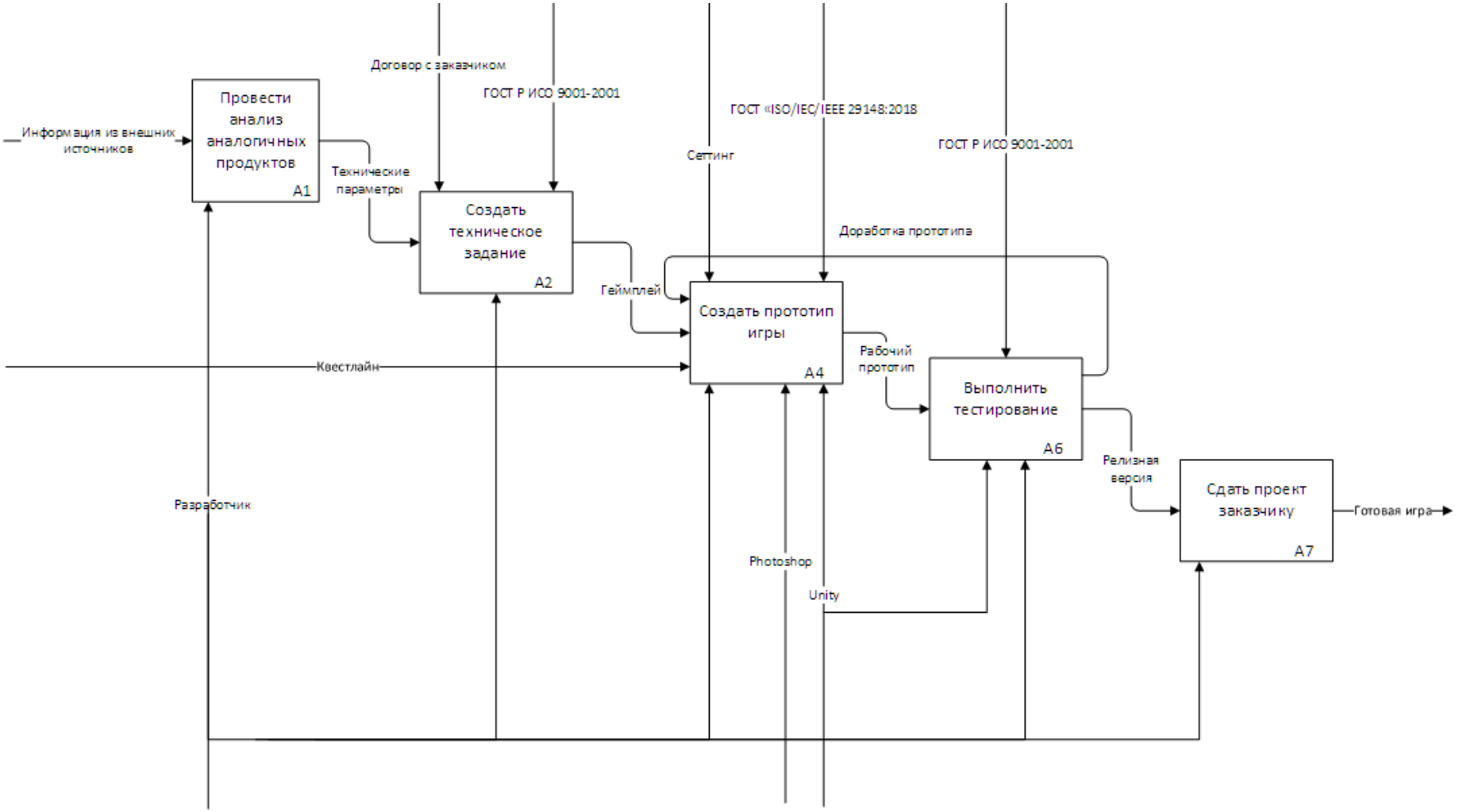

Рисунок 3 – Декомпозиция функциональной модели проекта

Суть блока декомпозиции «Провести анализ аналогичных продуктов» заключается в том, чтобы проанализировать программный рынок на наличие подобного рода информационных систем, чтобы создаваемый продукт имел преимущество перед своими конкурентами.

Суть блока декомпозиции «Создать техническое задание» заключается в том, чтобы четко определить этапы разработки и время их выполнения, план тестирования и методы проверки на наличие ошибок, а также определить задачи, необходимые для создания программы.

Суть блока декомпозиции «Создать прототип игры» заключается в том, чтобы создать некоторую версию игры, которая будет, в целом, работоспособная, но при этом некоторые функции могут выполняться в ней некорректно.

Суть блока декомпозиции «Выполнить тестирование» заключается в том, что на данном этапе происходит тестирование имеющейся версии игры. При наличии ошибок текущая версия отправляется на этап создания.

Суть блока декомпозиции «Сдать проект заказчику» заключается в том, чтобы представить заказчику разработанную и полностью завершенную игру, которая затем будет внедрена в институте для использования.

Затем выполним декомпозицию блока «Создать прототип игры» для понимания процессов, которые происходят на основной стадии создания программы (рисунок 4).

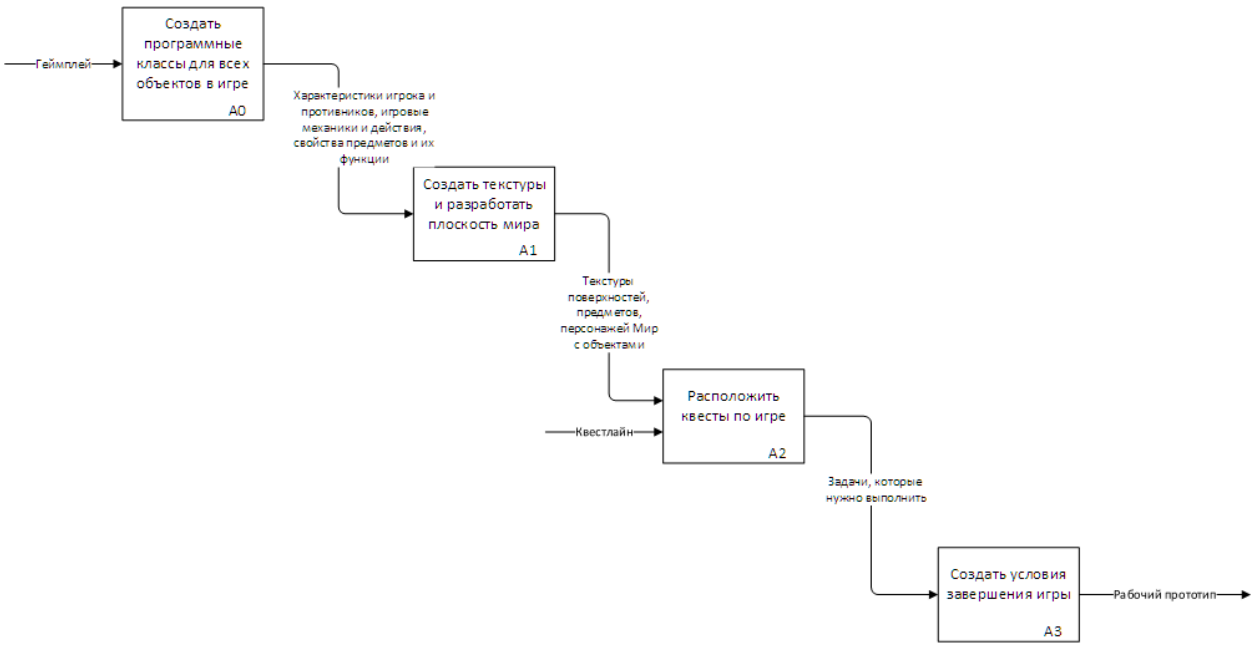

Рисунок 4 – Декомпозиция функционального блока «Создать прототип игры»

Разработка прототипа игры начинается с создания игровых классов для всех игровых объектов: блоки, игрок, противники, предметы, камера, следующая за игроком и т.д.

Затем создается графика: интерфейс игрового меню, модели персонажей, текстуры поверхностей и предметов, различные фоны.

Далее происходит генерация самого мира. Располагаются блоки игрового мира, платформы, предметы, противники и т.д. Таким образом, создается несколько уровней.

Потом созданные задания, которые будет выполнять игрок, необходимо расположить по ходу игры так, чтобы это было ненавязчиво, то есть через некоторые приблизительно равные промежутки.

В самом конце прописываются условия для победы и для поражения, то есть создаются триггеры для завершения игры, например, если игрок дошел до конца и выполнил успешно все задания, то он победил, а если уровень здоровья игрока стал равен нулю, то он проиграл.

1.7 Обоснование и выбор средств проектных решений

Для проекта было решено взять готовый игровой движок, так как абсолютно нецелесообразно писать свой собственный из-за повышения затрат ресурсов и достаточно небольшого масштаба проекта. Игровой движок должен удовлетворять следующим требованиям:

– возможность разработки на языке программирования C# – требование обусловлено тем, что данный язык программирования очень распространен, и у разработчика имеется определенный опыт работы с ним;

– наличие встроенной системы тестирования приложения – требование обусловлено большей удобностью разработки;

– возможность бесплатного использования – требование обусловлено небольшим капиталом на разработку;

– игровой движок должен быть достаточно распространен – требование обусловлено тем, что опыт разработчика в сфере компьютерных игр очень мал, поэтому наличие большого количества документации играет важную роль.

Теперь рассмотрим существующие популярные решения, подходящие под данные требования.

Unity

Unity – это инструмент для разработки двух- и трёхмерных приложений и игр, работающий под операционными системами Windows, OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Windows Phone, Апаго4, Apple iOS, Linux, а также на игровых консолях Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One. Есть возможность создавать приложения для запуска в браузерах с помощью специального подключаемого модуля Unity (Unity Web Player), а также с помощью реализации технологии WebGL.

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 три сценарных языка: C#, JavaScript (модификация), Boo (диалект Python). Расчёты физики производит физический движок PhysX от NVIDIA.

Проект в Unity делится на сцены (уровни) – отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев и настроек. Сцены могут содержать в себе как, собственно, объекты (модели), так и пустые игровые объекты – объекты, которые не имеют модели (пустышки). Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. Также у объектов есть название (в Unity допускается наличие двух и более объектов с одинаковыми названиями), может быть тег (метка) и слой, на котором он должен отображаться. Так, у любого объекта на сцене обязательно присутствует компонент Transform – он хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент Mesh Render, делающий модель объекта видимой.

Unity поддерживает систему Level Of Detail, суть которой заключается в том, что на дальнем расстоянии от игрока высоко детализированные модели заменяются на менее детализированные, и наоборот, а также систему Occlusion culling, суть которой в том, что у объектов, не попадающих в поле зрения камеры не визуализируется геометрия и коллизия, что снижает нагрузку на центральный процессор и позволяет оптимизировать проект [12].

Unreal Engine 4

Unreal Engine – игровой движок, разрабатываемый и поддерживаемый компанией Epic Games. Первая игра, созданная на этом движке (Unreal) появилась в 1998 году.

Написанный на языке C++, данный движок позволяет создавать игры для большинства операционных систем и платформ: Microsoft Windows, Linux, Mac OS и Mac OS X; консолей Xbox, Xbox 360, PlayStation 2, PlayStation 3, PSP, PS Vita, Wii, Dreamcast, GameCube и др., а также на

различных портативных устройствах, например, устройствах Apple (iPad, iPhone), управляемых системой iOS и прочих. (Впервые работа с iOS была представлена в 2009 году, в 2010 году продемонстрирована работа «Движка» на устройстве с системой webOS).

Для упрощения портирования движок использует модульную систему зависимых компонентов; поддерживает различные системы рендеринга (Direct3D, OpenGL, Pixomatic: в ранних версиях: Glide, S3, PowerVR), воспроизведения звука (EAX, OpenAl, DirectSound3D; ранее: A3D), средства голосового воспроизведения текста, распознавание речи, модули для работы с сетью и поддержки различных устройств ввода [13].

Сравнение этих двух игровых движков по нескольким критериям представлено в таблице 2.

Таблица 2 – Сравнение Unity и Unreal Engine

	Unity	Unreal Engine
Требуемый уровень знаний	Достаточно просто обучиться разработке небольших проектов, имея знания C#.	Достаточно просто научиться создавать небольшие проекты, имея знания C++, однако, также необходимы навыки моделирования.
Размеры проектов	Любые	Любые
Область применения	Различные проекты: 2D и 3D приложения, как для стационарных устройств, так и для мобильных. Имеет большой комплекс функций и инструментария для разработки 2D игр.	Имеет очень сильный функционал 3D разработки. Есть возможность разработки 2D при помощи такого инструмента, как Paper2D.
Платформы	Windows, macOS, Linux и другие актуальные платформы	Windows, macOS, Linux и другие актуальные платформы

Окончание таблицы 2

Стоимость	Бесплатно, если годовой доход составляет менее $100 000;$40 в месяц, если годовой доход не превышает $200 000; $150 в месяц, если годовой доход больше $200 000.	Бесплатно, если доход игры не превышается $3000 за квартал, иначе 5% с продаж отчисляются Epic Games.

В результате изучения и анализа данных игровых движков, их возможностей и характеристик было решено выбрать Unity для реализации проекта, так как для разработки 2D-игр он является более удобным из-за того, что обладает большим функционалом и поддержкой 2D-разработки.

Однако, игровой движок – это не единственное программное обеспечение, требуемое для создания игры. Также, необходимо использование Microsoft Visual Studio 2017 для создания функционального кода игры и Adobe Photoshop для работы над визуальной составляющей игры. Ниже приведено описание этих программных средств.

Microsoft Visual Studio 2017 – полнофункциональная интегрированная среда разработки для написания, отладки, тестирования и развертывания кода на любой платформе. Данная среда разработки как нельзя лучше подходит для написания программного кода на языке C#. Во-первых, у разработчика имеется опыт работы с данной программой, а во-вторых, Visual Studio и Unity обладают отличной интеграцией.

Adobe Photoshop – программное обеспечение для графического дизайна и обработки изображений, главным преимуществом является большое количество обучающего материала для данного программного обеспечения и наличие очень разнообразных функций работы с изображениями.

1.8 Выводы по разделу «Анализ предметной области. Выбор средств проектных решений»

Таким образом, можно подвести итоги аналитического раздела:

– выполнен анализ предметной области и ее бизнес-процессов. Было выявлено, что в ХТИ – филиале СФУ проводятся различные мероприятия (конкурсы, олимпиады, профориентация и т.д.), которые выполняют функцию привлечения абитуриентов;

– построена модель бизнес-процесса разработки игры в нотации IDEF0, а также выполнена детальная декомпозиция;

– выполнен обзор подобных функционирующих систем и сделан вывод о том, что они из себя представляют;

– был описан сценарий разрабатываемой игры, механики и ее правила;

– были обоснованы выбор жанра игры, а также средства ее разработки. Игра будет представлять собой 2D-платформер, так как это самое выгодное и популярное решение, и для ее разработки игровой движок Unity будет идеальным вариантом. Для работы с графикой будет использоваться Adobe Photoshop, как универсальное решение.

2 Описание разработки игры

2.1 Общий алгоритм функционирования

Общий алгоритм функционирования можно представить при помощи UML-диаграммы состояний (рисунок 5) которая описывает все состояния системы: перед началом игры, в течение и после завершения.

Рисунок 5 – Диаграмма состояний игры

Рассмотрим подробнее каждый блок алгоритма:

– загрузка уровня – пользователь запускает игру, и происходит загрузка уровня, затем он может приступать непосредственно к игре;

– игровой процесс – в этом блоке происходят все основные действия, такие как генерация противников, их передвижение и самого игрока, контроль здоровья врагов и игрока, а также выполнение заданий, расчет стрельбы и так далее. Также проверяются условия, которые приводят к другому блоку;

– конец игры – игровой процесс далее невозможен вследствие того, что игрок либо полностью прошел ее, либо его здоровье стало равно нулю. Выводится его результат игры, после чего предоставляется возможность начать заново или закрыть игру.

2.2 Создание игровых классов и объектов

Класс или скрипт – это программный код, который содержит в себе описание некоторого игрового объекта.

Разработка прототипа игры начинается с создания общего класса и нескольких функций и свойств, которые будут наследоваться следующими классами. Например, здоровье персонажа, получение и нанесение урона. Так как этими свойствами обладают как игрок, так и несколько видов

противников, то нецелесообразно каждому их классу прописывать данные свойства.

Таким образом, можно упростить количество строк повторяющегося кода и улучшить менеджмент ресурсов игры.

Поэтому создадим класс «Unit». В нем будут содержаться всего два метода RecieveDamage и Die. Они будут отвечать за то, что при получении урона игровой объект будет уничтожаться. Очевидно, что это закономерно как для игрока, так и для противника. Класс представлен на рисунке 6.

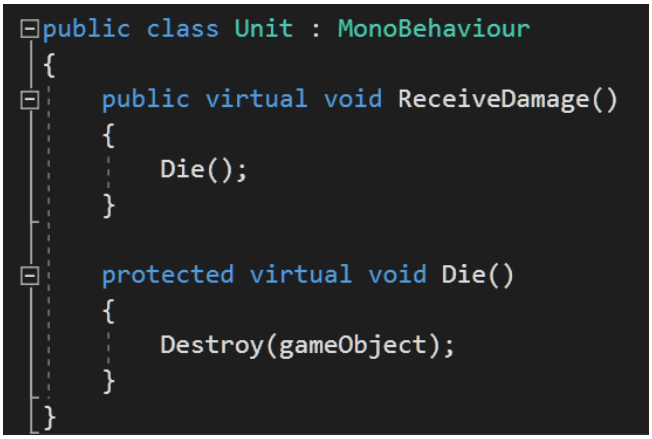

Рисунок 6 – Основной класс «Unit»

2.2.1 Класс игрока

Рассмотрим последовательность создания игрового объекта на примере создания класса игрока «Character». Для начала необходимо создать пустой объект. После чего необходимо создать дочерний 2D элемент «Sprite» (в 2D-играх спрайт – это растровый графический объект). Данная схема будет закономерно использоваться для создания всех последующих объектов. Алгоритм создания представлен на рисунке 7.

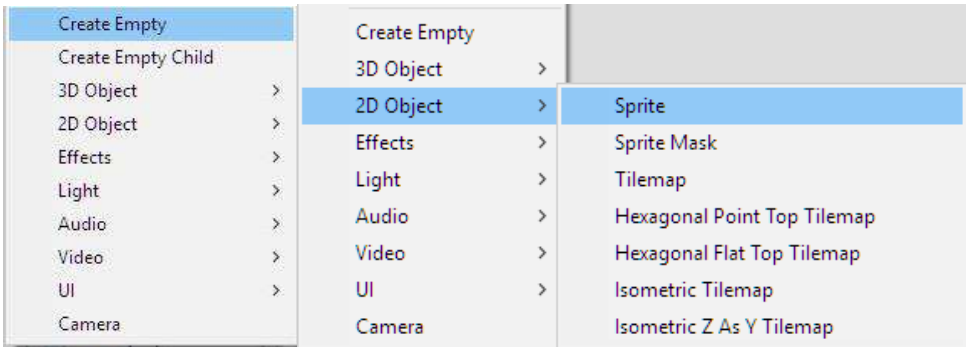

Рисунок 7 – Алгоритм создания игрового объекта и его спрайта

Назовем этот объект «Character» и присвоим ему спрайт (рисунок 8).

Рисунок 8 – Присвоение спрайта объекту

Затем необходимо создать анимацию для действий игрока: если он стоит, бежит или прыгает. Для этого необходимо зайти в подменю «Animation», создать «Sprite» и расположить на временной шкале несколько спрайтов. В зависимости от их количества зависит плавность и детализация анимации. На рисунке 9 представлено меню создания анимации.

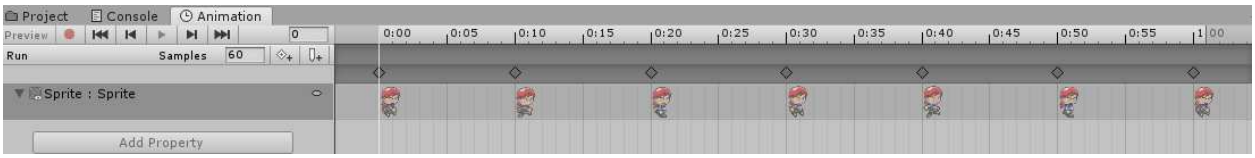

Рисунок 9 – Создание анимации бега для «Character»

Таким же образом создается анимация прыжка и нахождения на месте. Затем необходимо при помощи средства «Animator» связать между собой все анимации (рисунок 10).

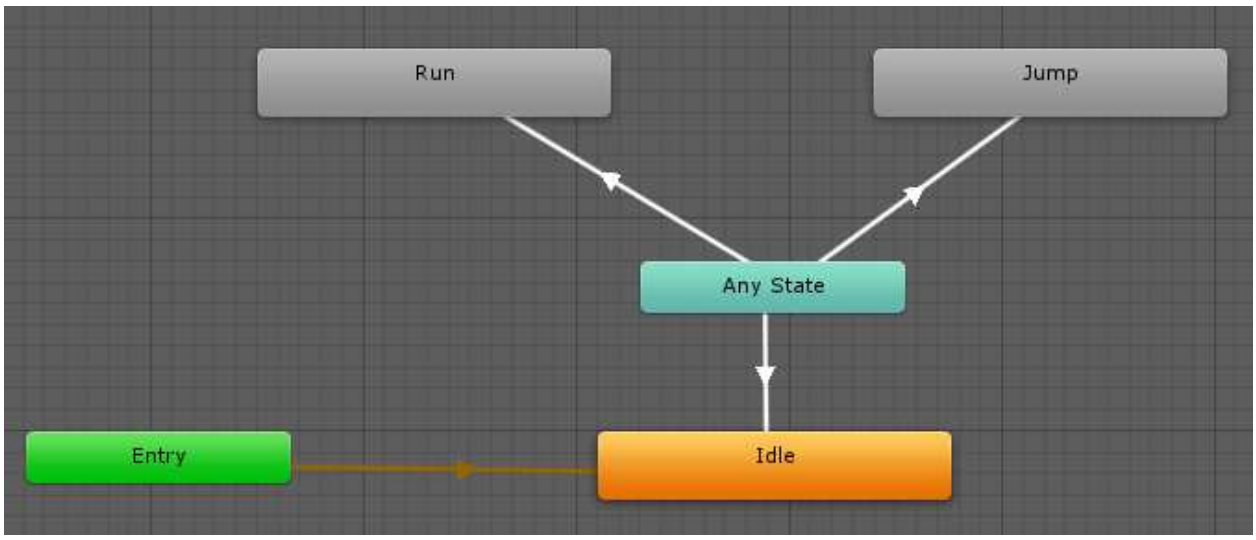

Рисунок 10 – Связь анимации через Any State

Класс «Character» является самым массивным из всех, так как он отвечает не только за действия персонажа, но и за запуск и обработку определенных событий, которые запускаются, например, если модель игрока коснулась игрового триггера (механизма, проверяющего присутствие каких-либо объектов игрового мира в заданном пространстве или расстояние от этих объектов до специальной точки).

Для взаимодействия с игровым миром игроку нужна физическая модель. При помощи компонента Physics 2D необходимо добавить элементы: Box Collider 2D, который отвечает за обработку столкновений с другими объектами, и Rigidbody 2D – физическая модель игрока для взаимодействия с физикой мира (рисунок 11).

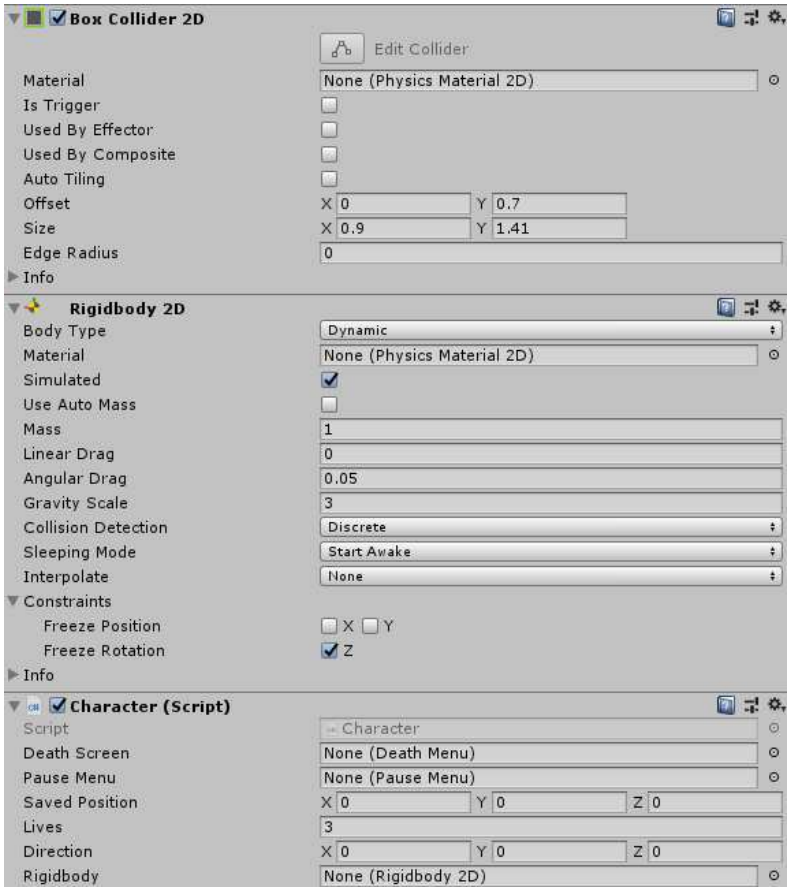

Рисунок 11 – Добавление компонентов Physics 2D

2.2.2 Классы противников

Как и в случае с созданием игрока, для начала необходимо создать пустые объекты: «Monster», «MovableMonster» «ShootableMonster» и «VerticalMovingMonster». Присвоим спрайты, настроим анимацию и добавим компоненты Physics 2D, так как противники тоже взаимодействуют с физикой мира. После чего необходимо создать для каждого из них свой класс. Но, так как все из них являются противниками, то некоторые свойства у них будут общие. Создадим класс «Monster», который наследует методы из «Unit». Но, кроме того, он будет являться родительским для последующих классов других противников, которых в игре есть еще два вида (перемещающиеся и стреляющие), то в классе «Monster» создадим методы Awake, Start и Update, которые будут ими использоваться. Protected virtual значит, что к данным методам могут иметь доступ (читать и перезаписывать) только наследующие классы. Код класса «Monster» представлен на рисунке 12.

```
public class Monster : Unit
{
    protected virtual void Awake() { }
    protected virtual void Start() { }
    protected virtual void Update() { }


    protected virtual void OnTriggerEnter2D(Collider2D collision)
    {
        Bullet bullet = collision.GetComponent<Bullet>();
        Unit unit = collision.GetComponent<Unit>();

        if (bullet)
        {
            ReceiveDamage();
            FindObjectOfType<ScoreBar>().Score += 5;
        }

        if (unit && unit is Character)
        {
            if (Mathf.Abs(unit.transform.position.x - transform.position.x) < 0.7F)
            {
                ReceiveDamage();
                FindObjectOfType<ScoreBar>().Score += 10;
            }
            else unit.ReceiveDamage();
        }
    }
```

Рисунок 12 – Класс «Monster»

Далее необходимо создать классы для оставшихся типов противников. Класс «MoveableMonster» задает методы и функции противника в игре, который умеет перемещаться по горизонтали (рисунок 13).

Рисунок 13 –Противник, перемещающийся горизонтально

Класс «ShootableMonster» задает методы и функции противника в игре, который является неподвижным, но может выпускать в игрока летящие снаряды (рисунок 14).

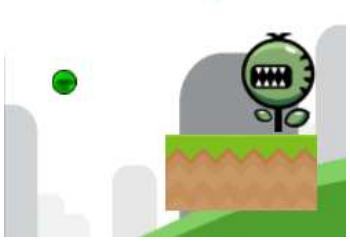

Рисунок 14 – Противник, умеющий стрелять

Класс «VerticalMovingMonster» задает методы и функции противника в игре, который умеет перемещаться по вертикали, но ему нельзя нанести урон (рисунок 15).

Рисунок 15 –Противник, перемещающийся вертикально

Теперь на основе существующих классов можно создать противников с другими спрайтами. Все имеющиеся противники представлены на рисунке 16.

Рисунок 16 –Противник, перемещающийся вертикально

2.2.3 Прочие классы

Помимо игрока и противников для игры необходимо создать следующие классы: «Bullet», «CameraController», «Coin», «DeathMenu», «DisableBlock», «FallBlock», «Heart», «HorizontalMovablePlatform», «InstructionMenu», «LivesBar», «MainMenu», «PauseMenu», «QuestMenu», «QuestTrigger», «Restart», «ScoreBar», «SpikeFall», «VerticalMovablePlatform», «WinMenu».

Каждый из них отвечает за определенную функцию в игре, вытекающую из названия, а именно:

Класс «Bullet» отвечает за свойства объекта «Пуля», а именно, ее создание и кем она создается, математическую функцию ее перемещения и ее отображение.

Класс «CameraController» отвечает за то, чтобы камера следовала за игроком.

Класс «Coin» отвечает за подбор монет игроком и начисление очков.

Класс «DeathMenu» отвечает за отображение интерфейса после того, как игрок потеряет все свои жизни.

Класс «DisableBlock» отвечает за особые блоки, которые отключаются при контакте с игроком.

Класс «FallBlock» отвечает за особые блоки, которые падают при контакте с игроком.

Класс «Heart» отвечает за игровой предмет «Жизнь», который может использоваться игроком для увеличения количества жизней.

Класс «HorizontalMovablePlatform» отвечает за перемещение платформ в горизонтальной плоскости.

Класс «InstructionMenu» отвечает за отображение интерфейса кнопок управления в игре.

Класс «LivesBar» отвечает за отображение и функционирования интерфейса количества оставшихся жизней игрока.

Класс «MainMenu» отвечает за отображение интерфейса главного меню.

Класс «PauseMenu» отвечает за остановку игры и отображение интерфейса паузы.

Класс «QuestMenu» отвечает за отображения интерфейса с вопросами, на которые игрок должен ответить перед началом уровня.

Класс «QuestTrigger» отвечает за обработку событий при активации флажка игроком.

Класс «Restart» отвечает за перезапуск уровня, если у игрока не осталось жизней, или если игрок захотел начать игру заново.

Класс «ScoreBar» отвечает за отображение количества набранных очков.

Класс «SpikeFall» отвечает за обработку событий при падении игрока в пропасть.

Класс «VerticalMovablePlatform» отвечает за перемещение платформ в вертикальной плоскости.

Класс «WinMenu» отвечает за отображение интерфейса при успешном прохождении игры.

2.3 Создание интерфейсов

Для создания интерфейсов используются компоненты Unity UI (User Interface) «Canvas», «Image», «Text», «Button».

2.3.1 Интерфейс вопросов

Иерархия элементов представлена на рисунке 17, а интерфейс – на рисунке 18. Вне зависимости от ответа в нижней части будет выведена соответствующая надпись (верный ответ или нет).

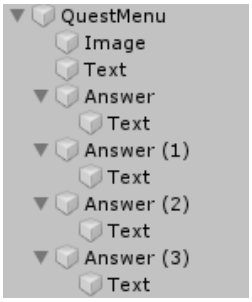

Рисунок 17 – Иерархия элемента «Canvas» для интерфейса вопросов

Рисунок 18 – Интерфейс вопросов

Для создания и хранения вопросов использовалась встроенная библиотека Unity System.Collections и System.Collections.Generic. Процесс хранения, добавления и редактирования вопросов и ответов показан на рисунках 19, 20.

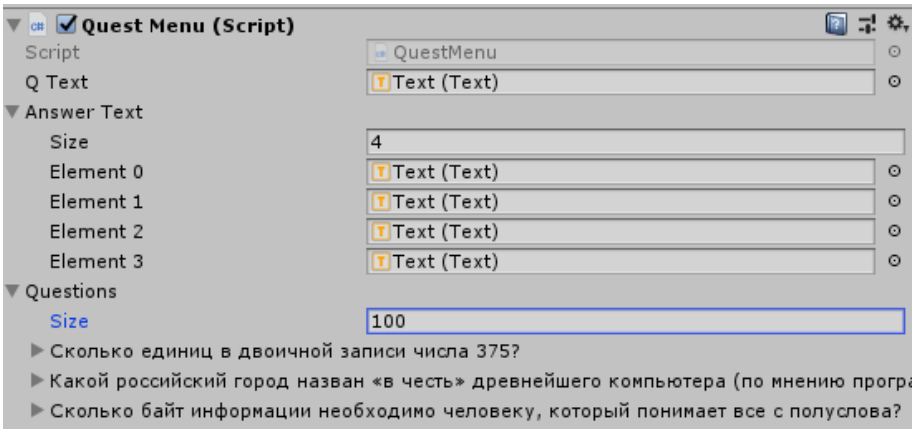

Рисунок 19 – Хранилище вопросов и ответов

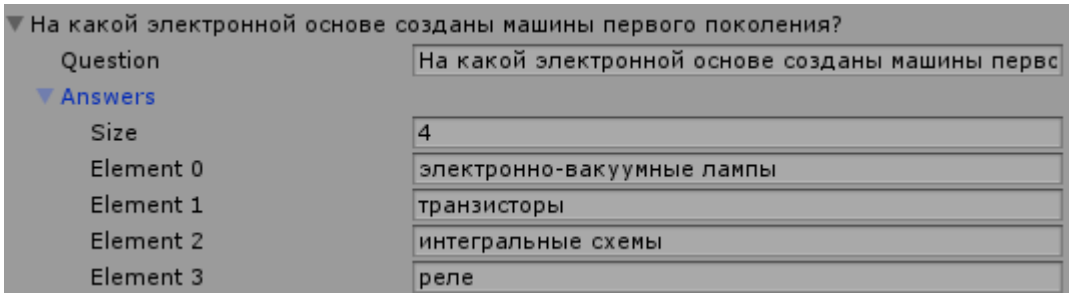

Рисунок 20 – Редактирование вопросов и ответов

Как можно заметить, данное хранилище находится под управлением скрипта «QuestMenu». Код генерации вопросов представлен на рисунке 21, а код обработки ответов на рисунке 22.

```
void Start()
{
    questBoard = gameObject.GetComponent<QuestBoard>();
    character = FindObjectOfType<Character>();
    qList = new List<object>(questions);
    QuestionGenerate();
void QuestionGenerate()
{
    if (qList.Count > 0)
    {
        randQ = Random.Range(0, qList.Count);
        currentQ = qList[randQ] as QuestionList;
        qText.text = currentQ.question;
        List<string> answers = new List<string>(currentQ.answers);
        for (int i = 0; i < currentQ.answers.Length; i++)
        {
            int rand = Random.Range(0, answers.Count);
            answerText[i].text = answers[rand];
            answers.RemoveAt(rand);
        }
    }
    else
    {
        Resume();
    }
}
```

Рисунок 21 – Код генерации вопросов

```
public void AnswerButton(int index)
{
    if (answerText[index].text.ToString() == currentQ.answers[0])
    {
        result.text = "Верно!";
        rightCount++;
        FindObjectOfType<ScoreBar>().Score += 50;
    }
    else
    {
        result.text = "Неверно!";
        wrongCount++;
    }
    if (rightCount == 5)
    {
        character.Lives++;
        rightCount = 0;
    }
    if (wrongCount == 3)
    {
        character.Lives--;
        wrongCount = 0;
    }
    qList.RemoveAt(randQ);
    QuestionGenerate();
}
```

Рисунок 22 – Код кнопки вариантов ответа

Когда данный скрипт становится активным (void Start), то происходит вывод графического интерфейса, генерируется лист вопросов и выводится сам вопрос на экран. Генерация вопросов (void QuestionGenerate) происходит по циклическому алгоритму: из созданного листа вопросов случайно выбирается один и удаляется из списка, таким же образом генерируются следующие вопросы. Если вопросы в списке закончились, то данное меню закрывается, и игрок может приступить к игре.

2.3.2 Главное меню

Главное меню содержит несколько кнопок: «Начать игру», «Об авторе» и «Выйти» (рисунок 23). При нажатии на кнопку «Начать игру» запустится новое меню (рисунок 24), в котором будет кратко рассказано об игре, что нужно делать, а также кнопки управления. При нажатии кнопки «Об авторе» откроется меню, которое содержит информацию о создателе данной игры. Кнопка «Выход» завершает работу приложения.

Рисунок 23 – Главное меню

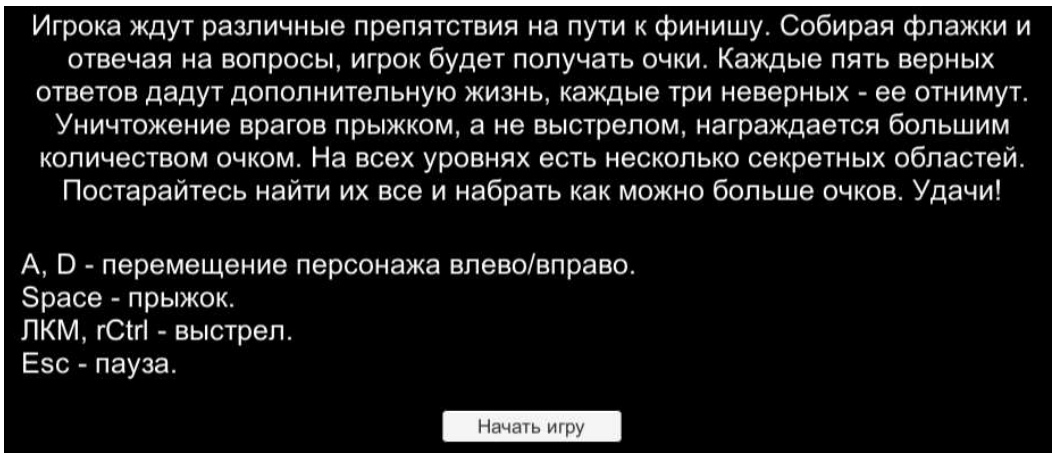

Рисунок 24 – Меню инструкции

2.3.3 Меню окончания игры

Если у игрока кончились жизни, то срабатывает событие и происходит отображение интерфейса завершения игры (рисунок 25). Если игрок дошел до конца и взял финишный флаг, то отображается победное меню (рисунок 26).

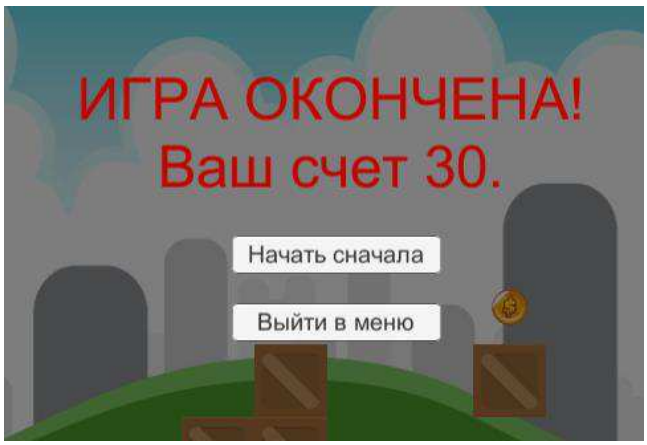

Рисунок 25 – Игрок проиграл

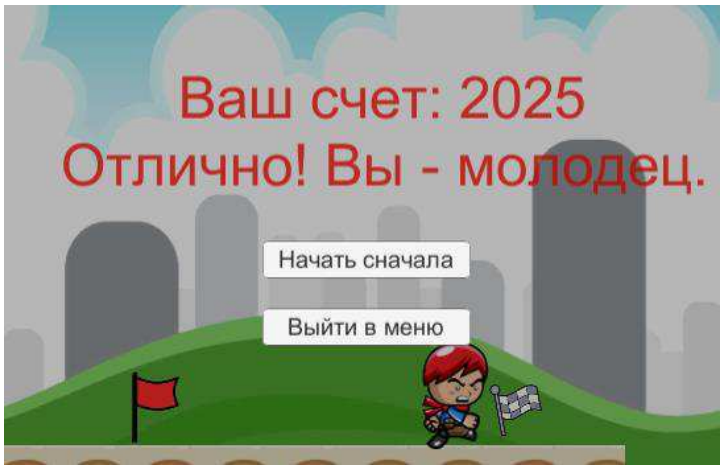

Рисунок 26 – Игрок прошел игру

2.4 Создание уровня

Во-первых, необходимо создать блоки, по которым будет перемещаться игрок. Для этого также создается пустой объект, которому добавляется спрайт и BoxCollider 2D. Компонент Rigidbody 2D нужен только для падающих блоков. На рисунке 27 представлены обычные блоки, на которые не действует гравитация.

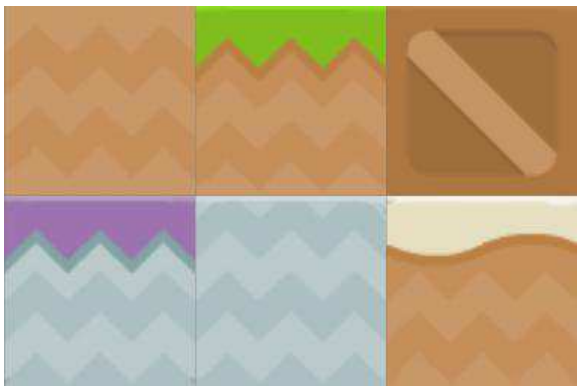

Рисунок 27 – Статичные блоки

Теперь необходимо создать исчезающие блоки. Они будут выглядеть так же, как и обычные, однако, если игра обнаружит коллизию (столкновение) между игроком и таким блоком, то блок разрушится. Код класса таких блоков представлен на рисунке 28.

```
public class DisableBlock : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        if (unit && unit is Character)
        {
            Destroy(gameObject);
        }
    }
```

Рисунок 28 – Код класса «DisableBlock»

Также, на последнем уровне будут находиться падающие блоки. Им необходим компонент Rigidbody 2D с определенными параметрами, которые

представлены на рисунке 29. Все созданные и используемые блоки в игре продемонстрированы на рисунке 30.

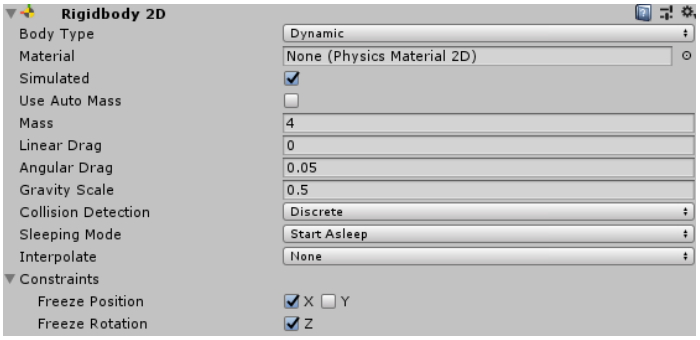

Рисунок 29 – Rigidbody 2D объекта «FallBlock»

Рисунок 30 – Игровые блоки

После чего расположим эти блоки в некотором порядке, чтобы получился игровой уровень. Также необходимо разместить флажки, которые активируют меню вопросов. Они должны располагаться через примерно равные промежутки. Также разместим противников и остальные предметы: дополнительные жизни, монеты и ловушки, чтобы получился цельный уровень. Созданные уровни представлены на рисунках 31-33.

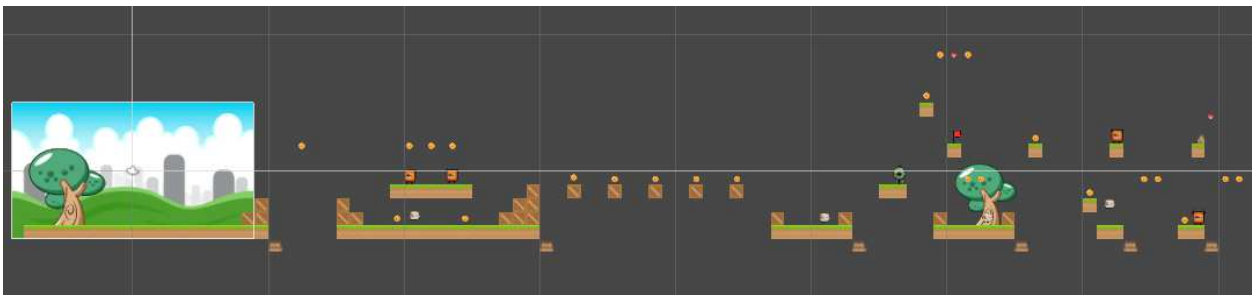

Рисунок 31 – Сцена первого уровня

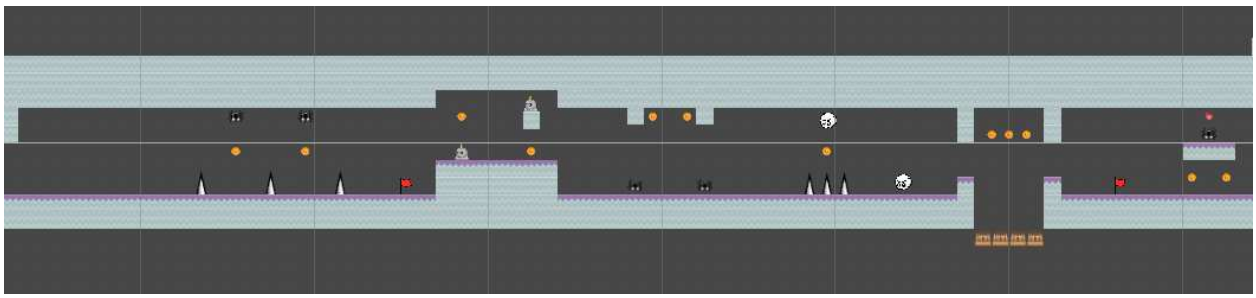

Рисунок 32 – Сцена второго уровня

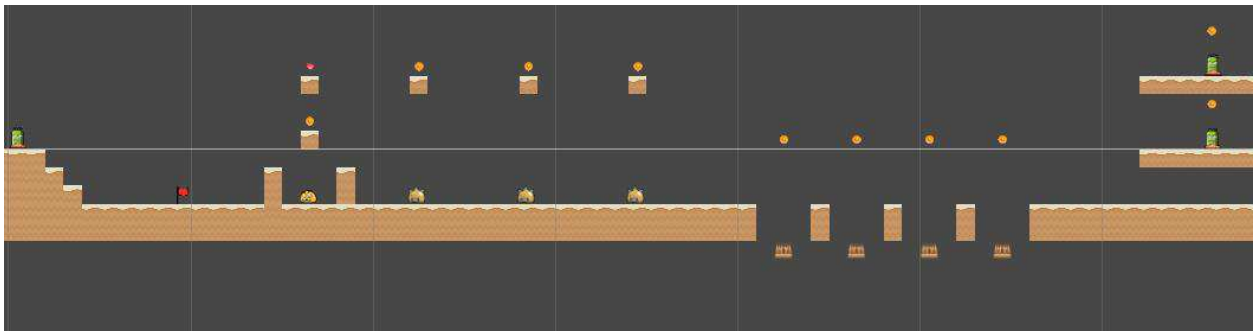

Рисунок 33 – Сцена третьего уровня

2.5 Выводы по разделу «Описание разработки игры»

Таким образом, можно подвести итоги проектного раздела:

– разработан 2D-платформер, отвечающий всем заданным требованиям. Созданы интерфейсы здоровья игрока, количества очков, главное меню и меню завершения игры. Созданы различные противники и реализована механика активации квестового меню (меню вопросов и ответов). В игре имеется три уровня. На данный момент имеется сто вопросов для игры. Это число можно без проблем увеличить;

– критические баги и ошибки были исправлены в процессе разработки;

– графика, используемая в игре, является находится в свободном доступе [14, 15];

– полный код всех игровых классов приведен в приложении А.

3 Расчёт затрат и оценка экономической эффективности реализации проекта «Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Сроки реализации проекта:

– консультации с заказчиком, анализ предметной области – 5 дней;

– разработка прототипа – 23 день;

– доработка – 7 дней;

– введение в эксплуатацию – 5 дней.

3.1 Расчет затрат реализации проекта

По данным из раздела 1 можно рассчитать затраты реализации по методике ТСО, которые рассчитывается по формуле 1:

$$TCO = DE + IC_1 + IC_2, \quad (1)$$

где DE – прямые затраты;

IC – косвенные затраты первой и второй группы.

Для данного проекта косвенные расходы не являются большими, поэтому ТСО ≈ DE.

3.1.1 Капитальные затраты

Расчет капитальных затрат производится по формуле 2:

$$К = К_{пр} + К_{тс} + К_{лс} + К_{по} + К_{ио} + К_{об} + К_{оэ}, \quad (2)$$

где $К_{пр}$ – затраты на разработку информационной системы;

$К_{тс}$ – затраты на технические средства управления;

$К_{лс}$ – затраты на создание линий связи, а также интернет соединения;

$К_{по}$ – затраты на программные средства для использования готового программного продукта;

$К_{ио}$ – затраты на формирование информационной базы;

$К_{об}$ – затраты на обучение персонала;

$К_{оэ}$ – затраты на опытную эксплуатацию.

Для данного проекта К = $К_{пр}$, так как отсутствуют остальные виды затрат.

Рассчитаем затраты на разработку ИС по формуле 3:

$$К_{пр} = К_{зп} + К_{ипс} + К_{свт} + К_{проч}, \quad (3)$$

где $К_{зп}$ – затраты на заработную плату разработчиков;

$К_{ипс}$ – затраты на инструментальные программные средства для проектирования;

$К_{свт}$ – затраты на средства вычислительной техники для проектирования;

$К_{проч}$ – прочие затраты на разработку.

Для разработки игры из оборудования необходим только компьютер. В данной ситуации ноутбук будет выгоднее, так как в таком случае разработчик не привязан к определенному рабочему месту и может выполнять работы в любом удобном месте. Таким образом, был выбран ноутбук со следующими характеристиками:

– процессор: Intel Core i7 4700MQ;

– видеокарта: Nvidia GeForce GTX 860M;

– частота процессора: 2400 МГц;

– оперативная память: 8 Гб;

– видеопамять: 2048 Мб;

Стоимость данного ноутбука составляет 40 тыс. руб.

Затраты на инструментальные программные средства для проектировщика в данном проекте рассчитываются из суммы затрат на программное обеспечение, их стоимость указана в таблице 3.

Таблица 3 – Расчет стоимости программного обеспечения

Наименование	Назначение	Стоимость программного обеспечения с учетом срока использования	Срок использования
Unity3d	Платформа для разработки приложений	Бесплатно	30 дней
Adobe Photoshop	Набор программного обеспечения для растровой графики	499 руб./месяц	30 дней
Microsoft Visual Studio 2019	Полнофункциональная интегрированная среда разработки для написания, отладки, тестирования и развертывания кода на любой платформе	бесплатно	45 дней
Windows 10 home	Набор программного обеспечения для работы человека с ЭВМ.	8499 руб.	бессрочно
Open Office	Для создания документации по проекту	бесплатно	10 дней

ОС Windows будет актуальна ещё два года, за это время создается примерно 4 проекта, следовательно, общая стоимость делится 4.

Получается, что $К_{ипс} = 499 + 8499/4 = 2624$ руб.

Однако, так как длительность проекта составляет 1.5 месяца, то стоимость вычислительной техники полностью записать в проект нельзя, нужно узнать срок службы комплектующих и рассчитать амортизацию оборудования на срок эксплуатации в проекте.

По статистике, средний срок службы компьютерных комплектующих составляет от 3 до 5 лет. Принимая в расчет то, что ноутбук обладает меньшей надежностью, чем персональный компьютер вследствие того, что является более сложным техническим устройством и, таким образом, более подвержен поломкам, то следует взять минимальный срок службы, то есть 3 года. Теперь рассчитаем амортизационную стоимость по формулам 4 и 5.

$$А_{год} = С_б * Н_{ам}, \quad (4)$$

где $А_{год}$ – амортизация за год использования;

$С_б$ – балансовая стоимость;

$Н_{ам}$ – норма амортизации.

$$А_{пр} = А_{год} / К_{рдгм} * К_{дэ}, \quad (5)$$

где $А_{пр}$ – проектная амортизация;

$К_{рдг}$ – количество рабочих дней в 2020 году;

$К_{дэ}$ – количество дней эксплуатации.

$А_{год} = 40\,000 * 0.33 \approx 13\,333$ руб.

$А_{пр} = \frac{13333}{247} * 45 = 2429$ руб.

Таким образом, $К_{свт} = 2429$ руб.

Рассчет заработной платы разработчика представлен в таблице 4.

Таблица 4 – Расчет заработной платы разработчика

Состав заработной платы	Сумма, руб
Оклад	20000
Районный коэффициент	6000
Северный коэффициент	6000
НДФЛ (13%)	4160
Отчисления во внебюджетные фонды (30,2%)	9664
Итого	41664

Значит, $К_{зп} = 41\,664$ руб.

В прочие затраты входят затраты на непредвиденные расходы. Принято, что на прочие затраты необходимо оставлять не меньше 3% от общих расходов на ПО и затраты на вычислительную технику. На прочие расходы был взят порог отчислений равный 10%. Значит, $К_{проч} = 505$ руб.

$К_{пр} = 2429 + 2624 + 41\,664 + 505 = 47\,222$ руб.

Состав затрат на разработку изображен на рисунке 31.

Рисунок 31 – Затраты на разработку

3.1.2 Эксплуатационные затраты

Рассчитаем эксплуатационные затраты при помощи формулы 6:

$$C = C_{зп} + C_{ао} + C_{то} + C_{лс} + C_{ни} + C_{проч}, \quad (6)$$

где $C_{зп}$ – зарплата персонала, работающего с информационной системой;

$C_{ао}$ – амортизационные отчисления;

$C_{то}$ – затраты на техническое обслуживание;

$C_{лс}$ – затраты на использование глобальных сетей;

$C_{ни}$ – затраты на носители информации;

$C_{проч}$ – прочие затраты.

$C_{зп} = 0$, так как не требуется персонал.

Затраты на обслуживание составляют 2500 руб. в месяц.

$C_{то} = 2500$ руб.

Затраты на использование глобальных сетей, то есть постоянное Интернет-соединение составляют 300 руб. в месяц.

$C_{лс} = 300$ руб.

$C_{ни} = 0$, так как носители информации не нужны.

Прочие эксплуатационные затраты составляю 10% от общей суммы предыдущих эксплуатационных затрат.

$C_{проч}$ = 280 руб.

C = 2500 + 300 + 280 = 3080 руб.

Состав эксплуатационных затрат изображен на рисунке 32.

Рисунок 32 – Эксплуатационные затраты

3.1.3 Прямые затраты

Прямые затраты рассчитываются по формуле 7:

$$DE=DE_1+DE_2+ DE_3 +DE_4+ DE_5+ DE_6+ DE_7+ DE_8, \quad (7)$$

где DE_1– капитальные затраты;

DE_2 – расходы на управление информационными технологиями;

DE_3 – расходы на техническую поддержку;

DE_4 – расходы на разработку прикладного внутреннего ПО внутренними силами;

DE_5 – расходы на аутсорсинг;

DE_6 – командировочные расходы;

DE_7 – расходы на услуги связи;

DE_8 – прочие расходы.

Таким образом, рассчитаем прямые затраты:

DE1 = К = 47 222 = 47 222 руб.

DE2 = $С_{зп}$ = 0 руб.

DE3 = $С_{то}$ = 2500 руб.

DE4 = 0 руб.

DE5 = $К_{оэ}$ = 1500 руб., так как часть тестирования может проводиться при помощи сторонних сил.

DE6 = 0 руб.

DE7 = $С_{лс}$ = 300 руб.

DE8 = $С_{проч}$ = 280 руб.

Итого прямые затраты по формуле:

DE = 47 222 + 2500 + 1500 + 300 + 280 = 51 802 руб.

Так как в данном проекте TCO ≈ DE, то TCO = 51 802 руб.

3.2 Определение экономической эффективности реализации проекта

3.2.1 Анализ рынка продуктов-аналогов

Рынок 2D-игр очень огромен и представляет собой многообразную смесь жанров, геймплея и сюжета на любой вкус. Двумерные игры не утратили популярность, и по сей день они создаются как одиночными разработчиками, так и игровыми студиями. В частности, известная игра Dead Cells, вышедшая в 2018 году от студии Motion Twin, представляет собой типичный 2D-платформер с элементами «roguelike» (все уровни процедурно-генерируемые, а при смерти персонажа игра начинается с самого начала) [16].

Однако, разработанный проект все же можно назвать уникальным в плане того, что его основными задачами являются привлечение абитуриентов при поступлении в институт на определенное направление, а таких решений на рынке представлено совсем немного [17]. Например, как уже было упомянуто в разделе 1, в Томском политехническом университете существует

несколько подобных игр, привлекающих абитуриентов и позволяющих получить дополнительные баллы. Тем не менее, схожим проектом является только интерактивный курс «Экспедиция», так как совпадают жанр и задача.

3.2.2 Экономическая эффективность реализации проекта

Данный проект не нацелен на получение прямого дохода, так как он является бесплатным, однако, экономическая эффективность будет выражаться в том, что реализация данного проекта поспособствует увеличению известности Хакасского технического института – филиала СФУ, а также позволит привлечь больше абитуриентов. Стоимость обучения одного студента составляет около 70 тыс. рублей в семестр [18]. В настоящий момент стоимость обучения студента на платной основе эквивалентна бюджетной. Таким образом, вне зависимости от того, будет студент обучаться на бюджетной основе или же на платной, средства на обучение будут идти в институт: в первом случае это будет субсидия на выполнение государственного задания, во втором – собственные средства обучающегося.

3.3 Оценка рисков реализации проекта

Данный проект подвержен малому числу рисков, но часть из них может оказать большое влияние на выполнение проекта. Риски проекта представлены в таблице 7.

Таблица 7 – Оценка рисков проекта

№	Описание риска	Вероятность возникновения риска	Степень воздействия риска на проект
1	Лимитированное время разработки	Низкая	Низкая
2	Невостребованность продукта	Низкая	Средняя-высокая
3	Отказ заказчика от проекта	Низкая	Низкая-критичная

Лимитированное время разработки. Данный риск имеет низкую вероятность возникновения, так как масштаб проекта невелик, следовательно, не должно возникнуть ситуаций, при которых завершение проекта может затянуться. Собственно, поэтому и степень воздействия данного риска мала. Для полного предотвращения данного риска необходимо грамотно распланировать время этапов разработки, а также контролировать выполнения этих этапов.

Невостребованность продукта. Данный риск имеет низкий шанс возникновения, так как, во-первых, целевая аудитория является абитуриентами, либо станут ими в будущем, как следствие, подобного рода контент будет им интересен, во-вторых, данный продукт почти не имеет аналогов, в-третьих, продукт доступен для общего пользования. Однако, если все же случится так, что продукт действительно окажется невостребованным и не заинтересует конечного пользователя в достаточной степени, то это будет критично для проекта. Для предотвращения такого развития событий можно принять ряд следующих мер: размещение на сайте заказчика ХТИ – филиала СФУ информации с подробным описанием готового приложения, его функций и назначения, а также наполнение игры контентом, который заинтересует пользователей и будет служить для продвижения игры. Вдобавок, на различных профориентационных меропиятиях, проводимых институтом, таких, как дни открытых дверей, фестивали, клубы и т.д., необходимо рассказывать о данной игре.

Отказ заказчика от проекта. Сложно спрогнозировать степень влияния данного риска, так как данный проект уникален в своем роде, а также является узконаправленным. Поэтому если произойдет отказ от проекта, то есть вероятность найти нового Заказчика, которого заинтересует данный проект (скорее всего, другое высшее учебное заведение), однако, неизвестно заинтересует ли этот проект кого-либо еще, кроме настоящего Заказчика. А это значит, что проект может остаться вообще нереализованным. Чтобы снизить степень воздействия данного риска на проект, следует принять

следующие меры: провести анализ рынка и выяснить, есть ли у других учебных заведений потребность в такого рода проектах.

3.4 Выводы по разделу «Расчёт затрат и оценка экономической эффективности реализации проекта «Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Таким образом, можно подвести итоги экономического раздела:

– получены следующие данные:

– капитальные затраты = 47 222 руб.

– эксплуатационный затраты = 3080 руб.

– общая стоимость владения, рассчитанная по методике ТСО = 51 802 руб.

– вероятность возникновения рисков в процессе реализации проекта низкая, тем не менее, были созданы варианты их предотвращения или устранения.

– экономическая эффективность проекта – косвенная, и выражается в привлечении абитуриентов на направление «Прикладная информатика» в ХТИ – филиал СФУ.

ЗАКЛЮЧЕНИЕ

Результатом данной выпускной квалификационной работы стала разработанная игра в жанре 2D-платформер для направления «Прикладная информатика». В частности, были выполнены следующие задачи:

1. Проанализированы имеющиеся практики для привлечения абитуриентов. ХТИ – филиал СФУ проводит профориентационные мероприятия и знакомство с направлениями подготовки для учащихся школ и абитуриентов, а также устраивает различные конкурсы и олимпиады.

2. Выбраны средства разработки программного продукта: игровой движок Unity, так как для разработки 2D-игр он является очень удобным из-за того, что обладает большим функционалом и поддержкой 2D-разработки, Microsoft Visual Studio 2017 для написания программного кода для игры на языке программирования C# и графический редактор Photoshop.

3. Разработан 2D-платформер, содержащий в себе элементы квеста. Игра имеет несколько уровней и большое количество разнообразных вопросов из области информационных технологий, на которые предстоит ответить игроку во время игры.

4. Были рассчитаны капитальные, эксплуатационные и прямые затраты.

5. Проведен расчет экономической эффективности проекта, которая будет выражена в увеличении популярности ХТИ – филиала СФУ, а также в привлечении абитуриентов на направление «Прикладная информатика».

В дальнейшем возможно усовершенствование проекта или создание на его основе аналогичного, который уже будет выполнять функцию привлечения абитуриентов на другие направления обучения в ХТИ – филиал СФУ. Также, если будет создана соответствующая юридическая основа, то можно начислять дополнительные баллы за прохождение данной игры при наборе определенного результата.

СПИСОК СОКРАЩЕНИЙ

ПО – Программное обеспечение.

ОС – Операционная система.

ХТИ – Хакасский технический институт.

СФУ – Сибирский федеральный университет.

ИТ – Информационные технологии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Naramura, Yuki (January 23, 2019). «Peak Video Game? Top Analyst Sees Industry Slumping in 2019». Bloomberg L.P. Retrieved January 29, 2019.

2. Бакалавриат – Прикладная информатика [Электронный ресурс]. – Режим доступа: http://khti.sfu-kras.ru/obuchenie/bakalavriat-prikladnaya-informatika.php.

3. СФУ – Рейтинг и учет индивидуальных достижений [Электронный ресурс]. – Режим доступа: http://admissions.sfu-kras.ru/rating.

4. The ESA – Computer video game report 2016 информатика [Электронный ресурс]. – Режим доступа: http://www.theesa.com/about-esa/essential-facts-computer-video-game-industry/Essential-Facts-2016.pdf.

5. Интерактивные обучающие игры // Интернет-лицей ТПУ [Электронный ресурс]. – Режим доступа: https://il.tpu.ru/game.

6. 2D vs 3D: Game Development [Электронный ресурс]. – Режим доступа: https://meliorgames.com/game-development/2d-vs-3d-games-differences-benefits-and-costs/

7. Pile Jr, John (May 2013). 2D Graphics Programming for Games. New York, NY: CRC Press. ISBN 978-1466501898.

8. How 3-D graphics works [Электронный ресурс]. – Режим доступа: https://computer.howstuffworks.com/3dgraphics.htm.

9. A Detailed Cross-Examination of Yesterday and Today's Best-Selling Platform Games [Электронный ресурс]. – Режим доступа: http://www.gamasutra.com/view/feature/1851/a_detailed_crossexamination_of_.php.

10. The Specusphere – Gamespeak: A glossary of Gaming Terms [Электронный ресурс]. – Режим доступа: http://www.specusphere.com/joomla/index.php?option=com_content&task=view&id=232&Itemid=32.

11. Цуканова, О. А. Методология и инструментарий моделирования бизнеспроцессов : учебное пособие / О. А. Цуканова. – СПб.: Университет ИТМО : 2015. – 100 с.

12. Хокинг, Д. Unity в действии. Мультиплатформенная разработка на C#: книга / Д. Хокинг. – СПБ: Manning, 2019. – 352 с.

13. Сайт разработчиков Unreal Engine [Электронный ресурс]. – Режим доступа: https://www.unrealengine.com/en-US/

14. X Free 2D Game Assets [Электронный ресурс]. – Режим доступа: https://bevouliin.com/category/free_game_asset/

15. Plaformer Tiles [Электронный ресурс]. – Режим доступа: https://opengameart.org/content/platformer-tiles

16. Сайт разработчиков Dead Cells [Электронный ресурс]. – Режим доступа: https://dead-cells.com/

17. Информация об олимпиадах [Электронный ресурс]. – Режим доступа: http://khti.sfu-kras.ru/postuplenie/olimpiady/olimpiady-dlya-starsheklassnikov.php

18. Стоимость обучения [Электронный ресурс]. – Режим доступа: http://khti.sfu-kras.ru/obuchenie/stoimost-obucheniya.php

19. ГОСТ Р ИСО 9001-2001 Системы менеджмента качества. Требования. Введ. 31.08.2001.

20. ГОСТ ISO/IEC/IEEE 29148:2018 Программная и системная инженерия. Процессы жизненного цикла. Разработка требований. Введ. 28.11.2018.

21. Выполнение и защита выпускной квалификационной работы по направлению 09.03.03 «Прикладная информатика» [Электронный ресурс] : метод. указания / сост. Е. Н. Скуратенко, В. И. Кокова, И. В. Янченко ; Сиб. федер. ун-т, ХТИ – филиал СФУ. – Абакан : ХТИ – филиал СФУ, 2017.

22. СТО 4.2–07–2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности – Введ. 02.07.2014. – Красноярск: ИПК СФУ, 2014. – 60 с.

ПРИЛОЖЕНИЕ А

Class Bullet

```
    public class Bullet : MonoBehaviour
{
    private GameObject parent;
    public GameObject Parent { set { parent = value; } get { return parent; } }

    private float speed = 10.0F;
    private Vector3 direction;
    public Vector3 Direction { set { direction = value; } }

    public Color Color
    {
        set { sprite.color = value; }
    }

    private SpriteRenderer sprite;

    private void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>();
    }

    private void Start()
    {
        Destroy(gameObject, 1.0F);
    }

    private void Update()
    {
        transform.position = Vector3.MoveTowards(transform.position, transform.position +
direction, speed * Time.deltaTime);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, 0.5F);

        if (colliders.Length > 0 && !collision.GetComponent<QuestTrigger>() &&
!collision.GetComponent<WinMenu>() && !collision.GetComponent<Heart>() &&
!collision.GetComponent<Coin>()) Destroy(gameObject);

    }
}
```

Class CameraController

```
public class CameraController : MonoBehaviour
{
    [SerializeField]
    private float speed = 2.0F;

    [SerializeField]
    private Transform target;

    private void Awake()
    {
        if (!target) target = FindObjectOfType<Character>().transform;
```

```
    }

    private void Update()
    {
        Vector3 position = target.position; position.y = 0; position.z = -10.0F;

        transform.position = Vector3.Lerp(transform.position, position, speed *
Time.deltaTime);
    }
}
```

Class Character

```
public class Character : Unit
{
    public DeathMenu deathScreen;
    public PauseMenu pauseMenu;
    public Vector3 savedPosition;
    private int k = 0;

    [SerializeField]
    private int lives = 5;

    public int Lives
    {
        get { return lives; }
        set
        {
            if (value <= 5) lives = value;
            livesBar.Refresh();
        }
    }
    private LivesBar livesBar;

    [SerializeField]
    private readonly float speed = 5.0F;
    [SerializeField]
    private readonly float jumpForce = 15.0F;


    private bool isGrounded = false;

    private Bullet bullet;
    public Vector3 direction;

    private CharState State
    {
        get { return (CharState)animator.GetInteger("State"); }
        set { animator.SetInteger("State", (int)value); }
    }

    new public Rigidbody2D rigidbody;
    private Animator animator;
    private SpriteRenderer sprite;

    private float firerate;

    private readonly float invincibilityDuration = 2.0F;
    private bool isInvincible = false;

    private Color Color
    {
        set { sprite.color = value; }
    }
```

```
    private void Awake()
    {
        livesBar = FindObjectOfType<LivesBar>();
        rigidbody = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        sprite = GetComponentInChildren<SpriteRenderer>();

        bullet = Resources.Load<Bullet>("Bullet");
    }

    private void FixedUpdate()
    {
        CheckGround();
        Deaths(); // проверка смерти
    }

    private void Update()
    {
        if (isGrounded) State = CharState.Idle;

        firerate += Time.deltaTime;
        if (Input.GetButtonDown("Fire1") && Time.timeScale != 0 && firerate >= 1)
        {
            Shoot();
            firerate = 0;
        }

        if (Input.GetButton("Horizontal")) Run();
        if (isGrounded && Input.GetButtonDown("Jump") && Time.timeScale != 0) Jump();
        if (Input.GetButtonDown("Cancel") && Time.timeScale != 0) Pause();
    }

    private void Run()
    {
        direction = transform.right * Input.GetAxis("Horizontal");

        transform.position = Vector3.MoveTowards(transform.position, transform.position +
direction, speed * Time.deltaTime);

        sprite.flipX = direction.x < 0.0F;

        if (isGrounded) State = CharState.Run;
    }

    private void Jump()
    {
        rigidbody.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
    }

    private void Shoot()
    {
        Vector3 position = transform.position; position.y += 0.75F; position.x += 0.8F *
Mathf.Sign(direction.x);
        Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as
Bullet;

        newBullet.Parent = gameObject;
        newBullet.Direction = newBullet.transform.right * (sprite.flipX ? -1.0F : 1.0F);
    }

    public override void ReceiveDamage()
    {
        if (!isInvincible)
        {
            StartCoroutine(BecomeInvincible());
```

```
            Lives--; //invincibilityDuration
            Toss();
        }
    }

    private void CheckGround()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, 0.3F);

        isGrounded = colliders.Length > 1;

        if (isGrounded && colliders.All(x =>
!x.GetComponent<HorizontalMovablePlatform>()) &&
            colliders.All(x => !x.GetComponent<VerticalMovablePlatform>()) &&
colliders.All(x => !x.GetComponent<FallBlock>())) k = 0;

        if (!isGrounded)
        {
            State = CharState.Jump;

            if (k == 0)
            {
                savedPosition = transform.position; savedPosition.x += 0.5F *
Mathf.Sign(direction.x);
                Debug.Log(savedPosition);
                k++;
            }
        }
    }

    public void Toss()
    {
        rigidbody.velocity = Vector3.zero;
        rigidbody.AddForce(transform.up * 5.0F, ForceMode2D.Impulse);
        rigidbody.AddForce(-direction * 3.5F, ForceMode2D.Impulse);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, 0.3F);

        if (collision.GetComponent<Coin>() || collision.GetComponent<Heart>() ||
collision.GetComponent<Monster>() ||
            collision.GetComponent<ShootableMonster>() ||
collision.GetComponent<MoveableMonster>() ||
            collision.GetComponent<QuestTrigger>() || collision.GetComponent<WinMenu>()
|| collision.GetComponent<Spike>() ||
            collision.GetComponent<SpikeFall>() || collision.GetComponent<Bullet>() ||
            collision.GetComponent<HorizontalMovablePlatform>()) k++;

        Bullet bullet = collision.gameObject.GetComponent<Bullet>();
        if (bullet && bullet.Parent != gameObject)
        {
            ReceiveDamage();
        }
    }

    public void Deaths()
    {
        if (Lives < 1)
        {
            deathScreen.gameObject.SetActive(true);
            Character ch = gameObject.GetComponent<Character>();
            ch.gameObject.SetActive(false);
        }
```

```
    }

    private void Pause()
    {
        Time.timeScale = 0F;
        pauseMenu.gameObject.SetActive(true);
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.GetComponent<HorizontalMovablePlatform>() ||
            collision.gameObject.GetComponent<VerticalMovablePlatform>())
this.transform.parent = collision.transform;
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.gameObject.GetComponent<HorizontalMovablePlatform>() ||
            collision.gameObject.GetComponent<VerticalMovablePlatform>())
this.transform.parent = null;
    }

    private IEnumerator BecomeInvincible()
    {
        isInvincible = true;
        Color = Color.gray;
        yield return new WaitForSeconds(invincibilityDuration);
        Color = Color.white;
        isInvincible = false;
    }
}

public enum CharState
{
    Idle,
    Run,
    Jump,
    Death
}
```

Class Coin

```
public class Coin : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Character character = collision.GetComponent<Character>();

        if (character)
        {
            FindObjectOfType<ScoreBar>().Score += 5;
            Destroy(gameObject);
        }
    }
   }
```

Class DeathMenu

```
public class DeathMenu : MonoBehaviour
{
    public DeathMenu deathScreen;
    public MainMenu mainMenu;

    public void RestartGame()
    {
```

```
        SceneManager.LoadScene("SampleScene");
        Time.timeScale = 1F;
    }

    public void QuitToMainMenu()
    {
        SceneManager.LoadScene("SampleScene");
        Time.timeScale = 0F;
    }
    }
```

Class DisableBlock

```
ublic class DisableBlock : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        if (unit && unit is Character)
        {
            Destroy(gameObject);
        }
    }
}
```

Class FallBlock

```
public class FallBlock : MonoBehaviour
{
    private Vector3 direction;

    private float delay = 0;

    void Update()
    {
        Dead();
    }

    private void Dead()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * 1.0F, 0.05F);

        if (colliders.Length > 0 && colliders.All(y => y.GetComponent<Character>()))
        {
            delay += Time.deltaTime;
            if (delay >= 0.3) Destroy(gameObject);
        }
    }
}
```

Class Heart

```
public class Heart : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Character character = collision.GetComponent<Character>();

        if (character)
        {
            character.Lives++;
            Destroy(gameObject);
        }
```

```
    }
    }
```

Class HorizontalMovablePlatform

```
public class HorizontalMovablePlatform : MonoBehaviour
{
    [SerializeField]
    private float speed = 3.0F;

    private Vector3 direction;

    private SpriteRenderer sprite;

    void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>();
    }

    void Start()
    {
        direction = -transform.right;
    }

    void Update()
    {
        Move();
    }

    private void Move()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.right * direction.x * 2.0F, 0.1F);

        if (colliders.Length > 0 && colliders.All(x => !x.GetComponent<Character>()) &&
colliders.All(x => !x.GetComponent<Bullet>()) && colliders.All(x =>
!x.GetComponent<QuestTrigger>())) direction *= -1.0F; // поворот при столкновении
        transform.position = Vector3.MoveTowards(transform.position, transform.position +
direction, speed * Time.deltaTime); // движение
    }
}
```

Class InstructionMenu

```
public class InstructionMenu : MonoBehaviour
{
    public InstructionMenu instructionMenu;
    public MainMenu mainMenu;
    public Text text;

    public void NextScreen()
    {
        instructionMenu.gameObject.SetActive(false);
        Time.timeScale = 1F;
    }
}
```

Class LivesBar

```
public class LivesBar : MonoBehaviour
{
    const int lives = 5;

    private Transform[] hp = new Transform[lives];

    private Character character;
```

```
    private void Awake()
    {
        character = FindObjectOfType<Character>();

        for (int i = 0; i < hp.Length; i++)
        {
            hp[i] = transform.GetChild(i);
        }
    }

    public void Refresh()
    {
        for (int i = 0; i < hp.Length; i++)
        {
            if (i < character.Lives) hp[i].gameObject.SetActive(true);
            else hp[i].gameObject.SetActive(false);
        }
    }
}
```

Class MainMenu

```
public class MainMenu : MonoBehaviour
{
    public MainMenu mainMenu;
    public InstructionMenu instructionMenu;

    private void Start()
    {
        Time.timeScale = 0F;
    }

    public void Play()
    {
        instructionMenu.gameObject.SetActive(true);
        mainMenu.gameObject.SetActive(false);
    }

    public void Exit()
    {
        Application.Quit();
    }
}
```

Class Monster

```
public class Monster : Unit
{
    protected virtual void Awake() { }
    protected virtual void Start() { }
    protected virtual void Update() { }


    protected virtual void OnTriggerEnter2D(Collider2D collision)
    {
        Bullet bullet = collision.GetComponent<Bullet>();
        Unit unit = collision.GetComponent<Unit>();

        if (bullet)
        {
            ReceiveDamage();
            FindObjectOfType<ScoreBar>().Score += 5;
        }
```

```
        if (unit && unit is Character)
        {
            if (Mathf.Abs(unit.transform.position.x - transform.position.x) < 0.7F)
            {
                ReceiveDamage();
                FindObjectOfType<ScoreBar>().Score += 10;
            }
            else unit.ReceiveDamage();
        }
    }
}
```

Class MoveableMonster

```
public class MoveableMonster : Monster
{
    [SerializeField]
    private float speed = 3.0F;

    private Vector3 direction;

    private SpriteRenderer sprite;

    protected override void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>();
    }

    protected override void Start()
    {
        direction = -transform.right;
    }

    protected override void Update()
    {
        Move();
    }

    protected override void OnTriggerEnter2D(Collider2D collision)
    {
        Bullet bullet = collision.GetComponent<Bullet>();
        Unit unit = collision.GetComponent<Unit>();

        if (bullet)
        {
            ReceiveDamage();
            FindObjectOfType<ScoreBar>().Score += 5;
        }

        if (unit && unit is Character)
        {
            if (Mathf.Abs(unit.transform.position.x - transform.position.x) < 0.7F)
            {
                ReceiveDamage();
                FindObjectOfType<ScoreBar>().Score += 15;
            }
            else unit.ReceiveDamage();
        }
    }

    private void Move()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * 0.5F + transform.right * direction.x * 0.5F, 0.05F);
```

```
        if (colliders.Length > 0 && colliders.All(x => !x.GetComponent<Character>()) &&
colliders.All(x => !x.GetComponent<Bullet>()) && colliders.All(x =>
!x.GetComponent<QuestTrigger>())) direction *= -1.0F;
        transform.position = Vector3.MoveTowards(transform.position, transform.position +
direction, speed * Time.deltaTime);

        sprite.flipX = direction.x < 0.0F; // поворот спрайта при столкновении
    }
}
```

Class PauseMenu

```
public class PauseMenu : MonoBehaviour
{
    public PauseMenu pauseMenu;
    public MainMenu mainMenu;

    public void ResumeGame()
    {
        Time.timeScale = 1F;
        pauseMenu.gameObject.SetActive(false);
    }

    public void QuitToMainMenu()
    {
        SceneManager.LoadScene("SampleScene");
        Time.timeScale = 0F;
    }

    private void Update()
    {
        if (Input.GetButtonDown ("Cancel") && (Time.timeScale == 0)) ResumeGame();
    }
}
```

Class QuestMenu

```
public class QuestMenu : MonoBehaviour
{
    private Character character;
    //борд
    private QuestMenu questBoard;
    public Text qText;
    public Text[] answerText;
    public Text result;

    //вопросы и ответы
    public QuestionList[] questions;
    QuestionList currentQ;
    List<object> qList;
    int randQ;
    private int questCount = 4;
    private int rightCount = 0;
    private int wrongCount = 0;

    void Start()
    {
        questBoard = gameObject.GetComponent<QuestMenu>();
        character = FindObjectOfType<Character>();
        qList = new List<object>(questions);
        QuestionGenerate();
        Pause();
    }

    public void AnswerButton(int index)
```

```
    {
        if (answerText[index].text.ToString() == currentQ.answers[0])
        {
            result.text = "Верно!";
            rightCount++;
            FindObjectOfType<ScoreBar>().Score += 50;
        }
        else
        {
            result.text = "Неверно!";
            wrongCount++;
        }
        if (rightCount == 5)
        {
            character.Lives++;
            rightCount = 0;
        }
        if (wrongCount == 3)
        {
            character.Lives--;
            wrongCount = 0;
        }
        qList.RemoveAt(randQ);
        QuestionGenerate();
    }

    void QuestionGenerate()
    {
        if (questCount > 0)
        {
            randQ = Random.Range(0, qList.Count);
            currentQ = qList[randQ] as QuestionList;
            qText.text = currentQ.question;
            questCount--;
            List<string> answers = new List<string>(currentQ.answers);
            for (int i = 0; i < currentQ.answers.Length; i++)
            {
                int rand = Random.Range(0, answers.Count);
                answerText[i].text = answers[rand];
                answers.RemoveAt(rand);
            }
        }
        else
        {
            Resume();
            questCount = 4;
            QuestionGenerate();
        }
}

    public void Pause()
    {
        Time.timeScale = 0F;
    }

    public void Resume()
    {
        Time.timeScale = 1F;
        questBoard.gameObject.SetActive(false);
    }
}

[System.Serializable]
public class QuestionList
{
```

```
    public string question;
    public string[] answers = new string[4];
}
```

Class QuestTrigger

```
public class QuestTrigger : MonoBehaviour
{
    public QuestMenu questBoard;
    private QuestTrigger qTrigger;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        if (unit && unit is Character)
        {
            questBoard.gameObject.SetActive(true);
            qTrigger = gameObject.GetComponent<QuestTrigger>();
            qTrigger.gameObject.SetActive(false);
            Time.timeScale = 0F;
        }
    }
}
```

Class Restart

```
public class Restart : MonoBehaviour
{
    public void RestartGame()
    {
        SceneManager.LoadScene("SampleScene");
    }
}
```

Class ScoreBar

```
public class ScoreBar : MonoBehaviour
{
    public Text Number;
    public int Score { set; get; }

    void Update()
    {
        Number.text = Score.ToString();
    }
}
```

Class ShootableMonster

```
public class ShootableMonster : Monster
{
    [SerializeField]
    private float rate = 2.0F;
    [SerializeField]
    private Color bulletColor = Color.white;

    private Bullet bullet;

    protected override void Awake()
    {
        bullet = Resources.Load<Bullet>("Bullet");
    }

    protected override void Start()
    {
```

```
        InvokeRepeating("Shoot", 0.2F * rate, rate);
    }

    private void Shoot()
    {
        Vector3 position = transform.position; position.y += 0.5F; position.x += -0.8F;
        Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as
Bullet;

        newBullet.Parent = gameObject;
        newBullet.Direction = -newBullet.transform.right;
        newBullet.Color = bulletColor;
    }

    protected override void OnTriggerEnter2D(Collider2D collision)
    {
        Bullet bullet = collision.GetComponent<Bullet>();
        Unit unit = collision.GetComponent<Unit>();

        if (bullet)
        {
            ReceiveDamage();
            FindObjectOfType<ScoreBar>().Score += 5;
        }

        if (unit && unit is Character)
        {
            if (Mathf.Abs(unit.transform.position.x - transform.position.x) < 0.65F)
            {
                ReceiveDamage();
                FindObjectOfType<ScoreBar>().Score += 20;
            }
            else unit.ReceiveDamage();
        }
    }
}
```

Class SpikeFall

```
public class SpikeFall : Spike
{
    private Character character;

    private void Start()
    {
        character = FindObjectOfType<Character>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        if (unit && character)
        {
            character.transform.position = character.savedPosition - 0.8F *
character.direction;
            character.ReceiveDamage();
        }
    }
}
```

Class VerticalMovablePlatform

```
public class VerticalMovablePlatform : MonoBehaviour
{
```

```
    [SerializeField]
    private float speed = 3.0F;

    private Vector3 direction;

    private SpriteRenderer sprite;

    void Start()
    {
        direction = transform.up;
    }

    void Update()
    {
        Move();
    }

    private void Move()
    {
        if (transform.position.y > 3.5f)
            direction *= -1.0F;

        if (transform.position.y < -2f)
            direction *= -1.0F;

        transform.position = Vector3.MoveTowards(transform.position, transform.position +
direction, speed * Time.deltaTime);
    }
}
```

Class VerticalMovingMonster

```
public class VerticalMovingMonster : Monster
{
    [SerializeField]
    private float speed = 3.0F;

    private Vector3 direction;

    private SpriteRenderer sprite;

    protected override void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>();
    }

    protected override void Start()
    {
        direction = transform.up;
    }

    protected override void Update()
    {
        Move();
    }

    protected override void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        if (unit && unit is Character)
        {
            unit.ReceiveDamage();
        }
    }
```

```
    private void Move()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position +
transform.up * direction.y * 0.5F, 0.05F);

        if (colliders.Length > 0 && colliders.All(y => !y.GetComponent<Character>()) &&
            colliders.All(y => !y.GetComponent<Bullet>()) && colliders.All(y =>
!y.GetComponent<Coin>())) direction *= -1.0F;
        transform.position = Vector3.MoveTowards(transform.position, transform.position +
direction, speed * Time.deltaTime);

        sprite.flipY = -direction.y < 0.0F;
    }
}
```

Class WinMenu

```
public class WinMenu : MonoBehaviour
{
    public DeathMenu winMenu;
    public Text comment;
    private int localResult = 0;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Unit unit = collision.GetComponent<Unit>();

        if (unit && unit is Character)
        {
            localResult = FindObjectOfType<ScoreBar>().Score;
            comment.text = ($"Ваш счет: {localResult}\n");

            if (localResult < 800)
                comment.text += "Попробуйте еще раз.";
            else if (localResult > 800 && localResult < 1200)
                comment.text += "Нужно постараться!";
            else if (localResult > 1200 && localResult < 1600)
                comment.text += "Неплохо!";
            else if (localResult > 1600 && localResult < 2000)
                comment.text += "Хорошо!";
            else if (localResult > 2000 && localResult < 2500)
                comment.text += "Отлично! Вы - молодец.";
            else if (localResult > 2500)
                comment.text += "Легендарно! Мое почтение!";

            winMenu.gameObject.SetActive(true);
            Time.timeScale = 0F;
        }
    }
}
```

Выпускная квалификационная работа выполнена мной самостоятельно. Использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

Отпечатано в одном экземпляре.

Библиография 22 наименования.

Экземпляр сдан на кафедру.

«____» _____ 2020 г.

_____ Бобонаков Константин Владимирович
подпись

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Хакасский технический институт – филиал ФГАОУ ВО
«Сибирский федеральный университет»

Кафедра прикладной информатики, математики и естественно-научных дисциплин

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Е. Н. Скуратенко
«16» 06 2020 г.

БАКАЛАВРСКАЯ РАБОТА
09.03.03 Прикладная информатика

Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Руководитель _____ ст. преподаватель В. И. Кокова

Выпускник _____ К. В. Бобошилов

Консультанты
по разделам:

Экономический _____ Е. Н. Скуратенко

Нормоконтролер _____ В. И. Кокова

Абакан 2020

Студенту Бобошкову Константину Владимировичу

Группа ХБ 16-03

Направление 09.03.03 Прикладная информатика

Тема выпускной квалификационной работы: Разработка игры в жанре 2D-платформер для направления «Прикладная информатика»

Утверждена приказом по университету № 216 от 06.04.2020 г.

Руководитель ВКР: В. И. Кокова, ст. преподаватель, ХТИ – филиал СФУ

Исходные данные для ВКР: заказ ХТИ – филиала СФУ.

Перечень разделов ВКР:

1. Анализ предметной области. Выбор средств проектных решений.

2. Описание разработки игры.

3. Расчёт затрат и оценка экономической эффективности реализации проекта «Разработка игры в жанре 2D-платформер для направления «Прикладная информатика».

Перечень графического материала: нет

Руководитель ВКР _____ В. И. Кокова

Задание принял к исполнению _____ К. В. Бобошков

«06» апреля 2020 г.