МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет вычислительной техники

Кафедра МО и ПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Мультиплатформенное игровое приложение-стратегия по управлению нефтехранилищем

Студент _____ Ликсин Сергей Сергеевич

(подпись, дата) (ФИО полностью)

Руководитель _____ Шибанов С. В.

(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В

(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Макарычев П.П.

(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.

(подпись) (фамилия, инициалы)

Пенза, 2022

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет — Вычислительной техники

Кафедра — МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
«__» _____2022 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Ликсин Сергей Сергеевич гр. 18ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Мультиплатформенное игровое приложение-стратегия по управлению нефтехранилищем

Тема утверждена приказом ПГУ №327/О от "18" апреля 2022 г.

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Шибанов С. В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: игровое приложение способствующее развитию навыков импровизации, логического мышления, долгосрочного планирования и управления некоторой ситуацией

2) Среда выполнения: операционная система Windows 10

Языковые средства разработки: C#, PHP, SQL, UML.

Инструментальные средства: Visual Studio 2022, PostgreSQL 12, DataGrip, Draw.io, Unity 3d, Visual Studio Code, Visual Paradigm CE 16.3

Технологии разработки ООП, RAP – модель, компонентно – ориентированный подход

3) Основные функции приложения:

а) управление резервуаром;

б) отображение изменений цен на нефть;

в) отображение текущей суммы на счету нефтехранилища;

г) возможность улучшения резервуара;

д) возможность расширения нефтехранилища;

е) отображение списка резервуаров;

ж) возможность инвестирования в побочные здания

5. Перечень подлежащих разработке вопросов:

1)	*Анализ предметной области и постановка задачи;*
2)	*Проектирование;*
3)	*Реализация;*
4)	*Контроль качества программного обеспечения;*
5)	*План разработки бюджета;*
6)	*Оформление пояснительной записки.*

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10%	13.04.2022 – 27.04.2022	
2	Проектирование	25%	27.04.2022 – 10.05.2022	
3	Реализация	35%	10.05.2022 – 30.06.2022	
4	Контроль качества программного обеспечения	10%	31.05.2022 – 02.06.2022	
5	План разработки бюджета	5%	02.06.2022 – 03.06.2022	
6	Оформление пояснительной записки	15%	03.06.2022 – 17.06.2022	

Дата выдачи задания «22» ____декабря____2021 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

РЕФЕРАТ

Пояснительная записка к выпускной квалификационной работе содержит 104 страницы, 36 рисунков, 14 таблиц, 17 источников, 3 приложения.

НЕФТЕХРАНИЛИЩЕ, МУЛЬТПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ, МОДЕЛИРОВАНИЕ ПРОЦЕССА РАБОТЫ НЕФТЕХРАНИЛИЩА, ТРЕХУРОВНЕВАЯ АРХИТЕКТУРА.

Объект разработки – мультиплатформенное игровое приложение-стратегия по управлению нефтехранилищем.

Цель работы – разработка мультиплатформенного игрового приложения-стратегии, моделирующего работу нефтехранилища и предоставляющего пользователю всю необходимую информацию и все средства для управления работой предприятия.

В ходе выполнения выпускной квалификационной работы была исследована предметная область, разработана база данных и запросы к ней, разработан пользовательский интерфейс.

В результате было разработано игровое приложение-стратегия по управлению нефтехранилища Разработанное приложение отображает информацию о ценах на нефть, о бюджете предприятия и о состоянии нефтехранилища.

Содержание

Введение ... 8
1 Анализ предметной области и постановка задачи на разработку приложения ... 10
1.1 Анализ предметной области и сценарий работы приложения ... 10
1.1.1 Анализ предметной области игрового приложения ... 10
1.1.2 Анализ сценария работы игрового приложения ... 11
1.2 Актуальность разработки и сравнительный анализ приложений-аналогов ... 12
1.2.1 Актуальность разработки игрового приложения ... 12
1.2.2 Цели и задачи сравнительного анализа аналогов ... 13
1.2.3 Выбор анализируемых приложений-аналогов в жанре стратегических Idle-игр ... 13
1.2.4 Обзор приложений-аналогов в жанре стратегических Idle-игр ... 14
1.2.5 Результаты сравнительного анализа приложений-аналогов в жанре стратегических Idle-игр ... 19
1.3 Постановка задачи на разработку приложения ... 21
1.3.1 Цели разработки и назначение приложения ... 21
1.3.2 Классы пользователей приложения ... 22
1.3.3 Перечень основных функций приложения ... 22
1.4 Планирование и функционально-стоимостный анализ процесса разработки приложения ... 23
2 Анализ требований на разработку программного обеспечения ... 27
2.1 Анализ функциональных требований ... 27
2.2 Требования к концептуальной модели ... 31
2.3 Требования к обобщенной архитектуре приложения ... 36
2.4 Обоснование выбор технологий и средств разработки ... 45
2.4.1 Среда разработки игровых приложений Unity 3D ... 45
2.4.2 Сценарный язык программирования C# ... 46
2.4.3 Среда разработки программного обеспечения Visual Studio 2022 ... 47

2.5 Требования к программно-аппаратному обеспечению 48
2.5.1 Требования к аппаратно-техническому обеспечению 49
2.5.2 Требования к системному программному обеспечению 49
2.5.3 Требования к прикладному обеспечению 49
3 Разработка информационного программного обеспечения приложения 51
3.1 Разработка архитектуры развертывания информационного и программного обеспечения приложения 51
3.2 Разработка базы данных 51
3.2.1 Логическое моделирование базы данных 52
3.2.2 Физическое моделирование базы данных 54
3.2.3 Развертывание базы данных средствами СУБД PostgreSQL 54
3.3 Разработка серверного программного обеспечения 56
3.4 Разработка клиентского программного обеспечения 58
3.4.1 Разработка пользовательского интерфейса клиентского приложения ... 58
3.4.2 Разработка программного обеспечения клиентского приложения 62
4 Контроль качества программного обеспечения приложения 65
4.1 Планирование процесса контроля качества программного обеспечения . 65
4.1.1 Мероприятия по контролю качества программного обеспечения 65
4.1.2 Обоснование выбора модели тестирования 65
4.1.3 Обоснование выбора метрик программного кода 66
4.2 Тестирование программного обеспечения 66
4.2.1 Функциональное тестирование 66
4.2.2 Модульное тестирование 67
4.3 Оценка качества программного кода 68
4.3.1 Оценка качества программного кода средствами среды разработки Visual Studio 2022 68
4.3.2 Метрика Чепина 70
Заключение 72
Список использованных источников 73
Приложение А. Глоссарий 75

Приложение Б. Код программы 80
Приложение В. Результаты тестирования 98

Введение

Деятельность – специфическая человеческая активность, регулируемая сознанием, порождаемая потребностями и направленная на познание и преобразование внешнего мира и самого человека. Различают три основных вида деятельности: игру, учение и труд. Труд – это деятельность, целью которой является производство общественно необходимых продуктов. Деятельность человека, имеющая своей целью приобретение знаний, умений и навыков, называется учением. Целью игры является сама осуществляемая «деятельность», а не ее результаты [1].

Игры полезны независимо от возраста. Голландский историк Йохан Хейзинга (1872 – 1945) когда-то назвал игры отличительной особенностью нашего вида. Для него человечество отличается не как Homo Sapiens, «человек разумный», а как Homo ludens, «человек играющий» [2]. Игровая деятельность, в том или ином виде, влияет на состояние играющего. Чаще всего в качестве положительных аспектов выделяют то, что:

– игры снимают стресс, позволяя выплеснуть широкий спектр эмоций;

– игры создают состояние гипер-креативности (повышенного творческого потенциала);

– игры помогают людям социализироваться.

«Всё есть яд, и ничто не лишено ядовитости; одна лишь доза делает яд незаметным». Игра – неотъемлемая часть жизни человека, в том или ином виде «играет» каждый. У детей игры, зачастую, не несут какой-либо практической ценности. В более сознательном возрасте человек добавляет в игровую деятельность компоненты обучения, либо даже труда. Важно контролировать количество времени потраченного на игровую деятельность и не пренебрегать своими обязанностями.

Современные игры отличаются своим разнообразием. Однако человек постоянно ищет новые для себя способы проведения досуга, новые испытания и загадки.

Основной проблемой современных игровых приложений является однотипность ключевых механик. Зачастую смена видеоигры не приносит пользователю необходимых эмоций из-за неизменности действий, которые должен выполнять пользователь. Также ключевым вопросом современной игровой индустрии является доступность приложений в различных жизненных ситуаций. С каждым годом количество людей играющих со смартфона или планшета увеличивается, следовательно, мультиплатформенность игровых приложений становится все более необходимой.

Исходя из вышеперечисленных фактов, было решено, в рамках выпускной квалификационной работы, разработать мультиплатформенное игровое приложение-стратегию по управлению нефтехранилищем.

1 Анализ предметной области и постановка задачи на разработку приложения

1.1 Анализ предметной области и сценарий работы приложения

1.1.1 Анализ предметной области игрового приложения

Игровая индустрия насчитывает десятки игровых жанров. Одним из самых популярных жанров является жанр «Стратегии». В таких проектах игровая механика зачастую построена на умении логически мыслить и просчитывать ситуацию на пару ходов вперед.

Idle-игры (они же «кликеры» или инкрементальные игры) – это игры (зачастую в жанре стратегия), целиком посвященные управлению потоками доходов. Как и симуляторы, они фокусируются на принятии решений, которые увеличивают доход [3].

Хотя в жанре и есть место различным вариантам и экспериментам, можно выделить ключевые фундаментальные аспекты дизайна подобных игр [3]:

– наличие как минимум одной валюты или числа;

– это число растёт с определённой скоростью и не требует вмешательства игрока, либо требует незначительных действий с его стороны;

– скорость, с которой число увеличивается, можно увеличить, чтобы число увеличивалось быстрее.

Именно основной цикл, состоящий из накопления ресурсов, их постоянных затрат и ускорения дохода, определяет жанр и выделяет его среди игр, которые целиком рассчитаны на набор очков.

Нефтехранилище (нефтебаза, нефтетерминал) – промышленный объект для хранения нефти и нефтепродуктов. Нефтехранилище – это сложный составной объект, требующий тщательного контроля и своевременной реакции на изменения в его работе.

В состав данного промышленного объекта входит множество емкостей для хранения нефти и нефтепродуктов, называемых резервуарами. Основной

задачей резервуаров является сохранение нефти в изначальном состоянии, при должных температурах и недопущение падения качества нефтепродуктов.

Также важной частью каждого резервуара в нефтехранилище являются насосы. Насосы необходимы для транспортировки нефти в резервуары и из резервуаров в фуры для дальнейшей продажи нефти. Насосы в нефтехранилищах бывают двух видов: насос закачки и насос откачки. Первый, как понятно из названия, полностью работает на закачку нефти в свой резервуар. Аналогично второй позволяет выкачивать нефть из резервуара для дальнейшей транспортировки.

Как правило, управление небольшим нефтехранилищем, в один или два резервуара, не кажется сложной задачей. Однако, имеющее небольшую прибыль нефтехранилище, не может позволить себе дальнейшее развитие и автоматизацию процесса, без своевременно получаемой информации о состоянии резервуаров, о цене не нефтепродукты.

В случае же, с большими нефтехранилищами также необходима более слаженная работа, своевременное получение информации и возможность мгновенной реакции на изменение цены на рынке нефтепродуктов. В случае несоблюдения этих правил, нефтехранилище будет терять огромную часть потенциальной прибыли, что замедлит развитие промышленного объекта.

1.1.2 Анализ сценария работы игрового приложения

Основной сценарий работы приложения – моделирование процесса работы нефтехранилища, а также экономики предприятия. Пользователь управляет нефтехранилищем, в которое входит множество резервуаров: распоряжается бюджетом, следит за ценами, контролирует работу насосов, а также занимается расширением мощностей предприятия.

Цена на нефтепродукты периодически меняется то в большую, то в меньшую сторону. Задача игрока отслеживать изменение цен на

нефтепродукты, закупать по более низкой цене, а продавать по более высокой.

Заработанные средства пользователь вправе потратить на расширение мощностей нефтехранилища: расширение допустимого объема хранящихся нефтепродуктов в резервуаре, покупка более мощных насосов, а также найма менеджера для резервуара.

Менеджеры предназначены для автоматизации работы резервуара (работы без непосредственного контроля со стороны игрока). Они обладают теми же возможностями, что и пользователь, в рамках своего резервуара. Единственным исключением является отсутствие у них доступа к улучшению резервуара и его насосов.

В качестве основной цели перед игроком ставится расширение своего предприятия, увеличение прибыльности нефтехранилища, а также открытие собственных малых бизнесов, инвестиции в инфраструктуру, образование и городские службы.

1.2 Актуальность разработки и сравнительный анализ приложений-аналогов

1.2.1 Актуальность разработки игрового приложения

Если приложению присваивается жанр «Стратегия», значит пользователю предстоит управлять какими-то ресурсами. Возможны постановки задач осуществить которые предлагается путем смекалки, хитрости и навыков логически мыслить. Наибольшей популярности достигли игры военного и экономического плана [4].

Социальные исследования, проводимые ведущими институтами мира, публикуют оптимистичные выводы. Человек, увлекающийся играми стратегической механики, развивает в себе целый набор полезных качеств [5]. Пользователь учится:

– логически мыслить;

– контролировать изменения в обозначенном пространстве;

– планировать свои действия;

– приспосабливаться к любым ситуациям.

Важной особенностью является то, что никаких возрастных ограничений к играм-стратегиям не существует. По-настоящему хорошая игра поможет развить навыки стремления к достижению поставленных целей в повседневных делах [4].

1.2.2 Цели и задачи сравнительного анализа аналогов

Сравнительный анализ аналогов позволяет выделить важные особенности, преимущества и недостатки конкурентов. Это позволит избежать распространенных ошибок в разработке игрового приложения, а также поможет установить необходимые для успешной реализации приложения условия. Можно выделить следующие цели проведения анализа систем:

– выявление основных преимуществ у приложений-конкурентов;

– выявление основных недостатков приложений-конкурентов;

– выявление необходимых базовых функций приложений конкурентов.

Результатом анализа будет список обязательных к реализации функций, а также возможные уникальные механики, которые помогут улучшить игровой опыт для пользователя, что приведет к повышению конкурентоспособности разрабатываемого игрового приложения.

1.2.3 Выбор анализируемых приложений-аналогов в жанре стратегических Idle-игр

Idle-игры крайне популярны в 2022 году, следовательно, для анализа необходимо выбрать самые известные и уникальные проекты, имеющие с разрабатываемым приложением схожую концепцию.

В качестве приложений-аналогов для сравнительного анализа были выбраны следующие игровые приложения:

– Cookie Clicker – аркадный кликер в жанре «стратегия»;

– Годвилль – браузерная Idle-игра для «нуля игроков» в жанре ролевая игра;

– Adventure Capitalist – кликер в жанре «стратегия», предлагающий игроку управлять бизнесом;

– Clicker Heroes – кликер в жанре «стратегия» с некоторыми элементами ролевой игры, предлагающий игроку управлять группой героев, сражающихся с монстрами;

– Creature Card Idle – гибрид Idle-игры и всего лучшего из карточных игр.

Данные игровые приложения распространяются как по системе Free-to-Play (FTP – условно бесплатная модель), так и по системе Pay-To-Play (PTP – необходимо купить игру). Также у некоторых приложений присутствует социальная составляющая (различные топы игроков, дуэли игроков и т.д.). Эта информация подробно представлена в таблице 1.

Таблица 1 – Информация об игровых приложениях

Название приложения	Модель распространения	Наличие социальной составляющей	Фирма производитель
Cookie Clicker	FTP	-	Julien Thiennot, DashNet
Годвилль	FTP	+	Godville Games Limited
Adventure Capitalist	FTP	-	Hyper Hippo Entertainment
Clicker Heroes	FTP	+	Playsaurus
Creature Card Idle	PTP	-	Fordesoft

Таким образом, для сравнительного анализа приложений – конкурентов были выбраны самые популярные и уникальные проекты игрового жанра.

1.2.4 Обзор приложений-аналогов в жанре стратегических Idle-игр

Cookie Clicker – это игра, разработанная Ортейлом на JavaScript, и впервые увидевшая свет 10 августа 2013 года. Это так называемая «игра на

увеличение». Смысл игры – печь печеньки, кликая мышью на большую печеньку, пока не наберется достаточное количество печенек для покупки строений и улучшений. Затем снова копить печеньки, и покупать больше строений и улучшений для производства большего количества печенек, и снова копить печеньки на покупку очередных строений и улучшений. По сути, игра бесконечна, и в нее можно играть до тех пор, пока не надоест. Для разнообразия геймплея в игре присутствуют достижения, а также множество забавных и интересных моментов. Основные элементы Idle-игр присутствуют, сложность игры невысока, проиграть игрок не может. Интерфейс прост, но приятен глазу. Основное меню игры представлено на рисунке 1.

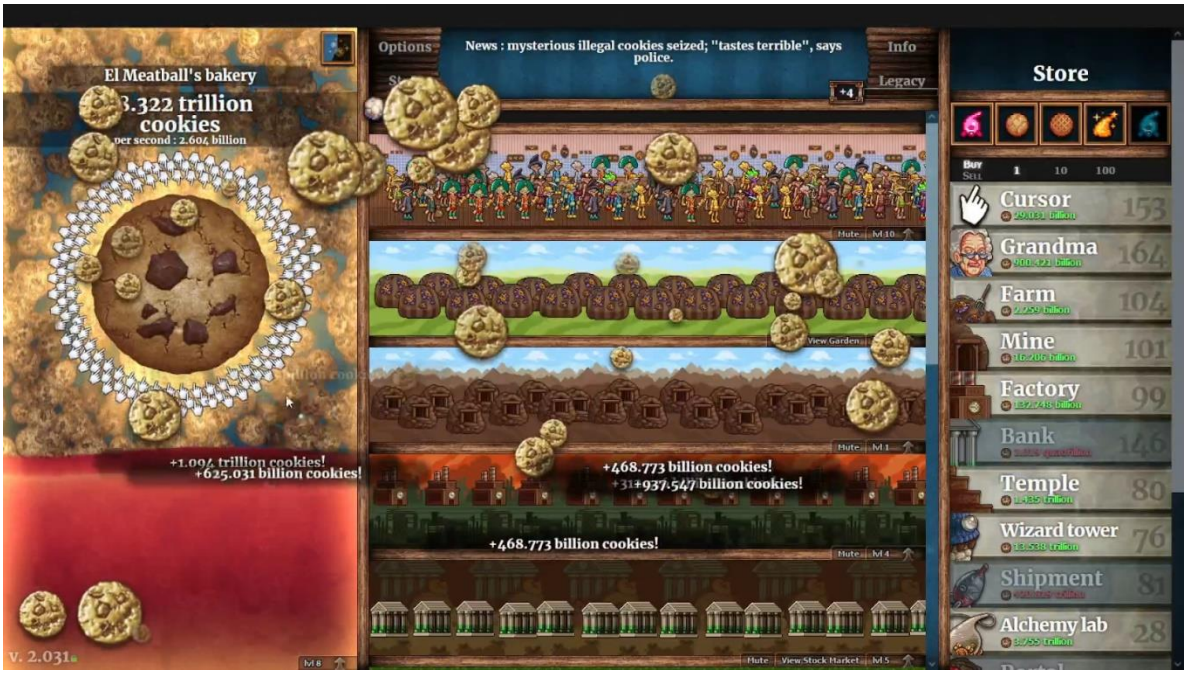

Рисунок 1 – Cookie Clicker

Игра и по сей день поддерживается и обновляется.

Годвилль – компьютерная ролевая игра для «нуля игроков», доступная на мобильных устройствах и в большинстве браузеров. Игра разрабатывается Михаилом Платовым и Дмитрием Косиновым. Игрок управляет персонажем, именуемым богом или богиней. Игроку предлагается примерить на себя роль бога и обзавестись своего рода апостолом – героем, который будет

совершенно самостоятельно развиваться, брать и выполнять задания, заводить питомцев, строить храм своему богу и заниматься прочими делами, которые игрок с удовольствием или без оного выполняет в RPG.

Данная игра относится к играм с «нулем игроков». Влияние игрока на происходящее незначительно, либо очень ограничено. Валюта присутствует, но пользователь не может ей распоряжаться – это за него делает программа. Интерфейс не отличается особой проработанностью, большая часть информации представлена в виде текста. Так как влияние игрока на происходящее в игре незначительно, какую-либо стратегию разработать не получится. Однако стоит отметить уникальный опыт, который получает игрок в процессе погружения в данную игру. Разработчики не поскупились на обильный юмор и отсылки, сама игра постоянно преподносит игроку странные и смешные истории и задания для вашего «героя». Одной из значительных особенностей этой игры является то, что игроку даже не обязательно тратить много времени на игру, игра продолжается и без его участия. Главная страница игры представлена на рисунке 2.

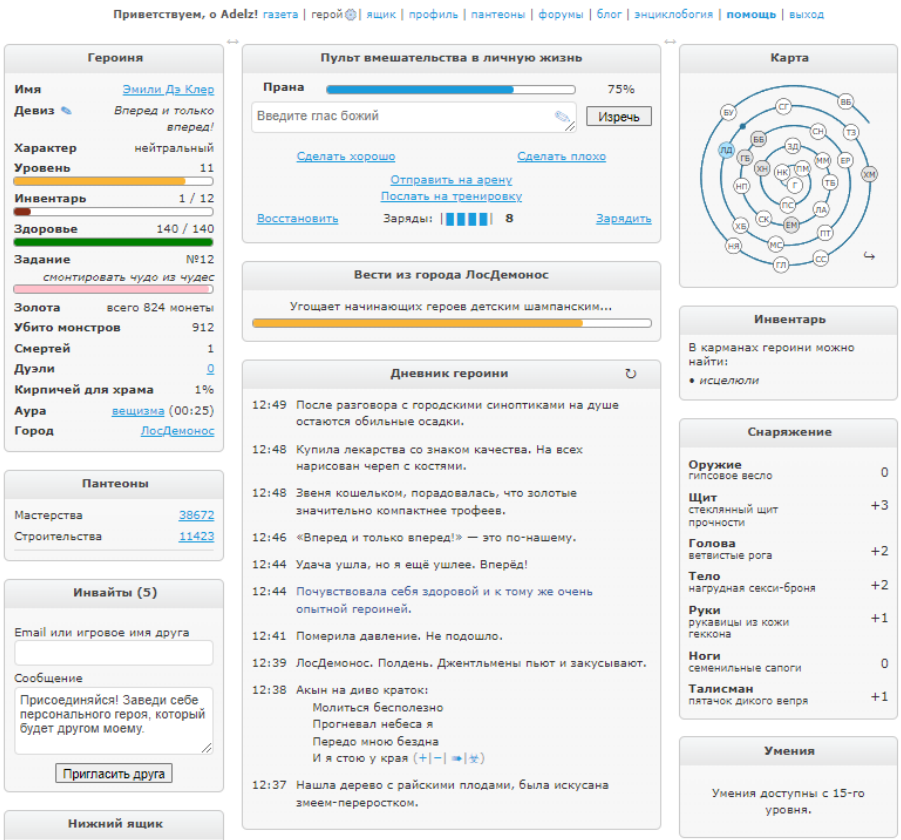

Рисунок 2 – Годвилль

AdVenture Capitalist является кликером, где игрок начинает свой путь с маленькой компании, а затем развивает ее до состояния огромной корпорации. Более того, когда ваша корпорация станет самой богатой на планете, у вас появится возможность развивать межгалактический бизнес. В игре присутствует валюта, красочный интерфейс и увлекающий геймплей. Пользователь непосредственно влияет на то, как происходит игра, без него ничего развиваться не будет. Автоматизация процесса присутствует, но происходит постепенно, поэтому участие игрока обязательно. Однако сложность стратегии в данной игре оставляет желать лучшего. Проиграть игрок не может, любые его действия только увеличивают прибыль и открывают все новые и новые возможности. Главное меню игры представлено на рисунке 3.

Рисунок 3 – Adventure Capitalist

Clicker Heroes – условно бесплатный аркадный кликер с ролевыми элементами в фэнтезийном сеттинге. Как и во многих играх такого типа, нужно много тыкать пальцем или покупать помощников, которые будут тыкать за вас. Кроме этого вы будете собирать золото, выбирать снаряжение и развивать навыки вашего персонажа. Присутствуют ключевые элементы любой Idle-игры: валюта, способы ее зарабатывать и увеличивать доходность происходящего. Также присутствует красочный интерфейс. Стратегия игры так же, как и в случае с Adventure Capitalist, проста и не дает игроку возможности окончательно проиграть. Несмотря на это игра требует от пользователя принятия каких-либо решений для продвижения выше по уровням. Основное окно игры представлено на рисунке 4.

Рисунок 4 – Clicker Heroes

Creature Card Idle – это игра в жанре Idle-игр, карточная и стратегия, разрабатываемая Fordesoft для платформы PC. На рисунке 5 представлено основное окно из игры.

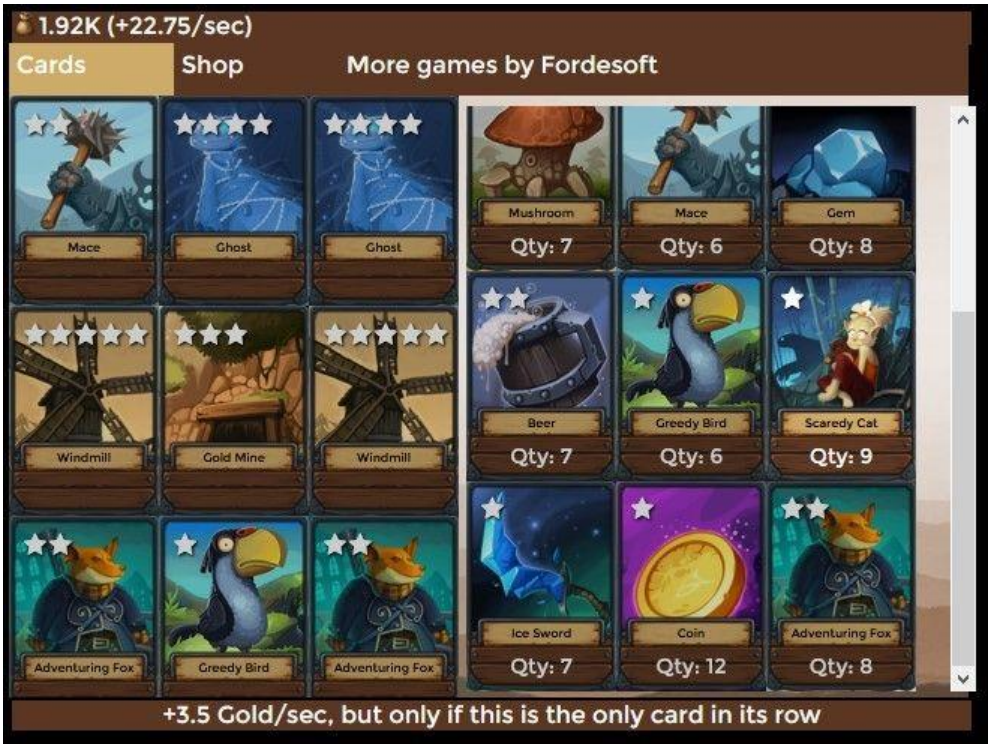

Рисунок 5 – Creature Card Idle

Окружение в игре относится к стилистике фэнтези. Стильный, хоть и простой, интерфейс, валюта, которую можно потратить на покупку карт. Многие основные особенности жанра в игре присутствуют. Механики игры усложняют процесс получения валюты: кроме простой покупки улучшений, необходимо их правильно расположить на игровом поле для более эффективной добычи.

1.2.5 Результаты сравнительного анализа приложений-аналогов в жанре стратегических Idle-игр

Для оценки, анализа и сравнения игровых приложений – аналогов были выделены критерии сравнения:

– наличие валюты;

– влияние игрока на экономику игры;

– интересный UX опыт, получаемый игроком;

– красочный UI – привлекающий внимание пользовательский интерфейс;

– сложность разрабатываемой стратегии игры.

Для критериев описанных выше используется трехбалльная шкала соответствия, от 0 – «полное несоответствие», до 2 – «полное соответствие требованиям». В таблице 2 приведены характеристики сравнения.

Таблица 2 – Шкала баллов

Балл	Характеристика сравнения
0	приложение не выполняет условия соответствия критерию
1	приложение частично выполняет условия соответствия критерию
2	приложение выполняет все условия соответствия критерию

В таблице 3 приведены результаты сравнения аналогов по критериальным оценкам.

Таблица 3 – Сравнение аналогов по критериальным оценкам

Критерии	Cookie Clicker	Годвилль	Adventure Capitalist	Clicker Heroes	Creature Card Idle	Разрабатываемая система
Наличие валюты	1	0	2	1	2	1
Влияние игрока на экономику игры	1	0	1	1	2	2
Интересный UX	0	2	1	1	1	2
Красочный UI	1	0	2	1	2	2
Сложность разрабатываемой стратегии игры	0	0	0	0	1	2

Из таблицы видно, что не все потребности пользователя удовлетворены. Так, Cookie Clicker набрал 3 балла, Годвилль – 2 балла, Adventure Capitalist – 5 баллов, Clicker Heroes – 4 балла, а Creature Card Idle – 8 баллов. Планируется, что разрабатываемое приложение будет в большей степени удовлетворять потребности пользователя, набирая – 9 баллов. В ходе анализа аналогов было принято решение о создании собственного игрового приложения, отвечающего большинству вышеперечисленных требований.

1.3 Постановка задачи на разработку приложения

1.3.1 Цели разработки и назначение приложения

Целью разработки служит разработанное мультиплатформенное игровое приложение-стратегия в жанре Idle-игр. Данное приложение должно обладать всеми фундаментальными аспектами дизайна инкрементальных игр, обладать удобным интерфейсом и моделировать процесс управления нефтехранилищем. Также необходимо предоставить пользователю возможность запускать игровое приложение на таких популярных операционных системах, как Windows и Android.

Основным назначением приложения можно назвать обучение простейшим экономическим законам посредством игры. Пользователю необходимо следить за изменением цены на условный товар (нефть), закупать товар после снижения цены и продавать его после ее повышения. Прибыль пользователь получает за счет разницы между стоимостью покупки и стоимостью продажи товара. На накоплении валюты игра не заканчивается. Пользователю предоставляется возможность расширения и улучшения своего нефтехранилища, такое как:

– покупка нового резервуара;

– увеличение объема уже имеющегося резервуара;

– улучшение насосов резервуара (как насоса закачки, так и насоса откачки);

– найм менеджеров для автоматического управления резервуаром;

– получение кратковременных бонусов, покупаемых за игровую валюту (так называемые побочные постройки).

Таким образом игроку предоставляется возможность поучаствовать в моделировании игровой ситуации, в которой он – директор нефтехранилища, а его задача – построить успешное предприятие.

1.3.2 Классы пользователей приложения

«Игрок» – основной и единственный класс пользователей игрового приложения по управлению нефтехранилищем. Его основные характеристики приведены в таблице 4.

Таблица 4 – Классы и характеристики пользователя

Класс пользователей	Характеристика пользователей
Игрок	Игрок имеет право просмотра игровых сцен, взаимодействия с игровыми объектами, а также возможность просмотра цен на нефтепродукты и результатов своих игровых действий.

«Игрок» – основной фактор, влияющий на изменение игровой ситуации. Без участия пользователя, по крайней мере, на старте новой игры, нефтехранилище не сможет функционировать.

1.3.3 Перечень основных функций приложения

Основная задача мультиплатформенного игрового приложения-стратегии по управлению нефтехранилищем состоит в том, чтобы предоставить основным ее пользователям удобный и доступный инструментарий для организации деятельности нефтехранилища и его резервуаров в частности.

Также приложение должно наглядно иллюстрировать состояние резервуара, количество игровой валюты в «бюджете» нефтехранилища, а также предоставлять исчерпывающую информацию о цене на нефтепродукты.

Таким образом, разрабатываемое приложение должно решать следующие задачи:

– отображение изменений цен на нефть;

– управление насосами резервуара;

– отображение текущей суммы на счету нефтехранилища;

– предоставлять возможность улучшения резервуара: увеличение доступного для хранения объема;

– возможность расширять нефтехранилище (покупать новый резервуар);

– возможность выбирать активный (управляемый игроком в текущий момент) резервуар;

– возможность инвестирования в рекламу (далее побочные здания).

Таким образом, были выделены основные функции разрабатываемого мультиплатформенного игрового приложения по управлению нефтехранилищем.

1.4 Планирование и функционально-стоимостный анализ процесса разработки приложения

В рамках планирования процесса разработки игрового приложения необходимо сформировать план выполнения работ. План включает в себя порядок действий, необходимых для завершения приложения, а также необходимые для этого ресурсы. В качестве основной причины планирования выступает определение промежуточных целей разработки и методов для достижения этих целей. Диаграмма Ганта отлично подходит для наглядной демонстрации плана разработки. На данной диаграмме отображены:

– длительность разработки;

– сроки выполнения этапов разработки;

– необходимые для реализации каждого этапа ресурсы.

Для представления планирования была выбрана программа Microsoft Project, как современный инструмент для планирования разработки. Процесс планирования процесса разработки включает в себя:

– выделение основных задач и подзадач;

– расчет длительности выполнения каждой задачи;

– список ресурсов, необходимых для выполнения задачи.

На рисунке 6 представлен заполненный лист ресурсов.

		Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на использ.	Начисление	Базовый календарь
1		Программист ПС	Трудовой		П		100%	180,00 ₽/ч	300,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный
2		Программист приложени	Трудовой		П		100%	180,00 ₽/ч	300,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный
3		Тестировщик	Трудовой		Т		100%	120,00 ₽/ч	200,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный
4		Аналитик	Трудовой		А		100%	80,00 ₽/ч	150,00 ₽/ч	0,00 ₽	Пропорциональное	Стандартный
5		Компьютер	Материальный		К			30 000,00 ₽		0,00 ₽	Пропорциональное	
6		Интернет	Затраты		И						Пропорциональное	
7		Печать	Затраты		П						Пропорциональное	

Рисунок 6 – Заполненный лист ресурсов

Для общего представления всех необходимых данных об основных задачах, подзадачах, а также о длительности и о сроках их выполнения в Microsoft Project реализована специальная таблица. Данная таблица представлена на рисунке 7.

		Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1		Начало проекта	1 день	Ср 13.04.22	Ср 13.04.22		
2		**Анализ предметной области и постановка задачи на разработку приложен**	**7 дней**	**Чт 14.04.22**	**Пт 22.04.22**	**1**	
3		Анализ предметной области и сценарии работы приложения	2 дней	Чт 14.04.22	Пт 15.04.22	1	Интернет[500,00 ₽];Ан
4		Актуальность разработки и сравнительный анализ приложений	2 дней	Чт 14.04.22	Пт 15.04.22	1	Аналитик[50%]
5		**Постановка задачи на разработку приложения**	**5 дней**	**Пн 18.04.22**	**Пт 22.04.22**	**4**	
6		Цели разработки и назначение приложения	2 дней	Пн 18.04.22	Вт 19.04.22	4	Программист ПС;Прогр
7		Классы пользователей приложения	1 день	Ср 20.04.22	Ср 20.04.22	6	Программист приложе
8		Перечень основных функций приложения	2 дней	Чт 21.04.22	Пт 22.04.22	7	Программист приложе
9		**Планирование и функционально – стоимостный анализ процесса разрабо**	**6 дней**	**Пн 25.04.22**	**Пн 02.05.22**	**8**	
10		Основы функционально – стоимостного анализа	2 дней	Пн 25.04.22	Вт 26.04.22	8	Аналитик
11		Планирование процесса разработки	2 дней	Ср 27.04.22	Чт 28.04.22	10	Программист ПС;Прогр
12		Анализ временных и финансовых затрат	2 дней	Пт 29.04.22	Пн 02.05.22	11	Аналитик
13		**Разработка требований к приложению**	**9 дней**	**Вт 03.05.22**	**Пт 13.05.22**	**12**	
14		Функциональные требования	2 дней	Вт 03.05.22	Ср 04.05.22	12	Программист приложе
15		Требования к концептуальной модели	1 день	Чт 05.05.22	Чт 05.05.22	14	Программист ПС
16		Требования к обобщенной архитектуре приложения	2 дней	Пт 06.05.22	Пн 09.05.22	15	Интернет[500,00 ₽]
17		Обоснование выбора технологий и средств разработки	1 день	Вт 10.05.22	Вт 10.05.22	16	Программист приложе
18		**Требования к аппаратно-программному обеспечению**	**3 дней**	**Ср 11.05.22**	**Пт 13.05.22**	**17**	
19		Требования к аппаратно-техническому обеспечению	3 дней	Ср 11.05.22	Пт 13.05.22	17	Программист ПС[50%]
20		Требования к системному программному обеспечению	2 дней	Ср 11.05.22	Чт 12.05.22	17	Программист ПС[50%]
21		Требования к прикладному программному обеспечению	2 дней	Ср 11.05.22	Чт 12.05.22	17	Программист приложе
22		**Разработка информационного программного обеспечения приложения**	**15 дней**	**Пт 13.05.22**	**Чт 02.06.22**	**21**	
23		Разработка архитектуры развертывания информационного и программного	7 дней	Пт 13.05.22	Пн 23.05.22	21	Программист ПС[50%];
24		Разработка баз данных	8 дней	Вт 24.05.22	Чт 02.06.22	23	Программист ПС[50%]
25		Разработка серверного программного обеспечения	8 дней	Вт 24.05.22	Чт 02.06.22	23	Интернет[500,00 ₽];Пр
26		Разработка клиентского программного обеспечения	8 дней	Вт 24.05.22	Чт 02.06.22	23	Программист приложе
27		**Контроль качества программного обеспечения приложения**	**7 дней**	**Пт 03.06.22**	**Пн 13.06.22**	**26**	
28		**Планирование процесса контроля качества программного обеспечен**	**3 дней**	**Пт 03.06.22**	**Вт 07.06.22**	**26**	
29		Мероприятия по контролю качества программного обеспечения	1 день	Пт 03.06.22	Пт 03.06.22	26	Аналитик;Программис
30		Обоснование выбора модели тестирования	1 день	Пн 06.06.22	Пн 06.06.22	29	Тестировщик
31		Обоснование выбора метрик программного кода	1 день	Вт 07.06.22	Вт 07.06.22	30	Аналитик
32		Тестирование программного обеспечения	2 дней	Ср 08.06.22	Чт 09.06.22	31	Тестировщик
33		Оценка качества программного кода	2 дней	Пт 10.06.22	Пн 13.06.22	32	Аналитик
34		Оформление документации	2 дней	Вт 14.06.22	Чт 16.06.22	33	Интернет[500,00 ₽];Ан
35		Сдача проекта	1 день	Пт 17.06.22	Пт 17.06.22	34	
36		Проект завершен	0 дней	Пт 17.06.22	Пт 17.06.22	35	

Рисунок 7 – Перечень задач и ресурсов проекта

После заполнения таблицы с перечнем задач и ресурсов проекта Microsoft Project создает графическую иллюстрацию плана проекта в виде диаграммы Ганта.

На рисунке 8 представлена диаграмма Ганта для разработанных задач.

Рисунок 8 – Диаграмма Ганта

Также в Microsoft Project присутствует функция автоматического формирования отчетов. Один из распространенных отчетов, формируемых программой отчет «Бюджет».

Данный отчет отображает задачи проекта в порядке убывания по количеству потраченных денежных средств. На рисунке 9 представлен подобный отчет, сформированный исходя из нашей таблицы с перечнем задач и ресурсов проекта.

Отчет о бюджете от Ср 08.06.22
Бюджет.mpp

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
3	Анализ предметной области и сцена	0,00 ₽	Пропорциональное	31 140,00 ₽	0,00 ₽	31 140,00 ₽
23	Разработка архитектуры развертыва	0,00 ₽	Пропорциональное	15 120,00 ₽	0,00 ₽	15 120,00 ₽
26	Разработка клиентского программно	0,00 ₽	Пропорциональное	11 520,00 ₽	0,00 ₽	11 520,00 ₽
34	Оформление документации	0,00 ₽	Пропорциональное	9 960,00 ₽	0,00 ₽	9 960,00 ₽
6	Цели разработки и назначение прил	0,00 ₽	Пропорциональное	7 040,00 ₽	0,00 ₽	7 040,00 ₽
25	Разработка серверного программно	0,00 ₽	Пропорциональное	6 260,00 ₽	0,00 ₽	6 260,00 ₽
11	Планирование процесса разработки	0,00 ₽	Пропорциональное	5 760,00 ₽	0,00 ₽	5 760,00 ₽
14	Функциональные требования	0,00 ₽	Пропорциональное	5 760,00 ₽	0,00 ₽	5 760,00 ₽
24	Разработка баз данных	0,00 ₽	Пропорциональное	5 760,00 ₽	0,00 ₽	5 760,00 ₽
8	Перечень основных функций прило	0,00 ₽	Пропорциональное	4 160,00 ₽	0,00 ₽	4 160,00 ₽
29	Мероприятия по контролю качества	0,00 ₽	Пропорциональное	3 520,00 ₽	0,00 ₽	3 520,00 ₽
21	Требования к прикладному програм	0,00 ₽	Пропорциональное	2 880,00 ₽	0,00 ₽	2 880,00 ₽
19	Требования к аппаратно-техническо	0,00 ₽	Пропорциональное	2 160,00 ₽	0,00 ₽	2 160,00 ₽
7	Классы пользователей приложения	0,00 ₽	Пропорциональное	2 080,00 ₽	0,00 ₽	2 080,00 ₽
32	Тестирование программного обеспе	0,00 ₽	Пропорциональное	1 920,00 ₽	0,00 ₽	1 920,00 ₽
15	Требования к концептуальной моде	0,00 ₽	Пропорциональное	1 440,00 ₽	0,00 ₽	1 440,00 ₽
17	Обоснование выбора технологий и с	0,00 ₽	Пропорциональное	1 440,00 ₽	0,00 ₽	1 440,00 ₽
20	Требования к системному программ	0,00 ₽	Пропорциональное	1 440,00 ₽	0,00 ₽	1 440,00 ₽
10	Основы функционально – стоимостн	0,00 ₽	Пропорциональное	1 280,00 ₽	0,00 ₽	1 280,00 ₽
12	Анализ временных и финансовых за	0,00 ₽	Пропорциональное	1 280,00 ₽	0,00 ₽	1 280,00 ₽
33	Оценка качества программного кода	0,00 ₽	Пропорциональное	1 280,00 ₽	0,00 ₽	1 280,00 ₽
30	Обоснование выбора модели тестир	0,00 ₽	Пропорциональное	960,00 ₽	0,00 ₽	960,00 ₽
4	Актуальность разработки и сравните	0,00 ₽	Пропорциональное	640,00 ₽	0,00 ₽	640,00 ₽
31	Обоснование выбора метрик програ	0,00 ₽	Пропорциональное	640,00 ₽	0,00 ₽	640,00 ₽
16	Требования к обобщенной архитекту	0,00 ₽	Пропорциональное	500,00 ₽	0,00 ₽	500,00 ₽
1	Начало проекта	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽
35	Сдача проекта	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽
36	Проект завершен	0,00 ₽	Пропорциональное	0,00 ₽	0,00 ₽	0,00 ₽
		0,00 ₽		125 940,00 ₽	0,00 ₽	125 940,00 ₽

Рисунок 9 – Отчет по бюджету

Отчет «Сводка по проекту» показывает прогресс и общие сведения по проекту.

На рисунке 10 отображена сводка по разрабатываемому проекту.

Даты

Начало:	Ср 13.04.22	Окончание:	Пт 17.06.22
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней

Длительность

Текущий план:	48 дней	Оставшаяся:	48 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	48 дней	Процент завершени	0%

Трудозатраты

Текущий план:	608 ч	Оставшиеся:	608 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	608 ч	Процент завершени	0%

Затраты

Текущий план:	125 940,00 ₽	Оставшиеся:	125 940,00 ₽
Базовый план:	0,00 ₽	Фактические:	0,00 ₽
Отклонение:	125 940,00 ₽		

Состояние задачи

Еще не начатые задачи	36
Выполняющиеся задачи	0
Завершенные задачи:	0
Итого задач:	36

Состояние ресурса

Трудовые ресурсы:	6
Трудовые ресурсы с прев. дост	0
Материальные ресурсы:	1
Итого ресурсов:	7

Рисунок 10 – Сводка по проекту

Сформированный отчет о бюджете показал, что общая стоимость разработки мультиплатформенного игрового приложения по управлению нефтехранилищем составила 125 940 рублей. Исходя из сводки по проекту, следует, что в ходе реализации проекта было выполнено 36 задач. Проект был выполнен на 100% за 48 дней.

2 Анализ требований на разработку программного обеспечения

Требование – это условие или возможность, которой должна соответствовать система. По сути, требования – это точное и подробное описание того, что должно предоставлять разрабатываемое программное обеспечение (ПО), а также ограничения, которые приложение должно соблюдать. Чаще всего выделяют два основных типа требований к информационным системам (ИС):

– функциональные требования;

– нефункциональные требования.

Еще до разработки ПО необходимо обозначить значительную часть требований.

2.1 Анализ функциональных требований

Мультиплатформенное игровое приложение управления нефтехранилища должно позволять взаимодействовать с соответствующими сущностями информационной системы нефтехранилища. Исходя из перечня основных функций игрового приложения, были составлены информационные потребности пользователя приложения, которые представлены в виде диаграммы вариантов использования на рисунке 11.

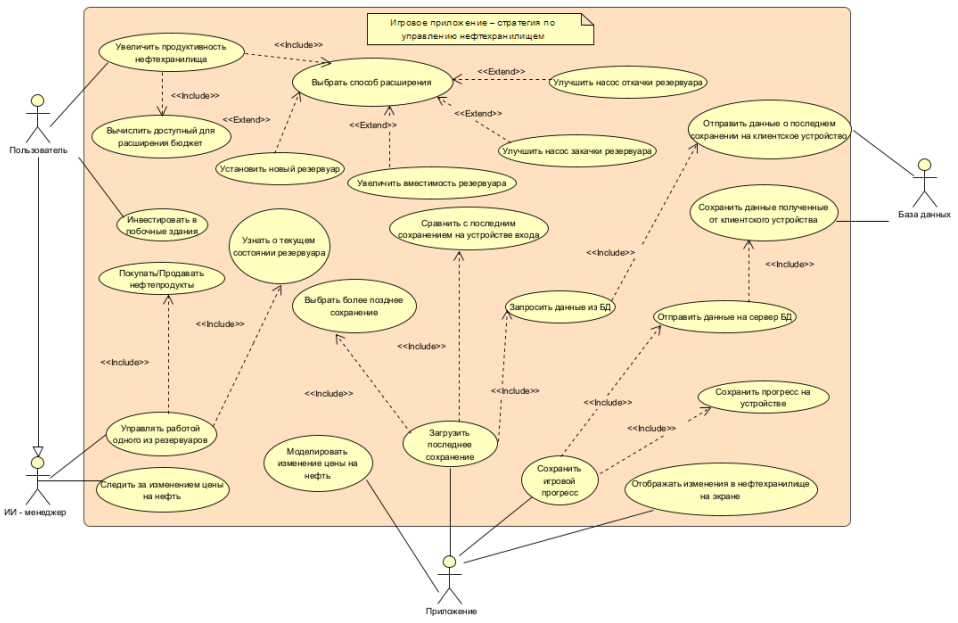

Рисунок 11 – Диаграмма вариантов использования

Была составлена спецификация для основных прецедентов. В таблице 5 приведено описание основного порядка выполнения пользователем варианта использования «Установить новый резервуар».

Таблица 5 – Спецификация прецедента «Установить новый резервуар»

Прецедент: Установить новый резервуар
ID: 1
Краткое описание: для увеличения объема хранимой нефти можно в нефтехранилище добавлять новый резервуар
Главные актёры: Пользователь.
Основной поток: а) Пользователь открывает список резервуаров б) Пользователь нажимает кнопку «Купить новый резервуар». в) Если у пользователя достаточно средств, то 1) Система списывает со счета нефтехранилища средства, создает новый резервуар и добавляет его в нефтехранилище 2) Система обновляет данные на экране, добавляя в список резервуаров созданный г) Иначе система выдает уведомление о том, что средств недостаточно

Управление работой резервуарами является одним из самых основных действий выполняемых пользователем. Именно это продвигает развитие нефтехранилища вперед.

В таблице 6 приведено описание основного порядка выполнения пользователем варианта использования «Управлять работой одного из резервуаров».

Таблица 6 – Спецификация прецедента «Управлять работой одного из резервуаров»

Прецедент: Управлять работой одного из резервуаров
ID: 2
Краткое описание: Для получения прибыли пользователь может покупать и закачивать нефть в резервуар, а потом выкачивать и продавать нефть за более высокую цену
Главные актёры: Пользователь.

Продолжение таблицы 6

Основной поток: а) Пользователь отслеживает цены на нефть б) Если цена ниже средней по рынку 1) Пользователь включает насос закачки, тем самым покупая нефть и закачивая ее в резервуар 2) Пользователь отключает насос откачки в) Если цена выше средней по рынку 1) Пользователь включает насос откачки, выкачивая нефть из резервуара и продавая ее 2) Пользователь отключает насос закачки г) Система отображает изменения на экране (изменение средств на счете нефтехранилища за счет покупки/продажи нефти)

В таблице 7 приведено описание основного порядка выполнения пользователем варианта использования «Увеличить вместимость резервуара».

Таблица 7 – Спецификация прецедента «Увеличить вместимость резервуара»

Прецедент: Увеличить вместимость резервуара
ID: 3
Краткое описание: Чтобы увеличить объем нефтепродуктов, хранимых в нефтехранилища, можно расширить уже существующий резервуар
Главные актёры: Пользователь.
Предусловия: пользователь имеет достаточно средств на счету нефтехранилища
Основной поток: а) Пользователь открывает меню улучшений резервуара б) Пользователь нажимает на кнопку улучшения в) Система изменяет данные о резервуаре г) Система отображает на экране изменившийся объем резервуара

Диаграмма деятельности – один из способов наглядного представления поведения прецедентов, поэтому был сформирован сценарий выполнения основных информационных процессов (рисунок 12).

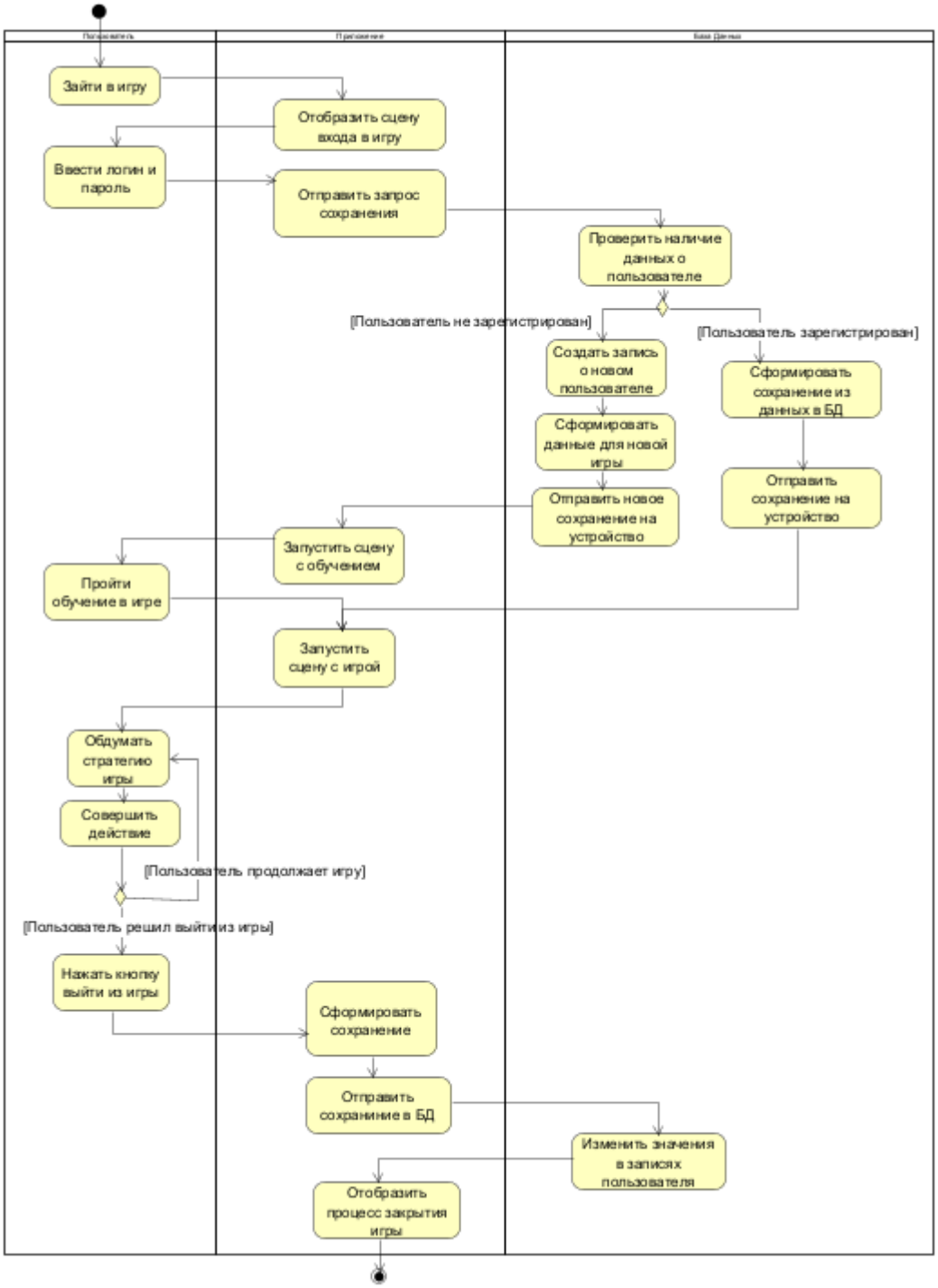

Рисунок 12 – Диаграмма деятельности

Разрабатываемое приложение должно обеспечивать пользователей следующими функциями:

– покупка/продажа нефтепродуктов;

– отображение изменяющейся цены на нефть;

– отображение состояния активного резервуара;

– возможность выбора активного резервуара;

– возможность покупки улучшений для активного резервуара;

– возможность тратить валюту на покупку и улучшение побочных зданий;

– отображение бюджета нефтехранилища;

– возможность покупки нового резервуара.

Исходя из вариантов использования, были сформированы функциональные требования с мультиплатформенному игровому – приложению по управлению нефтехранилищем.

2.2 Требования к концептуальной модели

Концептуальная модель представляет собой описание основных сущностей (таблиц) и связей между ними без учета принятой модели баз данных (БД) и синтаксиса целевой системы управления базами данных (СУБД).

Нефтехранилище является ключевым элементом предметной области. Оно состоит из резервуаров и управляется либо директором, либо менеджерами. У каждого резервуара присутствует 2 насоса, один из которых закачивает нефть в емкость, а другой выкачивает.

В таблице 8 представлены сущности, сформированные на основании входных и выходных данных бизнес-процессов.

Таблица 8 – Описание сущностей предметной области нефтехранилища.

№	Наименование	Описание	Количество экземпляров	Режим обновления
1	Нефтехранилище	Основной комплекс, включающий в себя множество емкостей для хранения нефтепродуктов	1 на каждого директора (игрока)	Каждый день
2	Резервуар	Основная емкость для хранения нефтепродуктов.	1 – 10 на каждое нефтехранилище	Каждый день

Продолжение таблицы 8

№	Наименование	Описание	Количество экземпляров	Режим обновления
3	Насос	Устройство для перекачки нефтепродуктов	По 2 на каждый резервуар	Каждый день
4	Менеджер резервуара	Управляющий одним единственным резервуаром.. Помогает автоматизировать процесс	По 1 на каждый резервуар	Каждый день
5	Директор нефтехранилища (игрок)	Управляющий всем нефтехранилищем. Распоряжается счетом нефтехранилища	Зависит от количества игроков	Каждый день

Часто на такой модели отображаются только имена сущностей (таблиц) без указания их атрибутов. Для построения концептуальной модели необходимо выделить сущности предметной области, их атрибуты и отношения. В таблице 9 приведена информация о типах связей между сущностями.

Таблица 9 – Описание типов связей между сущностями

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность	Ограничения
1	Директор (Игрок)	Нефтехранилище	Директор владеет нефтехранилищем	1:1	Обязательная	–
2	Нефтехранилище	Резервуар	В нефтехранилище есть множество резервуаров/Каждый резервуар обладает принадлежностью к нефтехранилищу	1:M	Обязательная	Нефтехранилище не может обладать менее чем 1 резервуаром. Резервуар обязан быть привязан к нефтехранилищу

Продолжение таблицы 9

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность	Ограничения
3	Резервуар	Насос	У резервуара есть 2 насоса (на закачку и на откачку)/Насос принадлежит определенному резервуару	1:M	Обязательная	У каждого резервуара есть ровно 2 насоса. Каждый насос прикреплен к резервуару

В таблице 10 представлено описание атрибутов сущности «Игрок».

Таблица 10 – Атрибуты сущности «Директор»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример
Идентификатор директора	Уникальный номер директора	Целый	4	0 – 4 294 967 296	Нет	PK	12
Логин	Кодовое имя директора	Символьный	128	A – яA – z0 – 9, – :ю	Нет		«Игрок»
Пароль	Кодовая последовательность для защиты данных директора	Символьный	128	A – яA – z0 – 9, – :	Нет		«Пароль»
Дата последнего сохранения	Дата необходимая для синхронизации между платформами	Дата	8	01.01.1970 – 31.12.2200	Нет		23.12.2021
Идентификатор нефтехранилища	Номер соответствующий принадлежащему директору нефтехранилищу	Целый	4	0 – 4 294 967 296	Нет	FK	12

В таблице 11 представлено описание атрибутов сущности «Нефтехранилище», а в таблице 12 сущности «Резервуар».

Таблица 11 – Атрибуты сущности «Нефтехранилище»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример
Идентификатор нефтехранилища	Уникальный номер нефтехранилища	Целый	4	0 – 4 294 967 296	Нет	PK	12
Количество резервуаров		Целый	2	0 – 65 636	Нет		12
Активный резервуар		Целый	2	0 – 65 636	Нет		12
Счет нефтехранилища	Количество денег, доступных для использования	Целый	4	0 – 4 294 967 296	Нет		12
Уровень менеджера	Менеджер, автоматически управляющий резервуаром	Целый	2	0 – 65 636	Есть		4

Таблица 12 – Атрибуты сущности «Резервуар»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример
Идентификатор резервуара	Уникальный номер резервуара	Целый	4	0 – 4 294 967 296	Нет	PK	12

Продолжение таблицы 12

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример
Идентификатор хранилища	Уникальный номер хранилища, к которому принадлежит резервуар	Целый	4	0 – 4 294 967 296	Нет	FK	12
Максимальный хранимый объем		Целый	4	0 – 4 294 967 296	Нет		12
Текущий хранимый объем		Целый	4	0 – 4 294 967 296	Нет		12
Минимальный хранимый объем		Целый	4	0 – 4 294 967 296	Нет		12

В таблице 13 представлено описание атрибутов сущности «Насос».

Таблица 13 – Атрибуты сущности «Насос»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример
Идентификатор насоса	Уникальный номер насоса	Целый	4	0 – 4 294 967 296	Нет	PK	12
Идентификатор резервуара	Уникальный номер резервуара, к которому принадлежит насос	Целый	4	0 – 4 294 967 296	Нет	FK	12
Мощность насоса	Количество нефти, которое насос перекачивает из точки А в точку Б за единицу времени	Целый	4	1 – 4 294 967 296	Нет		12

Продолжение таблицы 13

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример
Состояние насоса		Логический	1	0 – 1	Нет		1
Тип насоса		Логический	1	0 – 1	Нет		1

Из представленных ранее таблиц формируется концептуальная модель базы данных. Графическое представление концептуальной модели базы данных построенной средствами среды проектирования Visual Paradigm в виде диаграммы «сущность – связь» для предметной области информационной системы представлено на рисунке 13 [6].

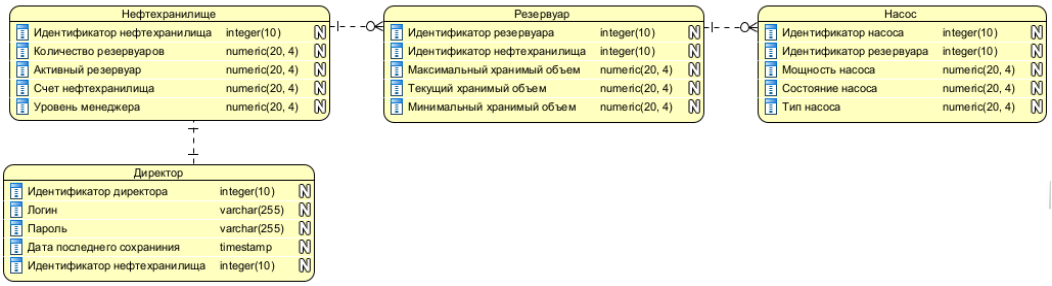

Рисунок 13 – Концептуальная модель базы данных в виде диаграммы «сущность – связь»

Следовательно, основными объектами в базе данных будут игроки, а также все физически существующие здания и устройства участвующие в процессе хранения, закачки и откачки нефтепродуктов.

2.3 Требования к обобщенной архитектуре приложения

При разработке информационных систем с применением баз данных (БД), как правило, используются две основные архитектуры:

– двухуровневая архитектура «Клиент – Сервер БД»;

– трехуровневая архитектура «Клиент – Сервер приложения – Сервер БД».

Двухуровневая архитектура основана на использовании клиентской части, которая содержит уровень представления данных, в то время, как на сервере баз данных располагается БД вместе с системами управления базами данных (СУБД) и прикладными программами.

Достоинствами двухуровневой архитектуры «Клиент – Сервер БД» обычно являются:

– снижение нагрузки на машины сервера и клиентов;

– снижение сетевого трафика и повышение эффективности обработки за счет оптимизации и буферизации ввода-вывода;

– защита данных средствами СУБД, позволяющая блокировать не разрешенные пользователю действия.

В то же время в качестве основного недостатка выделяют увеличенные потребности в ресурсах, связанные с повторением кода и запросов для нескольких клиентских приложений. На рисунке 14 представлена схема двухуровневой архитектуры.

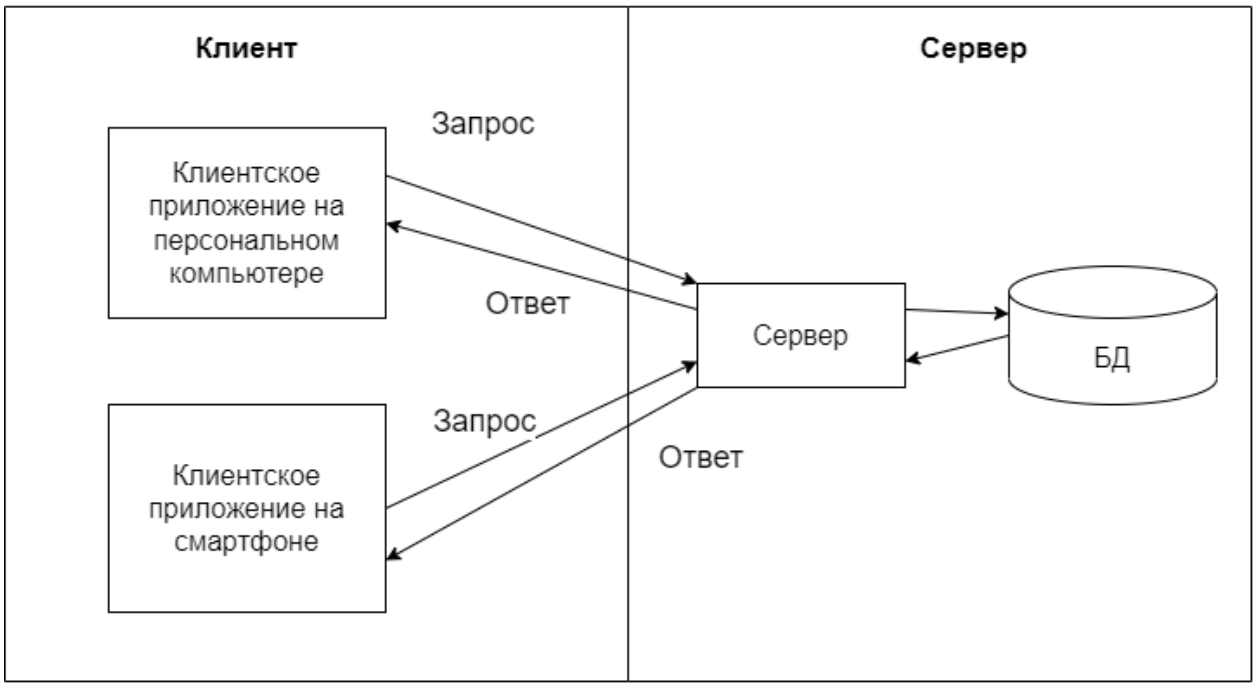

Рисунок 14 – Схема двухуровневой архитектуры

Трехуровневая архитектура предполагает построение программного средства (ПС) на основе 3 компонентов: клиентского приложения, сервера приложений и сервера БД.

Под клиентским приложением понимается компонент, предоставляемый конечному пользователю. Этот уровень не должен иметь прямых связей с базой данных, быть нагруженным основной бизнес-логикой и хранить состояние приложения по требованиям безопасности, масштабируемости и надежности ПС.

На промежуточном уровне располагается сервер приложений, на котором находится большая часть бизнес-логики ПС. Реализация данного компонента обеспечивается связующим программным обеспечением.

Сервер баз данных отвечает за хранение данных и находится на отдельном уровне. Сервер БД реализуется, как правило, средствами СУБД, подключение к которой обеспечивается только с уровня сервера приложений.

У трехуровневой архитектуры есть несколько преимуществ относительно клиент-серверной архитектуры:

– масштабируемость;

– конфигурируемость (изолированность уровней друг от друга);

– высокая безопасность;

– надежность.

Не лишена трехуровневая архитектура и недостатков (относительно архитектура клиент-сервер):

– высокая сложность создания приложений;

– относительная сложность в развертывании и администрировании;

– относительно высокие требования к производительности.

На рисунке 15 представлена схема трехуровневой архитектуры.

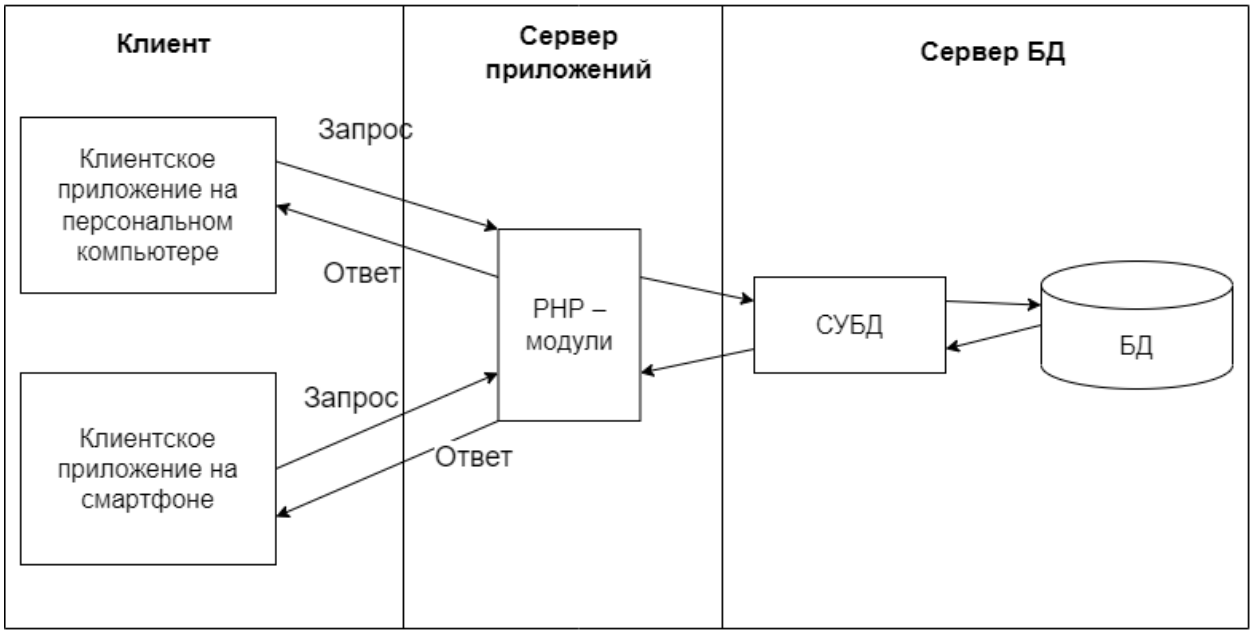

Рисунок 15 – Схема трехуровневой архитектуры

После изучения основных архитектур информационных систем было принято решение выбрать трехуровневую архитектуру «Клиент – Сервер приложения – Сервер БД».

Паттерны – это устойчивые конструкции в программировании [7]. Паттерны помогают сэкономить время на организацию кода и сосредоточиться на решении задачи. Существуют паттерны, позволяющие грамотно реализовать отдельные решения в коде, например паттерн «перечисление», есть паттерны, которые позволяют настраивать общение различных модулей, например паттерн «инверсия контроля». Применение паттернов позволяет упростить реализацию проекта, обеспечит читаемость кода, позволит использовать отдельные компоненты повторно, а также увеличит эффективность совместной разработки.

При разработке приложения с использованием пользовательского интерфейса используются паттерны, описывающие как эффективно разделить код приложения по папкам, что реализовывать в конкретных файлах, а также как настраивать связи между компонентами. Примером подобного паттерна является паттерн Model-View-Controller (MVC) и его аналоги Model-View-Presenter (MVP) и Model-View-ViewModel (MVVM) [8].

Все эти паттерны связаны с двумя общими сущностями – моделью (Model) и представлением (View).

Model (модель) – это часть, содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму модель.

Модель обладает множеством особенностей:

– это бизнес-логика приложения;

– обладает знаниями о себе самой и не знает о контроллерах и представлениях;

– может упрощать свою структуру до обычного слоя данных (DAO, база данных, XML-файл);

– зачастую используется как менеджер базы данных, набор объектов или просто логика приложения.

View (представление) – отображение данных, полученных от Model. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Представление обладает следующими признаками:

– реализует отображение данных, полученных от модели любым способом;

– лишь в исключительных случаях может содержать в себе бизнес-логику.

Впервые концепция MVC была описана в 1979 году. MVC состоит из трех компонентов: View (представление), Model (модель) и Controller (контроллер). Контроллер, содержит логику на изменение модели при определенных действиях пользователя.

Основная идея этого паттерна в том, что контроллер и представление зависят от модели, однако модель независима. Это как раз и позволяет разрабатывать и тестировать модель, ничего не зная о представлениях и контроллерах. При этом один и тот же контроллер, благодаря подобному строению, можно использовать для разных представлений. Так, например, контроллер может зависеть от пользователя, который вошел в систему.

Пользователь видит представление, на нем производит какие-то действия, эти действия представление перенаправляет контроллеру и подписывается на изменение данной модели, контроллер в свою очередь производит определенные действия над моделью данных, представление получает последнее состояние модели и отображает ее пользователю.

MVC используется, когда постоянная связь между представлением и другими частями приложения невозможна. Поскольку проектная идея MVC заключается в том, чтобы начать с модели без учета сложности пользовательского интерфейса, проблема, вызванная этим, заключается в том, что модели трудно соответствовать сложным и частым изменениям стороны представления.

По сравнению с этим MVP и MVVM более универсальны. Все они имеют независимый компонент (Presenter и ViewModel), соответствующие каждому View. Структура шаблона MVC представлена на рисунке 16.

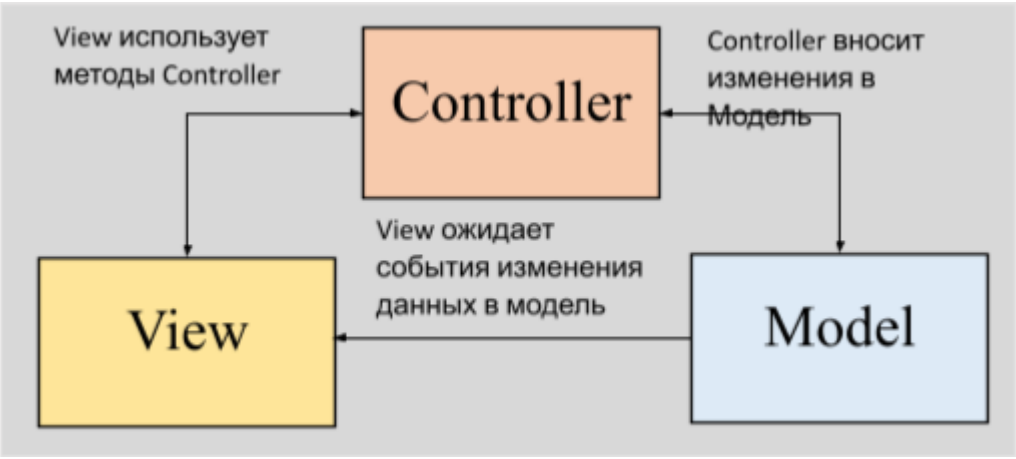

Рисунок 16 – Структура шаблона Model-View-Controller

Model-View-Presenter состоит из трех компонентов: Model (модель), View (представление) и Presenter. Представлению нет необходимости подписываться на изменения модели. На замену Controller приходит Presenter, который связывает компоненты представления с логикой модели.

Данный подход позволяет создавать абстракцию представления. Реализовать данный паттерн можно при помощи вынесения интерфейсов представления (IView). У каждого представления будут интерфейсы с определенными наборами методов и свойств, необходимых Presenter. Presenter в свою очередь инициализируется с данным интерфейсом, подписывается на события представления и по необходимости передает данные.

MVP используют в ситуации, когда связывание данных невозможно, либо требует слишком много дополнительного кода. Частым примером является технология Windows.Forms и реализация интерфейсов средствами среды разработки игровых приложений Unity.

В MVP Presenter полностью разделяют модель и представление, а основная логика программы реализована в Presenter. Мы можем изменять представление, не влияя на модель. Модель можно использовать более эффективно, поскольку все взаимодействия происходят в одном месте: внутри Presenter Мы можем использовать один Presenter для нескольких представлений без изменения логики Presenter. Эта возможность очень важна, потому что представление меняется чаще, чем модель. Если мы поместим логику в Presenter, то мы сможем протестировать логику без пользовательского интерфейса (модульный тест). Структура шаблона MVP представлена на рисунке 17.

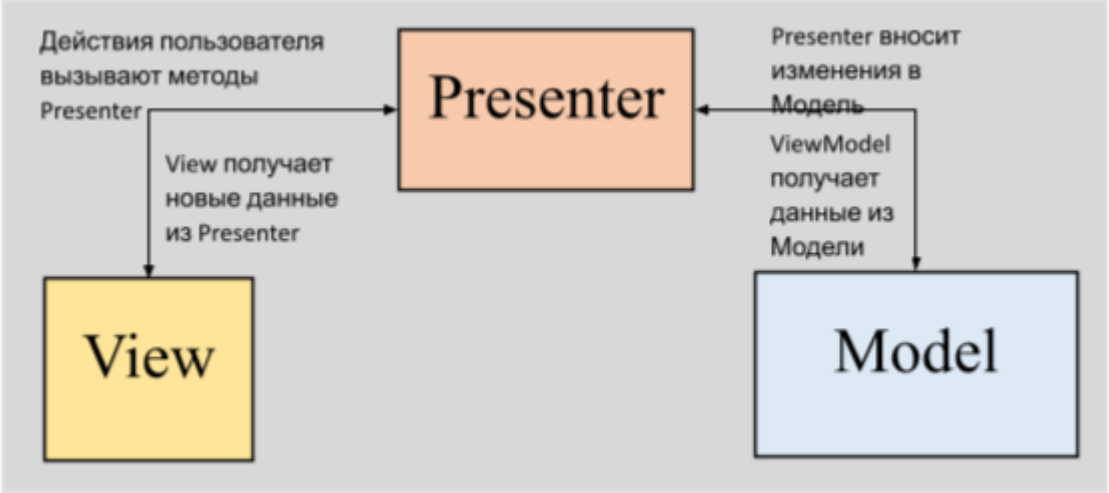

Рисунок 17 – Структура шаблона Model-View-Presenter

MVVM – шаблон, который появился для обхода ограничений паттернов MVC и MVP, и объединяющий некоторые из их сильных сторон. Эта модель впервые появилась в составе фреймворка «Small Talk» в 80-х, и была позднее улучшена с учетом обновленной MVP [9].

Шаблон MVVM имеет три основных компонента: Model (модель), View (представление), и ViewModel (представление – модель). В ViewModel содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя. Пользовательский интерфейс абстрагируется в независимые представления, а разработка, ориентированная на компоненты, преобразуется в разработку, ориентированную на представления. Каждое представление управляется независимой моделью представления (ViewModel).

MVVM чаще всего используется, когда возможно связывание данных без необходимости ввода специальных интерфейсов представления. Частым примером является технология WPF. При использовании этого паттерна, представление не реализует соответствующий интерфейс IView.

Представление должно иметь ссылку на источник данных, которым в данном случае является ViewModel. Элементы представления связаны с соответствующими свойствами и событиями ViewModel.

В свою очередь, ViewModel реализует специальный интерфейс, который используется для автоматического обновления элементов представления. Примером такого интерфейса в WPF может быть INotifyPropertyChanged. Структура шаблона MVVM представлена на рисунке 18.

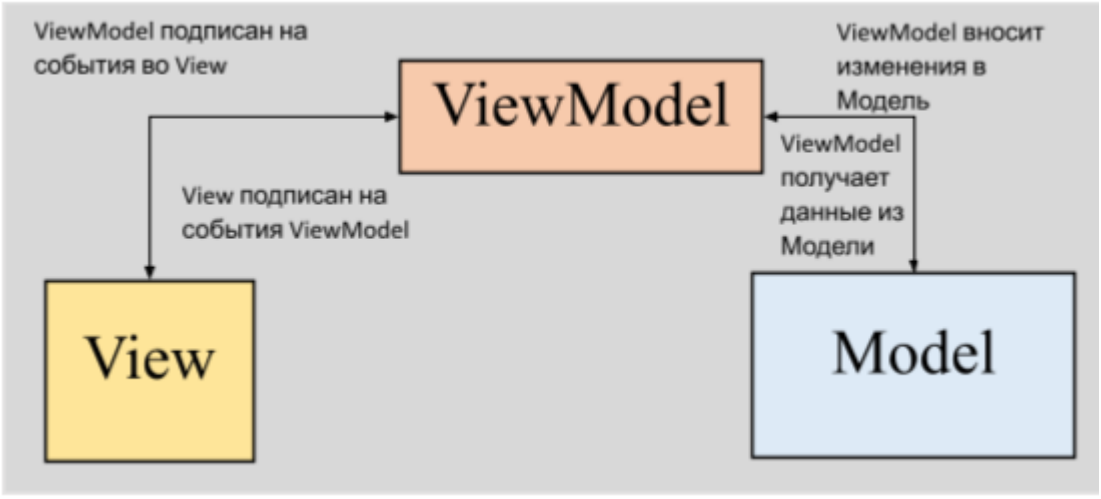

Рисунок 18 – Структура шаблона Model-View-ViewModel

Для реализации мультиплатформенного игрового приложения доступны все перечисленные выше паттерны проектирования пользовательского интерфейса. Однако MVVM и MVC являются избыточными. Для своей реализации MVVM потребует больше программного кода, чем основная логика нашего приложения. При этом у нас имеется возможность постоянного взаимодействия View и Model, следовательно использование MVC не обязательно. В ходе исследования было решено реализовать архитектуру приложения с использованием паттерна проектирования пользовательского интерфейса Model-View-Presenter.

2.4 Обоснование выбор технологий и средств разработки

2.4.1 Среда разработки игровых приложений Unity 3D

Современные разработчики программного обеспечения обладают широким спектром доступных технологий и средств разработки. Для разработки игрового ПО существует огромное количество сред разработки. Самыми распространенными технологиями для создания игровых приложений являются: Unity, Unreal Engine 4, Adobe AIR [10].

Unity с большим отрывом лидирует в этом списке. Среда разработки, изначально выпущенная в 2005 году эксклюзивно для Mac, сегодня стала самым распространенным кроссплатформенным инструментом для создания игр. Благодаря удобству использования, Unity делает создание игр максимально простым и комфортным, а мультиплатформенность движка позволяет разработчикам игровых приложений охватить как можно больше игровых платформ и операционных систем. На рисунке 19 представлен интерфейс среды разработки Unity со всеми основными панелями [4].

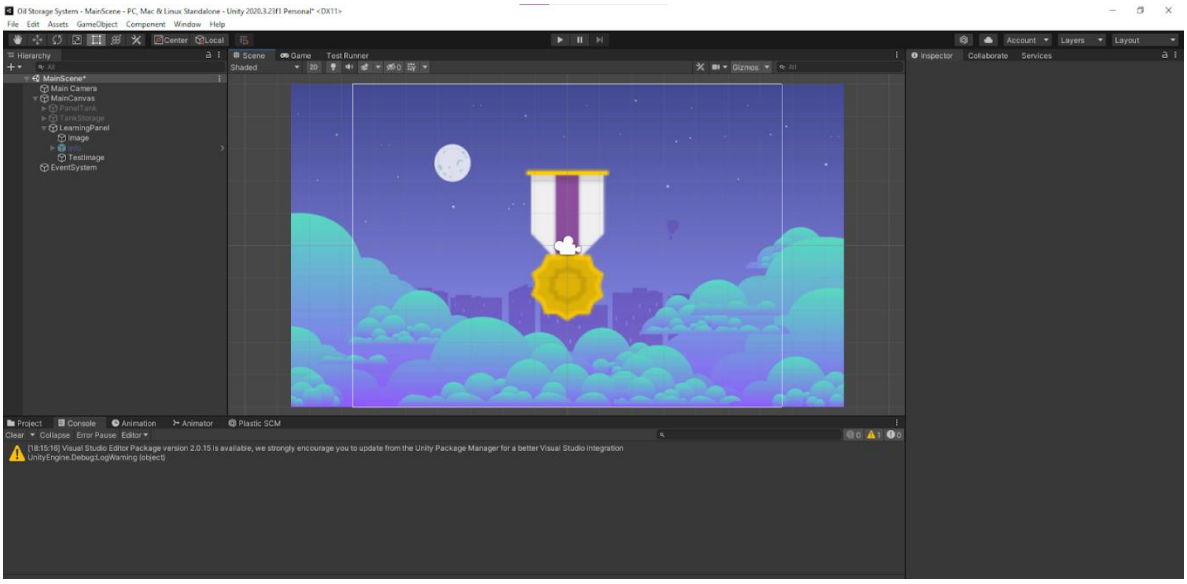

Рисунок 19 – Интерфейс среды разработки Unity

В качестве основных преимуществ Unity перед другими средами разработки игровых приложений выделяют:

– компонентно-ориентированное программирование;

– возможность использования бесплатной версии среды разработки с некоторыми ограничениями;

– огромная библиотека ассетов и плагинов (в том числе и бесплатных);

– поддержка огромного количества платформ, операционных систем и технологий;

– несколько доступных разработчику сценарных языков программирования: C#, JavaScript.

Исходя из вышеперечисленных преимуществ данной среды разработки, было решено использовать Unity для реализации небольшого мультиплатформенного приложения. Связано это с удобством и простотой интерфейса программы, наличию бесплатной версии программы, а также возможностью реализации игры на разные платформы и операционные системы. Также немаловажными факторами при выборе данного ПО являются наличие огромного количества дополнительных материалом и библиотек, доступных начинающему разработчику игр.

2.4.2 Сценарный язык программирования C#

Разработанный в начале 2000-х язык программирования (ЯП) C# является одним из самых распространенных языков программирования на данный момент. C# – ЯП сочетающий в себе объектно, аспектно, а также компонентно-ориентированные концепции. Данный язык программирования относится к семейству C – подобных языков, поэтому его синтаксис наиболее близок к таким языкам программирования как C++ и Java. C# перенял многое у своих предшественников, исправив многие значимые проблемы.

На данный момент язык программирования C# часто используется в разработке игровых приложений. Этому способствует тот факт, что с помощью C# можно упростить работу с DirectX, что помогает в разработке 3D приложений. В случае же разработки 2D приложений, разработчики часто полагаются на C# из-за:

– простоты в освоении;

– преимуществ языка компилируемого типа;

– надежности языка.

На рисунке 20 представлены фрагменты программного кода на языке C#.

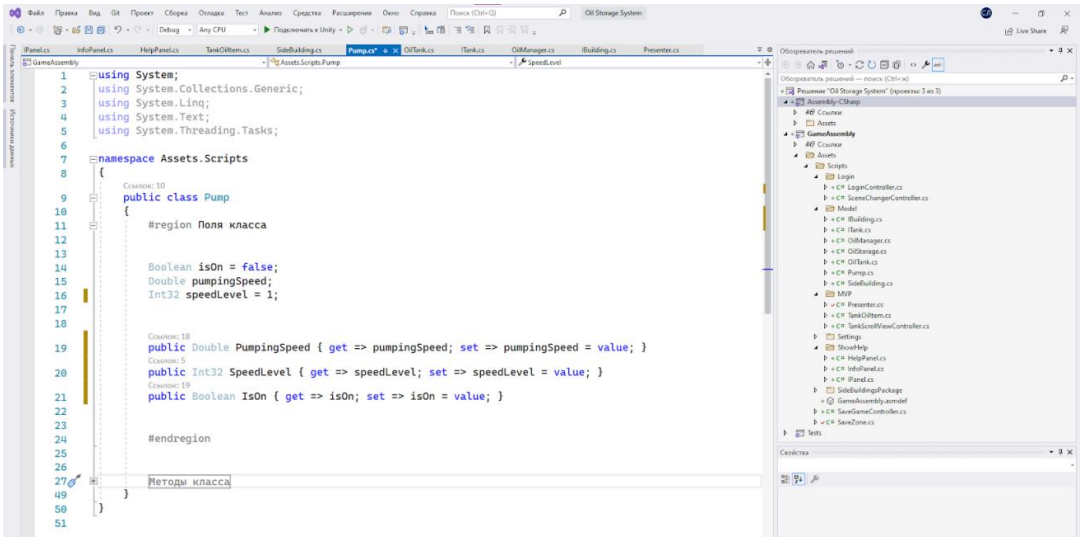

Рисунок 20 – Программный код на языке C#

Данный язык программирования обладает всеми необходимыми средствами и технологиями необходимыми для разработки мультиплатформенного игрового приложения с помощью среды разработки Unity.

2.4.3 Среда разработки программного обеспечения Visual Studio 2022

Кроме среды разработки игровых приложений Unity и языка программирования C# необходимо также выбрать среду разработки программного обеспечения, которая позволит писать программы на выбранном языке программирования.

Среди всех сред разработки программного обеспечения наиболее популярной остается интегрированная среда разработки Visual Studio. Ее эффективность доказана тем, что продукт остается лидером на рынке уже более двадцати лет [11].

Основное преимущество данной среды разработки в библиотеках, фреймворках и других инструментах. Также популярность Visual Studio обеспечивается теми языками программирования, возможность работы с которыми данная среда обеспечивает. На рисунке 21 представлено окно создания проекта в Visual Studio 2022.

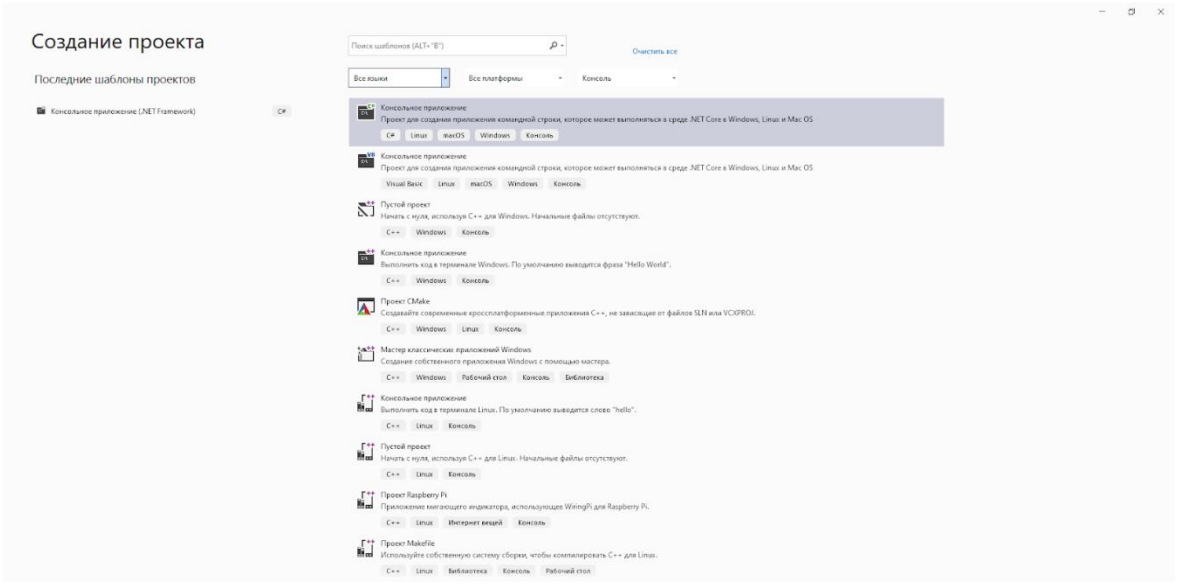

Рисунок 21 – Окно создания проекта в среде разработки Visual Studio 2022

В частности, Visual Studio поддерживает язык программирования C#, обеспечивает пользователя удобным интерфейсом для разработки, подсветкой синтаксиса и ошибок, а также, что не менее важно, обладает библиотекой, помогающий при разработке скриптов для Unity. Данные факторы послужили причиной того, что выбор пал на среду разработки программного обеспечения Visual Studio 2022.

2.5 Требования к программно-аппаратному обеспечению

Необходимым требованием к любому разрабатываемому программному обеспечению (ПО) является соответствие адекватным относительно сложности приложения программному и аппаратному обеспечению. В первую очередь эти требования способствуют более грамотному определению целевой аудитории пользователей ПО.

2.5.1 Требования к аппаратно-техническому обеспечению

Трехуровневая архитектура накладывает некоторые ограничения на используемые технические средства, в том числе, на устройства сервера и клиентов.

К минимальным техническим требованиям к устройству пользователя можно отнести:

– допустимая операционные система: Windows 7 or higher (64-bit) или Android 5.0;

– процессор: 1 GHz;

– оперативная память: 512 MB ОЗУ;

– место на диске: 1 GB.

Требования к компьютеру, на котором располагается сервер: любая ОС, поддерживающая PostgreSQL версии 8.0, 4ГБ ОЗУ, свободное место на жёстком диске по размеру планируемой базы данных.

Исходя из вышеперечисленных требований, можно сделать вывод, что большая часть устройств пользователей без особых проблем сможет поддерживать корректную работу игрового приложения.

2.5.2 Требования к системному программному обеспечению

Разрабатываемое игровое приложение должно работать на устройствах под управлением операционных систем (ОС) Windows (для персональных компьютеров) и Android (для смартфонов). Поддержка остальных ОС не планируется, но возможность портирования приложения под другие ОС необходимо обеспечить. Этому способствует разработка приложения в среде Unity.

2.5.3 Требования к прикладному обеспечению

Для разрабатываемого игрового приложения не требуется дополнительных прикладных программ. Версия для Windows поставляется в виде директории с файлом .exe, а также всеми необходимыми для

корректного функционирования ресурсами. Версия для Android предоставляется как файл-установщик с расширением .apk и после стандартной процедуры установки игровое приложение готово к запуску и корректной работе.

Управление в игровом приложении отличается для Windows и Android. В первом случае для игры будет достаточно работающего монитора и мыши. В случае установки игрового приложения на смартфон под управлением ОС Android достаточно только работающего сенсорного дисплея устройства.

3 Разработка информационного программного обеспечения приложения

3.1 Разработка архитектуры развертывания информационного и программного обеспечения приложения

В рамках реализации трехуровневой архитектуры приложения необходимо реализовать 3 основных модуля:

– сервер баз данных;

– сервер приложения;

– клиентское приложение.

Исходя из установленных требований, для реализации игрового приложения, необходимо обеспечить постоянный доступ к базе данных через информационно-коммуникационную сеть «Интернет».

Один из распространенных способов организации серверов – виртуальный хостинг. Это готовое решение для размещения сайтов, интернет-магазинов и прочих ресурсов. Также, виртуальный хостинг имеет широкий перечень ограничений в рамках разработки даже среднего уровня сложности. Конфигурации, ядро ОС, набор доступных языков, ресурсы процессора и так далее.

Heroku – один из самых популярных виртуальных хостингов. Это платформа как услуга, основанная на управляемой системе контейнеров, с интегрированными службами данных и мощной экосистемой для развертывания и запуска современных приложений. Данное решение отлично подойдет для проектов низкой и средней сложности, а также для тестирования работы приложения еще до аренды полноценного удаленного сервера.

Хостинг Heroku позволяет развернуть сервер приложений и сервер базы данных удаленно. На сервере приложений размещены PHP-модули, запросы к которым осуществляются из клиентского приложения. Сервер базы данных же представляет собой набор представлений и функций,

возвращающих необходимую информацию об игроках, их прогрессе и т.д. На рисунке 22 изображена обобщенная схема трехуровневой архитектуры игрового приложения.

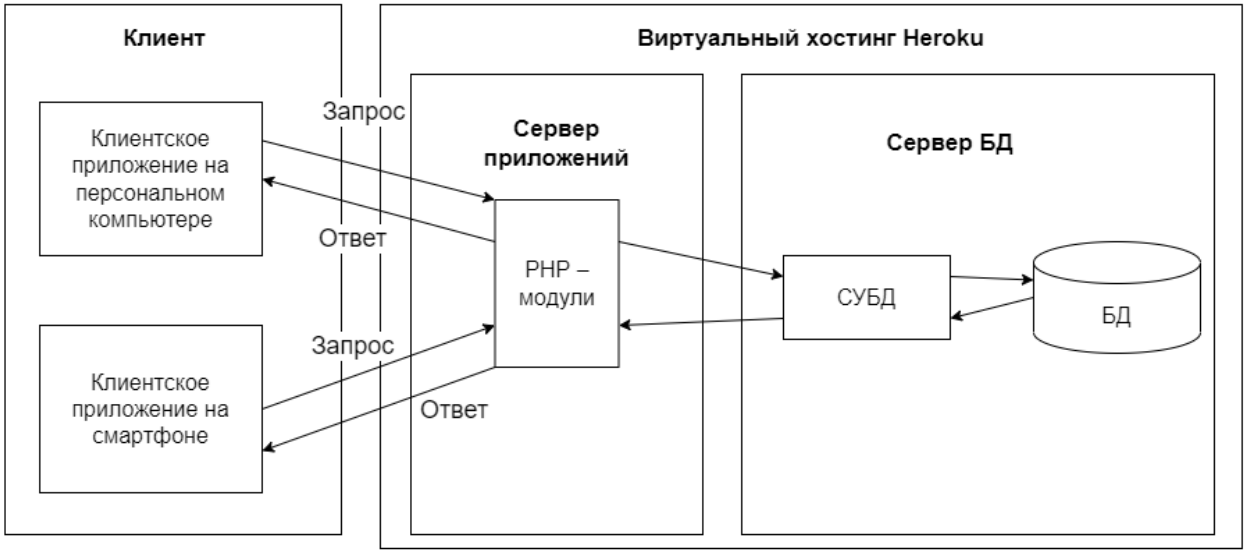

Рисунок 22 – Логическая модель БД

Таким образом, база данных была размещена на виртуальном хостинге, который в свою очередь выступает как сервер баз данных.

3.2 Разработка базы данных

3.2.1 Логическое моделирование базы данных

Дальнейшая разработка базы данных требует построение на основании концептуальной модели базы данных логической модели. Логическая модель – модель базы данных, выраженная в понятиях модели данных. Она позволяет расширить концептуальную модель путем определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними.

Игровые приложения зачастую уже в процессе эксплуатации получают различные дополнения. Следовательно, важно упростить процесс расширения базы данных. Было решено, в качестве основы для базы данных применить шаблон «Универсальная база данных».

В отличие от классической методики проектирования БД, в случае с «Универсальной БД» для каждой сущности не создается отдельная таблица. Сущности в подобной БД – объекты, хранящиеся в отдельной таблице, имеющий уникальный идентификатор и имеющий свойства, которые описывают атрибуты сущности. Также объекты могут быть связаны между собой произвольным образом. Такая связь характеризуется связанными объектами, а также типом связи [12].

Такая БД не привязана ни к какой бизнес модели и позволяет реализовать «над собой» практически любую бизнес-логику. Логика выделяется в отдельный программный слой и, как правило, реализуется на сервере приложений, где по запросу клиента создаются объекты, загружающие информацию о себе из БД и реализующие «поведение» объектов реального мира. В то же время, в силу однообразности модели хранения эти объекты не сложно создаются на основе базовых классов, инкапсулирующих функциональность по загрузке и сохранению свойств и связей в БД [6, 12]. Схема логической модели БД представлена на рисунке 23.

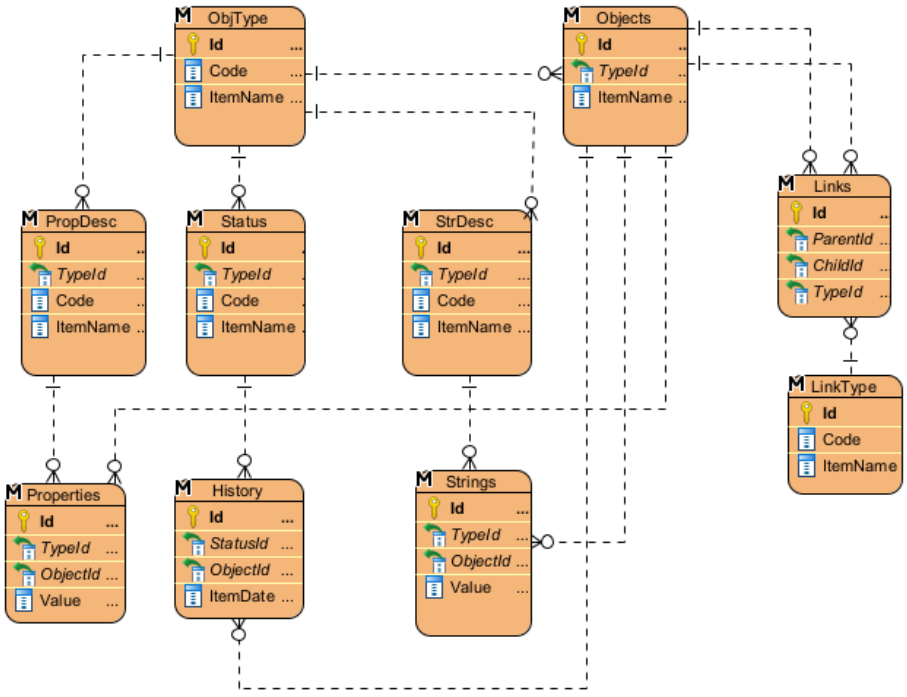

Рисунок 23 – Логическая модель БД

Логическая модель построена в среде Visual Paradigm [12, 13]. В логической модели представлены сущности, участвующие в бизнес-процессах управления нефтехранилищем.

3.2.2 Физическое моделирование базы данных

Физическая модель – логическая модель базы данных, выраженная в терминах языка описания конкретной СУБД. В данной выпускной квалификационной работе в терминах СУБД PostgreSQL. На рисунке 24 представлена физическая модель базы данных [6].

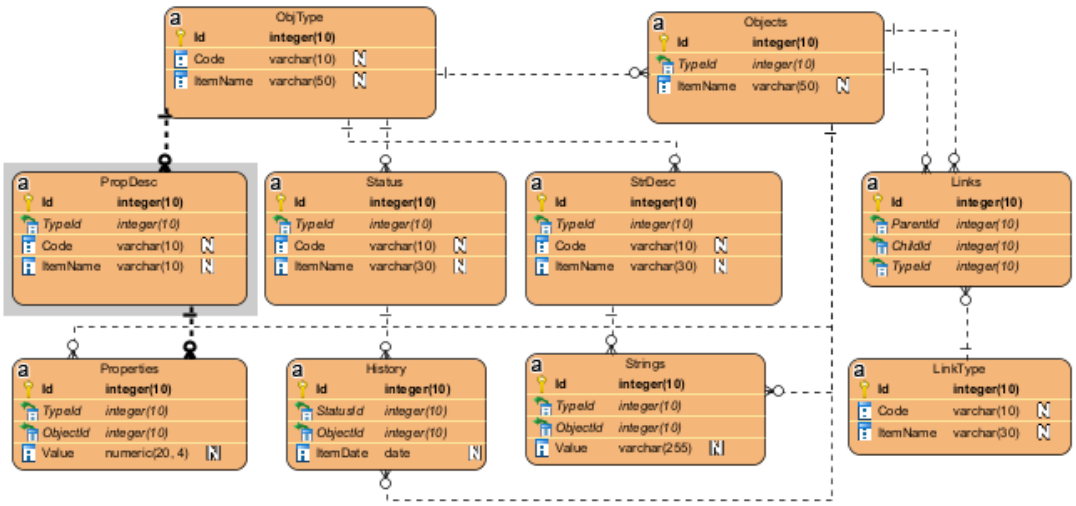

Рисунок 24 – Физическая модель БД

Физическая модель построена в среде Visual Paradigm. Согласно физической модели, была создана база данных в СУБД PostgreSQL.

3.2.3 Развертывание базы данных средствами СУБД PostgreSQL на Heroku

База данных размещена на виртуальном хостинге Heroku, что позволяет обеспечить доступ к базе данных из любой точки планеты. Единственное требование – наличие подключения к сети «Интернет». На рисунке 25 изображено меню управления базой данных и тарифами на хостинге Heroku. Предоставляется информация о количестве записей и их

лимит для тарифа, количество таблиц, а также объем занимаемой на сервере памяти.

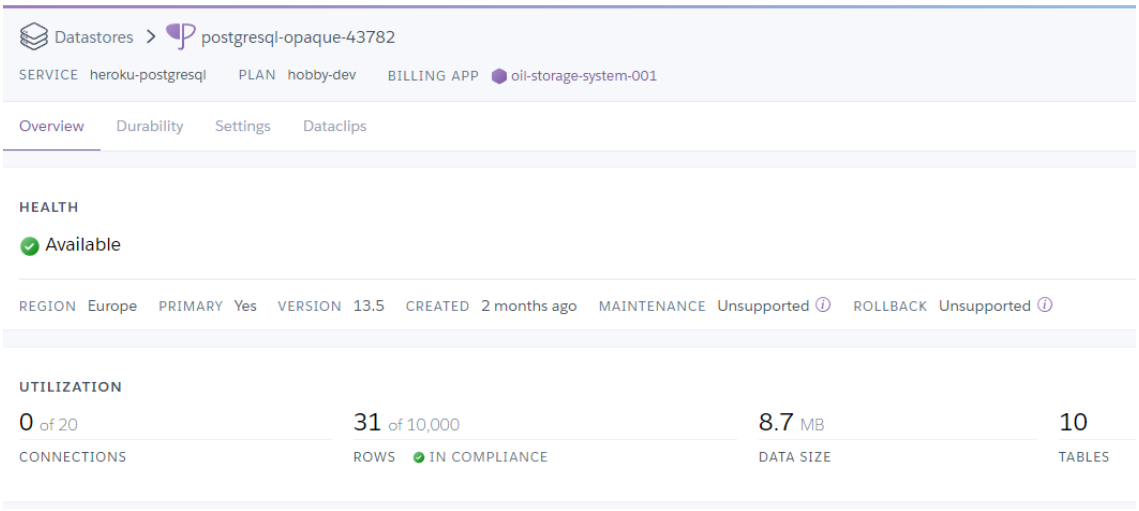

Рисунок 25 – Меню управления БД на хостинге Heroku

На основании физической модели базы данных средствами СУБД PostgreSQL в IDE DataGrip была развернута база данных игрового приложения. Итоговая схема базы данных изображена на рисунке 26.

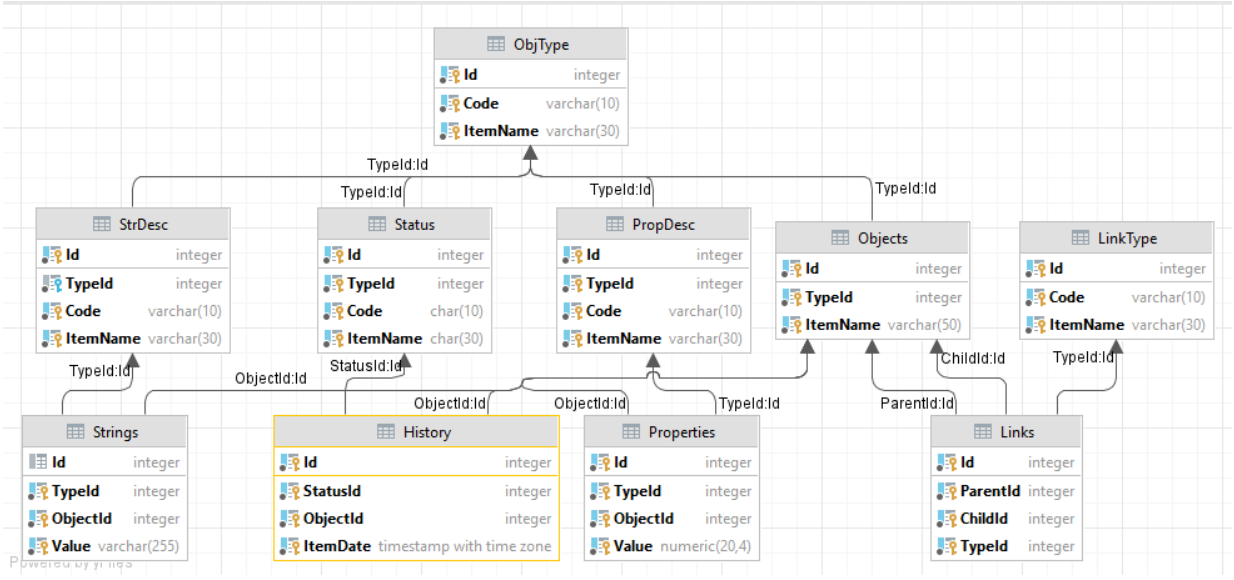

Рисунок 26 – Схема итоговой базы данных

Благодаря универсальной модели базы данных при добавлении в игровое приложение дополнительных сущностей не придется добавлять дополнительные таблицы. Необходимо будет только написать запросы, представления и процедуры.

3.3 Разработка серверного программного обеспечения

Серверная часть мультиплатформенного приложения – набор php-скриптов, которые располагаются на хостинге Heroku. Когда необходимо взаимодействие с базой данных, клиентское приложение обращается по адресу к скриптам на сервере, которые в свою очередь выполняют запросы к таблицам и представлениям базы данных. Следовательно, в рамках реализации серверного ПО, необходимо разработать набор php-скриптов, sql-запросов и представлений для базы данных.

Взаимодействие с базой данных разрабатываемого приложения предполагает получение и загрузку данных о пользователях, в первую очередь логины и пароли. Запрос на получение информации необходимой для авторизации пользователя представлен на рисунке 27. В результате выполнения данного запроса формируется таблица с логинами и паролями всех зарегистрированных на момент запроса пользователей.

```
SELECT Login."Value" as "Логин", Pass."Value" as "Пароль"
  FROM "Objects" O
    INNER JOIN "Strings" Login 1<->1..n: ON O."Id" = Login."ObjectId"
    INNER JOIN "StrDesc" LoginType 1..n<->1: ON Login."TypeId" = LoginType."Id"
    INNER JOIN "Strings" Pass 1<->1..n: ON O."Id" = Pass."ObjectId"
    INNER JOIN "StrDesc" PassType 1..n<->1: ON Pass."TypeId" = PassType."Id"
WHERE LoginType."Code" = 'Login' AND
      PassType."Code" = 'Password';
```

Рисунок 27 – Код запроса на получение данных о пользователях из базы данных

Результирующая таблица представлена на рисунке 28.

	"Логин"	"Пароль"
1	Admin1115	Admin1115

Рисунок 28 – Список пользователей с их логинами и паролями

Ключевыми процессами игрового приложения являются авторизация и регистрация пользователя. За оба этих процесса со стороны клиента отвечает

C#-скрипт «LoginController.cs». В зависимости от действий пользователя, требуется создать нового игрока, либо удостовериться в правильности введенного пользователем пароля. Код скрипта приведен в приложении Б.

В случае если пользователь нажал кнопку «авторизация», клиентское приложение отправляет запрос на выполнение серверного php-скрипта «login.php». В рамках данного скрипта выполняется запрос на данные пользователя с логином, полученным из запроса клиентского приложения. Если пользователя не существует, возвращается соответствующее сообщение. В случае, когда пользователь существует, полученный из БД пароль сравнивается с паролем, полученным от клиентского приложения. В случае совпадения возвращается сообщение об успехе, а в случае с несоответствием сообщение о неправильно введенных данных.

На рисунке 29 представлен пример взаимодействия клиента, сервера приложений и сервера базы данных.

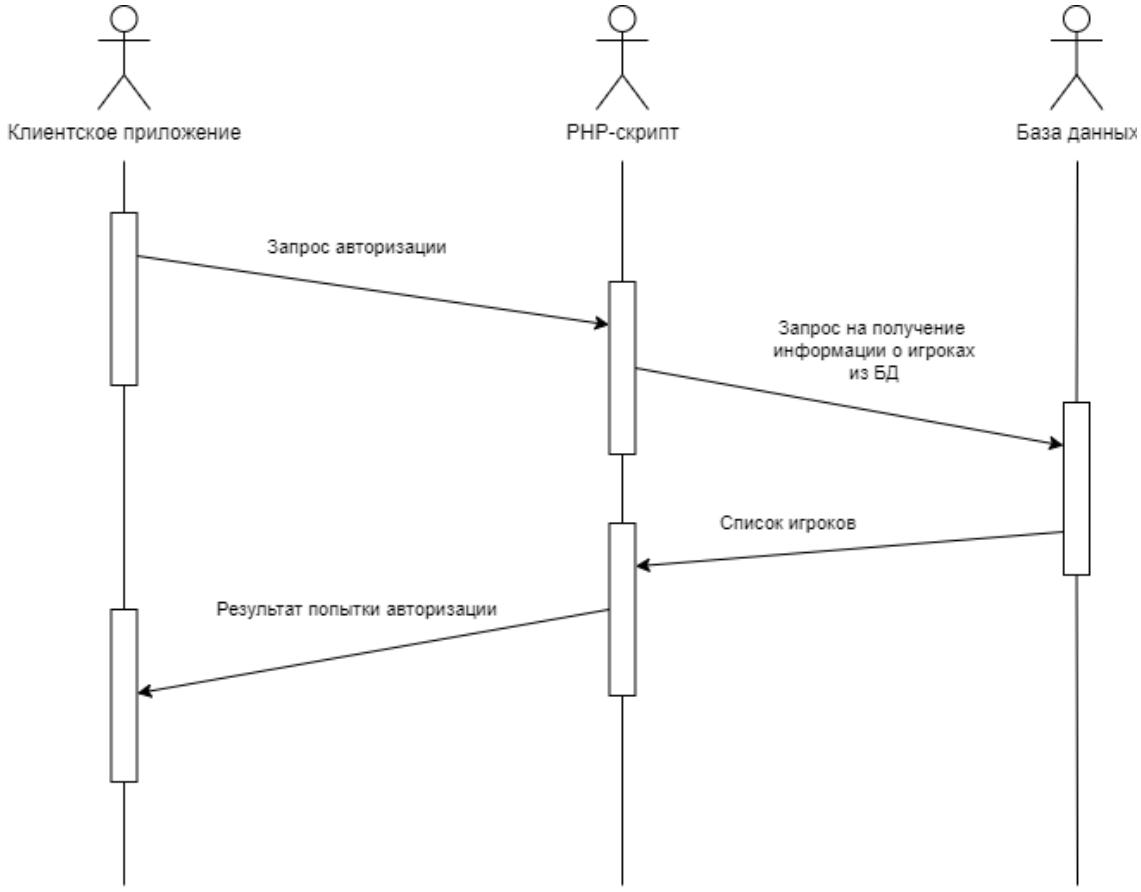

Рисунок 29 – Пример взаимодействия в трехуровневой структуре

Код скрипта «login.php» приведен в приложении Б.

Регистрация нового пользователя начинается так же, как и авторизация, с C#-скрипта «LoginController.cs», однако пользователем была нажата кнопка «регистрация». Клиентское приложение вызывает другой серверный php-скрипт «register.php».

В результате выполнения скрипта, если данные были введены корректно (пользователь не существует) клиентскому приложению возвращается сообщение о том, что пользователь создан. В противном случае будет возвращено соответствующее сообщение.

Скрипт «register.php» также приведен в приложении Б.

3.4 Разработка клиентского программного обеспечения

3.4.1 Разработка пользовательского интерфейса клиентского приложения

Пользовательский интерфейс клиентского приложения должен обеспечивать привычное и удобное представление информации, простое и эффективное выполнение основных функций пользователей.

Интерфейс пользователя для приложения управления нефтехранилищем будет состоять из четырех основных сцен [2, 5, 6]:

– окно управления резервуаром;

– окно выбора активного резервуара;

– окно управления дополнительными постройками;

– окно авторизации и регистрации.

В качестве основной игровой сцены выступает «окно управления резервуаром». Управление резервуаром осуществляется нажатием на кнопки запуска насосов закачки и откачки, а также с помощью панели улучшения резервуара.

На рисунке 30 представлено окно управления резервуаром.

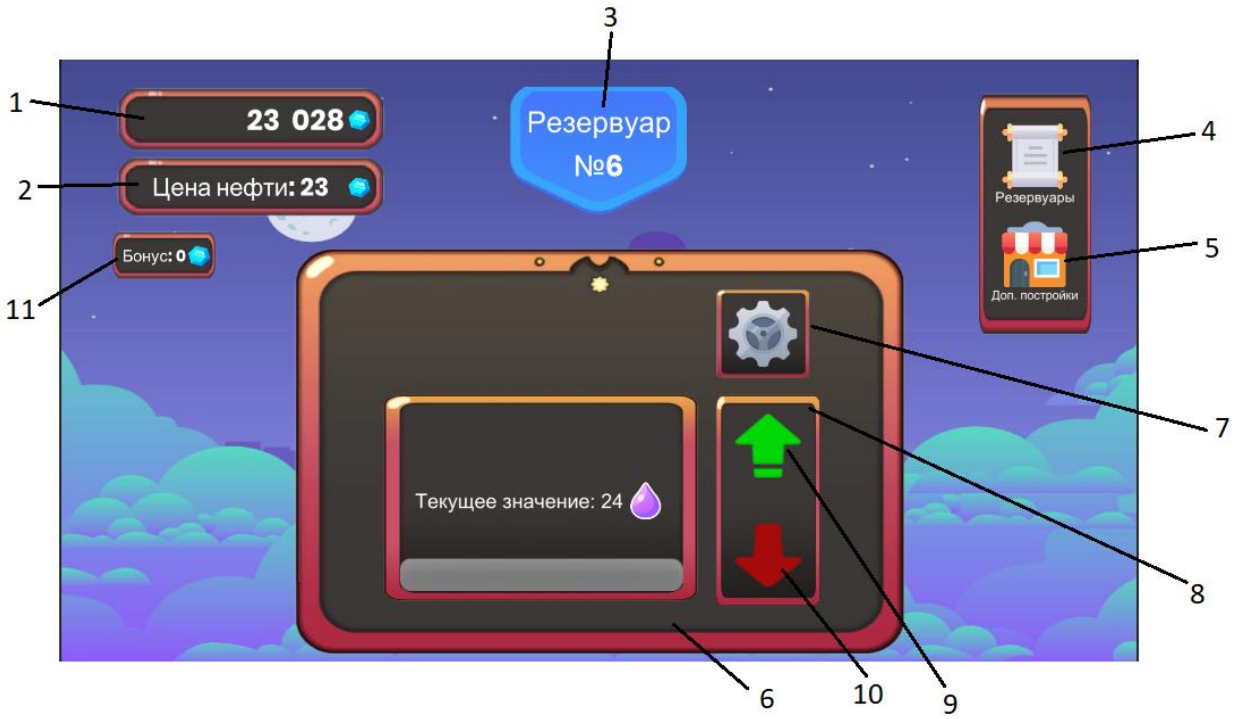

Рисунок 30 – Интерфейс окна управления резервуаром

На окне управления резервуаром можно выделить следующие значимые элементы:

1 – бюджет нефтехранилища;

2 – текущая цена на нефтепродукты;

3 – номер активного резервуара;

4 – кнопка «Резервуары», открывающая сцену выбора активного резервуара;

5 – кнопка «Доп. постройки», позволяющая перейти к окну управления дополнительными зданиями;

6 – панель управления резервуаром;

7 – панель улучшений резервуара;

8 – панель управления насосами резервуара;

9 – кнопка включения/выключения насоса закачки;

10 – кнопка включения/выключения насоса откачки.

Таким образом, игрок получает исчерпывающую информацию о состоянии бюджета, об основных параметрах, необходимых для торговли

нефтепродуктами, о наполненности резервуара, а также имеет возможность улучшать резервуар и переходить к другим игровым сценам.

Окно выбора активного резервуара отображает список уже входящих в нефтехранилище резервуаров, представляет возможности выбора активного резервуара с последующим переходом к окну управления резервуаром. На рисунке 31 представлено окно выбора активного резервуара.

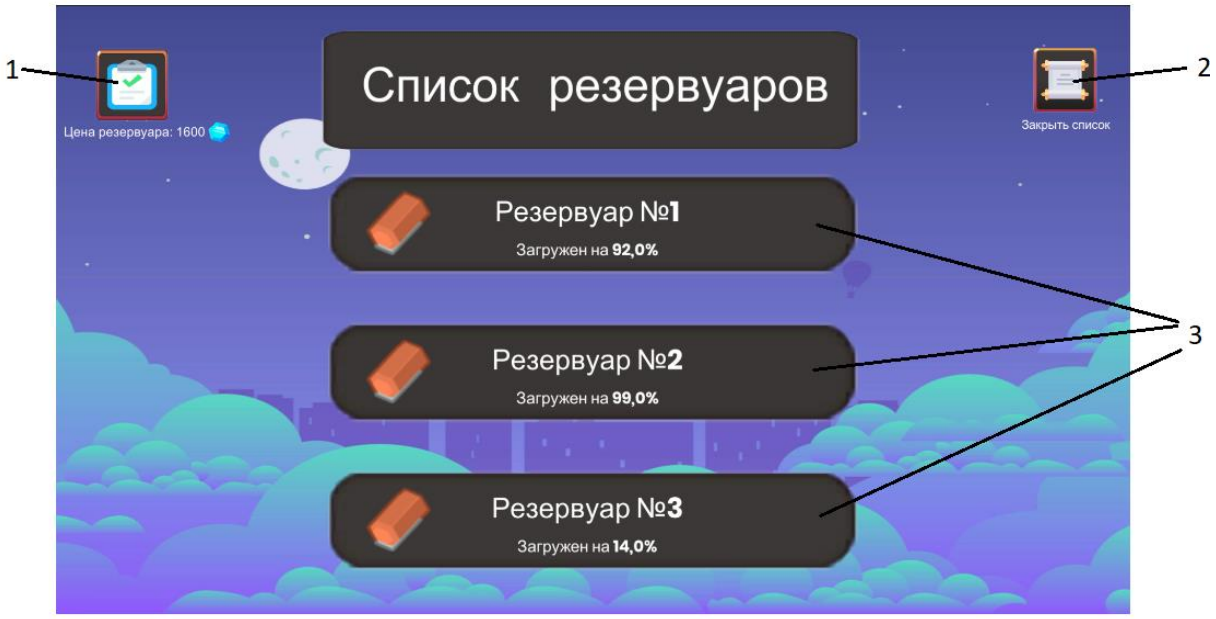

Рисунок 31 – Интерфейс окна выбора активного резервуара

На окне выбора активного резервуара можно выделить следующие значимые элементы:

1 – кнопка «Купить резервуар», позволяющая добавить к нефтехранилищу новый резервуар;

2 – кнопка «Закрыть список», позволяет вернуться к окну управления резервуаром, не меняя при этом активный резервуар;

3 – элементы списка резервуаров.

Также кнопка «Купить резервуар» позволяет добавить к нефтехранилищу еще один резервуар. Нажатие на элемент списка резервуаров поменяет номер активного резервуара, и вернет игрока к окну управления резервуаром. Однако после этого действия информация на экране будет выведена в соответствии с состоянием нового активного резервуара.

Окно управления дополнительными постройками позволяет игроку получить временные бонусы, помогающие увеличить эффективность работы нефтехранилища. На рисунке 32 представлено окно управления дополнительными постройками.

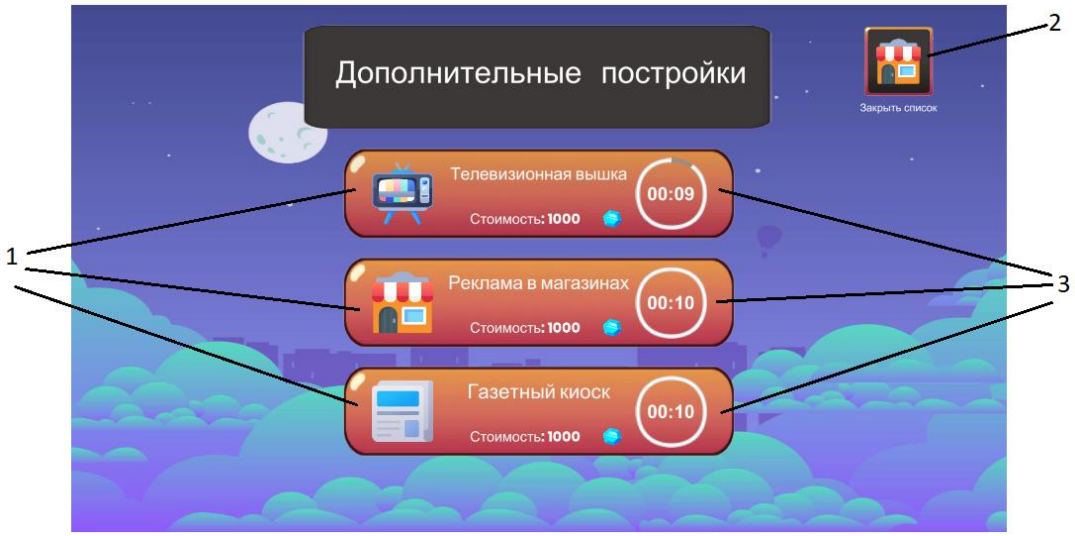

Рисунок 32 – Интерфейс окна управления дополнительными постройками

На окне управления дополнительными постройками можно выделить следующие значимые элементы:

1 – кнопки покупки бонусов;

2 – кнопка «Закрыть список», позволяет вернуться к окну управления резервуаром;

3 –таймеры действия бонусов.

Окно авторизации и регистрации открывается первым после запуска игрового приложения. Здесь игроку предстоит либо ввести данные от уже существующего пользователя и продолжить игру, либо создать нового пользователя и начать игру с нуля. На рисунке 33 представлено окно авторизации и регистрации.

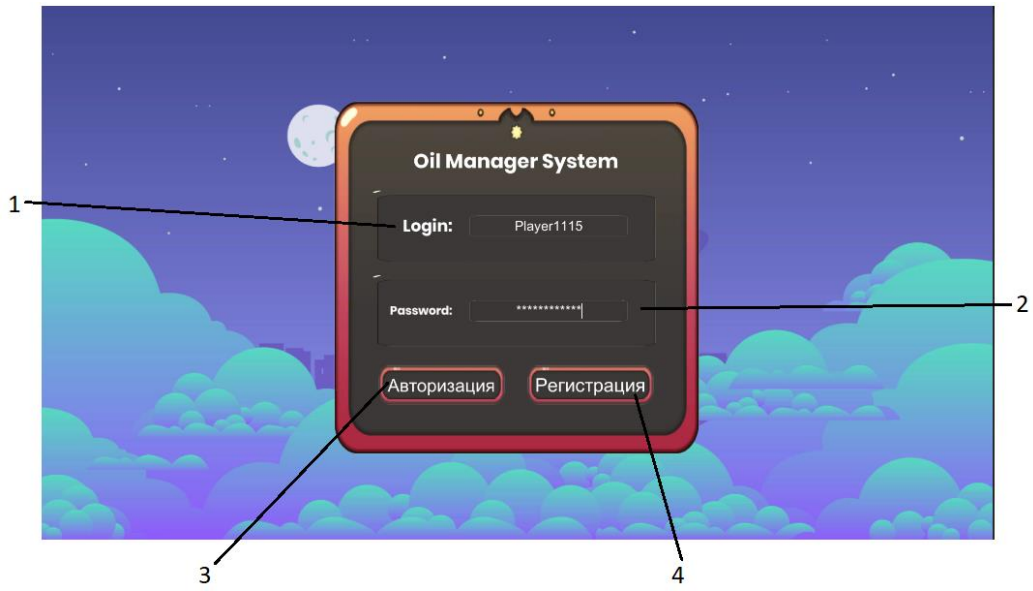

Рисунок 33 – Интерфейс окна авторизации и регистрации

На окне авторизации и регистрации можно выделить следующие значимые элементы:

1 – панель ввода логина;

2 – панель ввода пароля;

3 – кнопка «Авторизация»;

4 – кнопка «Регистрация».

В ситуации, когда пользователь не имеет своего аккаунта, имеется возможность зарегистрироваться. После прохождения простой процедуры регистрации пользователь сразу может начать игру. Для этого необходимо нажать кнопку «Авторизация»

3.4.2 Разработка программного обеспечения клиентского приложения

Было реализовано мультиплатформенное игровое приложение управления нефтехранилищем. Исходный код программы на языке C# представлен в приложении Б.

На диаграмме можно выделить следующие типы классов:

– классы модели;

– классы представления;

– классы Presenter.

К классам модели относятся классы, реализующие основные сущности приложения: нефтехранилище (OilStorage), резервуар (OilTank), насос (Pump), менеджер резервуара (Manager), а также класс отвечающие за взаимодействие с Базой данных.

Классы, отвечающие за отображение на экране информации о резервуарах, списке резервуаров и т.д. относятся к классам представления. К данным классам относятся базовые классы Unity, такие как Button, Image, Panel, Text и другие.

Классы, связывающие модель и представление, относятся к классам Presenter. Данный класс, как основа паттерна проектирования пользовательского интерфейса MVP является связующим звеном между классами модели и классами интерфейса [4].

На рисунке 34 изображена полная диаграмма классов разработанного игрового приложения.

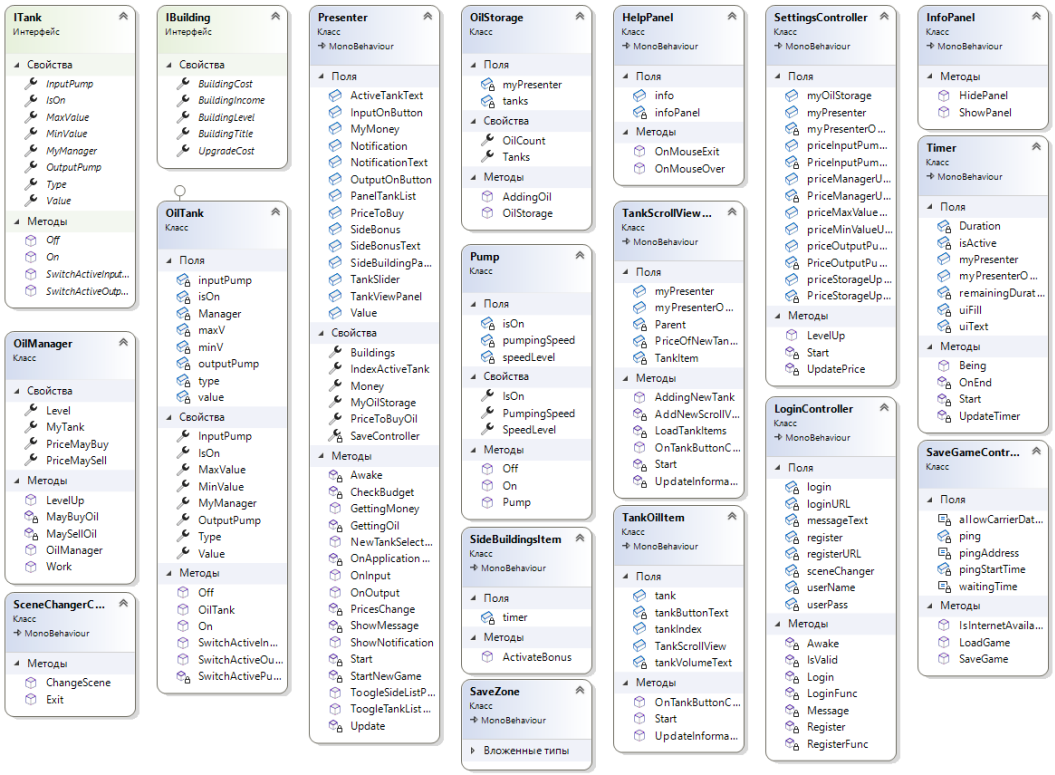

Рисунок 34 – Диаграмма классов

IBuilding – ключевой интерфейс для всех игровых построек.

ITank – интерфейс резервуаров.

OilStorage – класс модели, описывающий работу нефтехранилища.

OilTank – класс модели, описывающий работу насоса.

Pump – класс модели, описывающий работу насоса.

OilManager – класс модели, описывающий работу управляющего резервуаром.

SaveZone – структура данных сохранения прогресса.

SaveGameController – класс модели, контролирующий сохранение и загрузку прогресса игры в виде структуры SaveZone.

SettingsController – класс модели, отвечающий за улучшение резервуара. Также вычисляет стоимость улучшений.

Presenter – центральный класс одноименной группы классов архитектуры MVP. Обеспечивает взаимодействие с моделью и получение информации от представления на активном игровом окне.

HelpPanel – класс презентера. Позволяет отобразить всплывающие подсказки.

InfoPanel – класс информационной всплывающей подсказки.

TankOilItem – класс презентера, отвечающий за отрисовку элемента списка резервуаров на окне выбора активного резервуара.

TankScrollViewController – класс презентера, контролирующий работу контейнера, способного прокручивать содержимое в виде списка TankOilItem.

SideBuildingsItem – аналогично классу TankOilItem помогает отобразить список доступных дополнительных построек.

Timer – класс презентера, позволяющий отобразить время действия бонуса от дополнительной постройки.

Таким образом, была разработана клиентская часть программного обеспечения, удовлетворяющая всем поставленным требованиям.

4 Контроль качества программного обеспечения приложения

Важным этапом разработки является контроль качества программного обеспечения. Контроль качества означает проверку выполнения требований к ПО и общую оценку разработки [14]. Это необходимо в первую очередь для того, чтобы вычислить ошибки и исправить их еще до представления продукта конечному потребителю.

4.1 Планирование процесса контроля качества программного обеспечения

4.1.1 Мероприятия по контролю качества программного обеспечения

Контроль качества программного обеспечения всего представляет собой тщательное тестирование всех основных аспектов приложения, проверка выполнения требований к приложению, а также расчет наиболее используемых метрик качества кода. Контроль над качеством разрабатываемого игрового приложения было решено проводить с использованием обоих распространенных методов.

4.1.2 Обоснование выбора модели тестирования

Тестирование программного обеспечения – процедура проверки соответствия между реальным и предполагаемым поведением программы [15]. Процедура тестирования предполагает формирование набора тестов, которые выбираются некоторым определенным требованиями образом. В рамках данной выпускной квалификационной работы были выбраны две методологии тестирования: функциональное и модульное. Функциональное тестирование проведено в формате «черного ящика», тесты сформированы на основании функциональных требований [16]. С помощью модульных тестов проверена работоспособность модулей, с которыми чаще всего взаимодействует пользователь во время игры.

4.1.3 Обоснование выбора метрик программного кода

Основной причиной проведения оценки качества кода является возможность и удобство сопровождения и модификации разработанного программного обеспечения. Современное программное обеспечение характеризуется сложностью создания надежного и поддерживаемого кода. В связи с необходимостью получения более глубокого представления о разрабатываемом коде, применяются метрики кода, которые представляют собой набор оценок ПО. Используя преимущества метрик кода, разработчики могут понять, какие типы и/или методы следует переработать или тщательно протестировать [17].

4.2 Тестирование программного обеспечения

4.2.1 Функциональное тестирование

Функциональное тестирование – процесс, основанный на заранее известном или предполагаемом поведении пользователя, основанный на детальном анализе и изучении функциональных требований к приложению, системе или небольшому модулю. Такое тестирование проводится функциональными тестами, которые проектируются и создаются исходя из сформированных на этапе проектирования бизнес-процессов. При таком функциональном тестировании программиста интересует в целом, может ли пользователь от начала до конца пройти весь бизнес сценарий. Тестовые случаи при таком тестировании называют сценариями использования.

План тестирования приведен в таблице 14. Результаты тестов представлены в приложении В.

Таблица 14 – План функционального тестирования

№	Вариант использования	Тест	Результаты
1	Следить за изменением цены на нефть	Отображение цены на экране	Тест выполнен (рисунок В.1)

Продолжение таблицы 14

№	Вариант использования	Тест	Результаты
2	Управлять работой резервуара	Отображение состояния резервуара	Тест выполнен (рисунок В.2)
3	Управлять работой резервуара	Активация насоса закачки (покупка нефтепродуктов)	Тест выполнен (рисунок В.3 – В.4)
4	Управлять работой резервуара	Активация насоса откачки (продажа нефтепродуктов)	Тест выполнен (рисунок В.5 – В.6)
5	Вычислить доступный для расширения бюджет	Отображение бюджета нефтехранилища	Тест выполнен (рисунок В.7)
6	Расширение нефтехранилища (добавление нового резервуара)	Отображение купленного резервуара (если средств достаточно)	Тест выполнен (рисунок В.8)
7	Увеличить продуктивность нефтехранилища	Увеличить объем резервуара (если средств достаточно)	Тест выполнен (рисунок В.9)
8	Увеличить продуктивность нефтехранилища	Улучшить насос закачки резервуара	Тест выполнен (рисунок В.10)
9	Увеличить продуктивность нефтехранилища	Улучшить насос откачки резервуара	Тест выполнен (рисунок В.11)
10	Инвестировать в побочные здания	Покупка дополнительного здания	Тест выполнен (рисунок В.12)
11	Увеличить продуктивность нефтехранилища	Недостаток средств на покупку улучшения	Тест выполнен (рисунок В.13)

Исходя из результатов функционального тестирования, можно сделать вывод, что все основные функциональные требования к разрабатываемому игровому приложению были выполнены.

4.2.2 Модульное тестирование

Модульное тестирование (unit testing) – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения [15]. Цель данного тестирования заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Для проведения модульного тестирования были

использованы встроенные в Unity инструменты для написания модульных тестов. Результаты модульного тестирования представлены на рисунке 35.

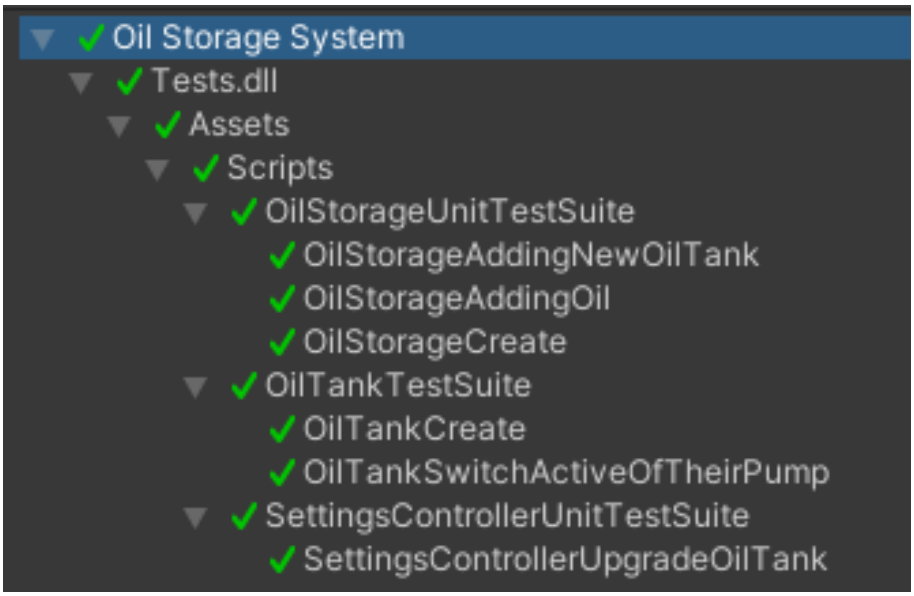

Рисунок 35 – Результаты прохождения модульных тестов

Модульные тесты прошли без ошибок и сбоев, что свидетельствует о надежности и стабильности информационной системы.

4.3 Оценка качества программного кода

4.3.1 Оценка качества программного кода средствами среды разработки Visual Studio 2022

Среда разработки программного обеспечения Visual Studio 2022 предоставляет удобные инструменты для оценки основных метрик программного кода, таких как:

– индекс удобства обслуживания;

– цикломатическая сложность;

– глубина наследования;

– объединение классов;

– строки исходного кода;

– строки исполняемого кода.

Индекс удобства обслуживания показывает относительную простоту обслуживания кода, для чего вычисляется значение индекса от 0 до 100. Высокое значение означает лучшую поддержку.

Цикломатическая сложность измеряет структурную сложность кода. Данный показатель формируется путем вычисления количества различных путей кода в потоке программы.

Глубина наследования указывает количество различных классов, наследуемых друг от друга.

Объединение классов измеряет связь с уникальными классами с помощью параметров, локальных переменных, возвращаемых типов, вызовов методов, универсальных или шаблонных экземпляров, базовых классов, реализаций интерфейсов, полей, определенных во внешних типах и оформлении атрибутов.

На рисунке 36 представлены сгенерированные с помощью Visual Studio 2022 метрики качества кода, перечисленные выше.

Иерархия	Индекс удобства подд...	Сложность органи...	Глубина наследова...	Взаимозависимост...	Строки исходного кода	Строки испо...
Tests (Debug)	73	16	5	12	0	74
GameAssembly (Debug)	84	206	5	52	1 510	351
{ } Assets.Scripts	84	206	5	52	1 510	351
IBuilding	100	10	0	0	19	0
ITank	100	19	0	2	41	0
SaveZone.SaveData	100	1	1	2	17	0
SaveZone	100	1	5	1	20	0
SideBuildingsItem	95	1	5	4	9	1
SceneChangerController	100	2	5	3	21	2
HelpPanel	90	3	5	4	20	3
InfoPanel	91	2	5	3	18	3
TankOilItem	87	4	5	8	44	5
Pump	92	9	1	0	41	12
OilManager	89	14	1	3	83	15
TankScrollViewController	74	9	5	14	67	20
Timer	72	7	5	12	71	22
OilTank	83	22	1	3	101	34
OilStorage	65	17	1	6	119	34
LoginController	71	17	5	15	160	42
SettingsController	59	15	5	12	182	43
SaveGameController	60	10	1	18	134	46
Presenter	74	43	5	20	305	69

Рисунок 36 – Метрики качества кода сгенерированные Visual Studio 2022

Метрики, сгенерированные средой разработки Visual Studio 2022, говорят о том, что код разрабатываемого игрового приложения в целом удовлетворяет всем требованиям качества кода.

4.3.2 Метрика Чепина

В процессе анализа качества кода разработанного программного обеспечения часто применяется метрика Чепина. Данная метрика относится к метрикам сложности потока управления данных. Суть данного метода состоит в оценке прочности отдельно взятого программного модуля информационной системы с помощью анализа характера использования переменных из списка ввода-вывода. Для этого выделяется множество переменных, составляющих список ввода-вывода, которое в свою очередь разбивается на 4 функциональные группы:

– вводимые переменные, необходимые для расчетов и для обеспечения вывода (P);

– модифицируемые и создаваемые внутри программы переменные (M);

– управляющие переменные (C);

– «паразитные», не используемые в программе переменные (T).

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе.

Основная формула для расчета качества кода:

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T,$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы. По мнению автора метрики, наибольший вес, равный 3, имеет функциональная группа C, так как она влияет на поток управления программы.

Весовые коэффициенты остальных групп распределяются следующим образом: $a_1 = 1$, $a_2 = 2$, $a_4 = 0{,}5$. Весовой коэффициент группы T не равен 0, поскольку «паразитные» переменные не увеличивают сложность потока данных программы, но иногда затрудняют ее понимание. В итоге, с учетом весовых коэффициентов, формула имеет вид:

$$Q = P + 2M + 3C + 0{,}5T.$$

В результате вычисляется индекс Q – индекс удобства обслуживания программного модуля. Проведем расчеты по формулам приведенного выше метода для класса OilStorage:

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T$$

$$a_1 = 1;\ a_2 = 2;\ a_3 = 3;\ a_4 = 0{,}5$$

$$P\ (\text{вводимые переменные}) = 2$$

$$M\ (\text{модифицируемые переменные}) = 1$$

$$C\ (\text{управляющие переменные}) = 3$$

$$T\ (\text{«паразитные» переменные}) = 2$$

$$Q = 2 + 2 * 1 + 3 * 3 + 0{,}5 * 2 = 15.$$

Проанализируем полученные результаты. Результат вычисления метрики Чепина – сложности потока управления данных равной 15 говорит о невысокой сложности потока управления данных в классе OilStorage. Метрики, сгенерированные в среде разработки программного обеспечения Visual Studio 2022, говорят о том, что все классы соответствуют нормам качества программного кода.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано мультиплатформенное игровое приложение-стратегия с использованием среды разработки Unity.

Был проведен анализ существующих игровых приложений, аналогичных созданному программному средству, а также поиск основных механик присутствующих в играх выбранного игрового жанра. Также было проведено проектирование, позволившее определить окончательную структуру игрового приложения, а затем был спроектирован простой и удобный пользовательский интерфейс.

Программная игра реализована средствами мультиплатформенного движка Unity, языка программирования C#. Диаграммы необходимые для проектирования и реализации приложения реализованы средствами онлайн конструктора draw.io и программного средства Visual Paradigm CE 16.3.

На этапе контроля качества программного обеспечения было проведено функциональное и модульное тестирование. На основе анализа результатов тестирования можно сделать вывод о том, что разработанное игровое приложение и все входящие в его состав объекты работают корректно. Также был проведен анализ качества программного кода, показавший хороший результат.

В ходе выполнения работы также было выполнено планирование проекта. Выполнение проекта длилось 48 дней, а прогнозируемые затраты равны 125 940 рублям. Оценка сроков и бюджета говорит о том, что данная разработка экономически выгодна.

Таким образом, в результате выполнения выпускной квалификационной работы были разработаны программные средства, отвечающие всем требованиям технического задания.

Список использованных источников

1. Деятельность человека и ее основные формы [Электронный ресурс]. – URL: https://www.grandars.ru/shkola/bezopasnost – zhiznede\yatelnosti/formy – deyatelnosti – cheloveka.html (дата обращения 10.05.2022).

2. Роль игры в жизни человека [Электронный ресурс]. – URL: https://www.stud24.ru/psychology/rol-igry-v-zhizni-cheloveka/227604-666425-page1.html (дата обращения 10.05.2022).

3. Idle-игры: Почему разработчикам стоит обратить внимание на жанр «кликеров» [Электронный ресурс]. – URL: https://vc.ru/flood/8083 – idle# (дата обращения 10.05.2022).

4. Хокинг Д. Unity в действии. Мультиплатформенная разработка на C#. – 2-е межд. Издание. – Спб.: Питер, 2018. – 352 с.

5. Какая польза от стратегических игр? [Электронный ресурс]. – URL: https://telegraf.by/press-relizi/kakaya-polza-ot-strategicheskih-igr/# (дата обращения 10.05.2022).

6. Дж. Рамбо, М. Блаха. UML 2.0. Объектно-ориентированное моделирование и разработка. – Спб.: Питер, 2005. – 545 с.

7. Погружение в паттерны проектирования / [Электронная книга] / 398 стр. / А. Швец. – URL: https://refactoring.guru/ru/design-patterns/book/ (дата обращения 10.05.2022)

8. Паттерны: MVC, MVP и MVVM [Электронный ресурс]. – URL: https://www.outcoldman.com/ru/archive/ (дата обращения 24.05.2022)

9. Архитектурные шаблоны в ABAP: MVC, MVP, MVVM, MVA [Электронный ресурс]. – URL: http://abap4.ru/mvc-like-design-patterns-in-abap.html (дата обращения 10.05.2022)

10. Самый популярный движок в Steam [Электронный ресурс]. – URL: https://app2top.ru/analytics/samyj – populyarnyj – dvizhok – v – steam – unity – no – na – unreal – engine – delayut – samye – dorogie – igry – 189951.html (дата обращения 10.05.2022).

11. Выбор лучших: интегрированная среда для веб – разработки – Visual Studio [Электронный ресурс]. – URL: https://webformyself.com/sreda – dlya – veb – razrabotki – visual – studio/ (дата обращения 10.05.2022).

12. БД – хранилище объектов – Анатолий Тенцер [Электронный ресурс]. – URL: https://compress.ru/article.aspx?id=11515 (дата обращения 10.05.2022).

13. Б. Мейер. Объектно-ориентированное конструирование программных систем. – М.: Русская редакция, 2005. – 1204 с.

14. Обеспечение и контроль качества программного обеспечения [Электронный ресурс]. – URL: https://unetway.com/blog/obespecenie-i-kontrol-kacestva-programmnogo-obespecenia (дата обращения 07.06.2022).

15. Теория тестирования ПО просто и понятно [Электронный ресурс]. – URL: https://habr.com/ru/post/587620/ (дата обращения 07.06.2022).

16. Функциональное тестирование [Электронный ресурс]. – URL: https://www.appline.ru/services/testing/functionalnoe-testirovanie (дата обращения 07.06.2022).

17. Значения метрик кода [Электронный ресурс]. – URL: https://docs.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values (дата обращения 07.06.2022).

Приложение А

ГЛОССАРИЙ

(справочное)

Игровое приложение – компьютерная программа, служащая для организации игрового процесса, связи с партнерами по игре, или сама выступающая в качестве партнёра.

Мультиплатформенное игровое приложение – игровое приложение, имеющее несколько версий, доступных на разных устройствах и операционных системах (например, на смартфонах, компьютерах, игровых приставках и т.д.).

Игровой жанр – классификация игровых приложений.

Idle – игры (они же «кликеры» или инкрементальные игры) – игры (зачастую в жанре стратегия), целиком посвященные управлению потоками доходов. Как и симуляторы, они фокусируются на принятии решений, которые увеличивают доход.

Аркада – жанр компьютерных игр, характеризующийся коротким по времени, но интенсивным игровым процессом. В узком смысле аркадными называются игры для аркадных игровых автоматов.

RPG «Role—Playing Game» – знаменитый жанр компьютерных и видео игр, где основой игрового процесса является отогревание определенной роли. Игрок берёт под контроль определённого героя или героиню, с набором стандартных навыков, характеристик и умений. Играм этого жанра свойственны очень глубокие диалоги, нелинейное прохождение, использование продвинутого инвентаря и большие и открытые для исследования игровые пространства.

Система Free-to-Play – система монетизации и способ распространения компьютерных игр. Позволяет играть пользователю без обязательного внесения денежных средств.

Система Pay-to-Play – система монетизации и способ распространения компьютерных игр. Позволяет играть пользователю только после покупки копии игры.

Интерфейс – граница между двумя функциональными объектами, требования к которой определяются стандартом; совокупность средств, методов и правил взаимодействия между элементами системы.

UI – user interface – интерфейс пользователя – интерфейс, обеспечивающий передачу информации между пользователем, человеком и программно-аппаратными компонентами компьютерной системы.

UX – дизайн взаимодействия с пользователем. Включает в себя традиционное взаимодействие человека с компьютером, в том числе все аспекты продукта, как они воспринимаются пользователями.

Сервер — это мощный компьютер, предназначенный для хранения информации и обеспечения доступа к ней с удалённых клиентских устройств. Тип данных, которые хранятся на сервере, зависит от его вида и назначения.

Нефтехранилище (нефтебаза, нефтетерминал) – промышленный объект для хранения нефти и нефтепродуктов. Нефтехранилище – сложный составной объект, требующий тщательного контроля и своевременной реакции на изменения в его работе.

Резервуар – происходит от слова «резерв» – герметично закрываемый или открытый, стационарный сосуд, наполняемый жидким или газообразным веществом.

Насос – гидравлическая машина, преобразующая механическую энергию приводного двигателя или мускульную энергию в энергию потока жидкости, служащую для перемещения и создания напора жидкостей всех видов.

Программное обеспечение – программа или множество программ, используемых для управления компьютером.

Информационная система – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы, которые обеспечивают и распространяют информацию.

Программное средство – объект, состоящий из программ, процедур, правил, а также, если предусмотрено, сопутствующих им документации и данных, относящихся к функционированию системы обработки информации.

Среда разработки – комплекс программных средств, используемый программистами для разработки программного обеспечения. Среда разработки включает в себя: текстовый редактор, Транслятор, средства автоматизации сборки, отладчик.

Компонентно-ориентированное программирование – парадигма программирования, существенным образом опирающаяся на понятие компонента – независимого модуля исходного кода программы, предназначенного для повторного использования и развёртывания и реализующегося в виде множества языковых конструкций (например, «классов» в объектно-ориентированных языках программирования), объединенных по общему признаку и организованных в соответствии с определёнными правилами и ограничениями.

Операционная система – комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Паттерн – устойчивая конструкция в программировании.

Контент – информация, наполняющая какую-либо информационную систему.

Игровой контент – набор механик, интерфейсов, задач предоставленный пользователю в рамках игровой ситуации.

Универсальная модель данных – неизменная (стандартная) для разных предметных областей схема базы данных в реляционной системе управления базами данных. Для универсальных моделей баз данных определен перечень понятий, которые могут быть использованы для описания множества данных, операций с данными, а также набора ограничений целостности данных.

Виртуальный хостинг – вид хостинга, при котором множество веб-сайтов расположено на одном веб-сервере. Это самый экономичный вид хостинга, подходящий для небольших проектов.

Приложение Б

КОД ПРОГРАММЫ

(рекомендуемое)

Листинг Б.1 – код основного класса Presenter

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using static Assets.Scripts.SaveZone;

namespace Assets.Scripts
{
    /// <summary>
    /// Класс, отвечающий за одну из важнейших частей
    /// архитектуры MVP
    /// </summary>
    public class Presenter : MonoBehaviour
    {
        #region Поля класса

        #region TankView(Текущий резервуар)

        public GameObject TankViewPanel;
        public GameObject InputOnButton;
        public GameObject OutputOnButton;
        public Slider TankSlider;
        public Text Value;
        public Text ActiveTankText;
        public Text MyMoney;
        public Text PriceToBuy;
        public Text SideBonusText;
        public GameObject Notification;
        public Text NotificationText;
        [HideInInspector] public Int32 SideBonus = 0;

        public Int32 IndexActiveTank { get; set; } = 0;

        #endregion

        #region Panels (Основные окна игры)

        public GameObject PanelTankList;
        public GameObject SideBuildingPanel;

        #endregion

        #region GameRules (Ключевые поля для игры)

        public OilStorage MyOilStorage { get; set; }

        public Double Money { get; set; } = 1000;

        public Double PriceToBuyOil { get; set; } = 20;
```

```
        public List<IBuilding> Buildings { get; set; }

        private SaveGameController SaveController { get; set; }
= new SaveGameController();

        #endregion

        #endregion

        #region Методы класса

        #region public

        /// <summary>
        /// Метод вычисляющий доход нефтехранилища
        /// (в том числе и отрицательный)
        /// </summary>
        /// <param name="delta">Изменение количества
нефтепродуктов за единицу времени</param>
        public void GettingMoney(Double delta)
        {
            Int32 bonus = SideBonus;
            if(delta < 0) bonus = -bonus;
            Money += delta * (PriceToBuyOil + bonus);
        }

        /// <summary>
        /// После нажатия на кнопку "Закачки" нефтепродуктов в
резервуар
        /// </summary>
        public void OnInput()
        {
            if (MyOilStorage.Tanks[IndexActiveTank].MyManager ==
null)
                MyOilStorage.Tanks[IndexActiveTank].SwitchActive
InputPump();
            else
                ShowNotification("Данный резервуар под контролем
менеджера");
        }

        /// <summary>
        /// После нажатия на кнопку "Откачки" нефтепродуктов в
резервуар
        /// </summary>
        public void OnOutput()
        {
            if (MyOilStorage.Tanks[IndexActiveTank].MyManager ==
null)
                MyOilStorage.Tanks[IndexActiveTank].SwitchActive
OutputPump();
            else
```

```
                ShowNotification("Данный резервуар под контролем
менеджера");
        }

        /// <summary>
        /// Открытие/Закрытие окна с выбором резервуара
        /// </summary>
        public void ToogleTankListPanel()
        {
            PanelTankList.SetActive(!PanelTankList.activeSelf);
            TankViewPanel.SetActive(!TankViewPanel.activeSelf);
        }

        public void ToogleSideListPanel()
        {
            SideBuildingPanel.SetActive(!SideBuildingPanel.activ
eSelf);
            TankViewPanel.SetActive(!TankViewPanel.activeSelf);
        }

        /// <summary>
        /// Установление нового активного резервуара
        /// </summary>
        /// <param name="id">Идентификатор нового активного
резервуара</param>
        public void NewTankSelected(Int32 id)
        {
            Int32 tmp = IndexActiveTank;
            IndexActiveTank = id;
            ToogleTankListPanel();
        }

        public void ShowNotification(String message)
        {
            if (Notification.activeSelf != true)
            {
                StartCoroutine(ShowMessage(message));
            }
        }

        #endregion

        #region private

        /// <summary>
        /// Метод, который вызывается приложением
        /// еще до отрисовки первого кадра сцены
        /// </summary>
        private void Awake()
        {
            SaveData data = SaveController.LoadGame();
            if (data != null)
            {
```

```
                MyOilStorage = new OilStorage(this,
data.TanksCount);
                Money = data.MoneyBalance;
                IndexActiveTank = data.IndexActiveTank;
                for (Int32 i = 0; i < data.TanksCount; i++)
                {
                    MyOilStorage.Tanks[i].Value =
data.TanksCurrentValue[i];
                    MyOilStorage.Tanks[i].MaxValue =
data.TanksMaxValue[i];
                    MyOilStorage.Tanks[i].MinValue =
data.TanksMinValue[i];
                    MyOilStorage.Tanks[i].InputPump.PumpingSpeed
= data.TanksInputSpeed[i];
                    MyOilStorage.Tanks[i].OutputPump.PumpingSpee
d = data.TanksOutputSpeed[i];
                    MyOilStorage.Tanks[i].InputPump.IsOn =
data.TanksInputPumpStatus[i];
                    MyOilStorage.Tanks[i].OutputPump.IsOn =
data.TanksOutputPumpStatus[i];
                    if (data.ManagerLevel[i] != 0)
                    {
                        MyOilStorage.Tanks[i].MyManager = new
OilManager(MyOilStorage.Tanks[i]);
                    }
                }
            }
            else
            {
                StartNewGame();

            }
        }

        /// <summary>
        /// Метод, который будет вызван приложением
        /// при отрисовке первого кадра сцены
        /// </summary>
        private void Start()
        {
            InvokeRepeating("GettingOil", 0, 1.0f);
            InvokeRepeating("PricesChange", 0, 3.0f);
            InvokeRepeating("CheckBudget", 0.0f, 0.5f);
        }

        /// <summary>
        /// Метод, который будет вызван приложением
        /// при отрисовке каждого кадры сцены
        /// </summary>
        private void Update()
        {
            if (TankViewPanel.activeSelf)
            {
                MyMoney.text = Money.ToString("### ###, ###");
```

```
                PriceToBuy.text = "Цена нефти: " +
PriceToBuyOil.ToString("0");
                SideBonusText.text = "Бонус: "
                    + SideBonus.ToString("0");
                Double max =
MyOilStorage.Tanks[IndexActiveTank].MaxValue;
                Double min =
MyOilStorage.Tanks[IndexActiveTank].MinValue;
                Double tmp =
MyOilStorage.Tanks[IndexActiveTank].Value;
                TankSlider.maxValue = (float)Math.Abs(max);
                TankSlider.minValue = (float)Math.Abs(min);
                TankSlider.value = (float)Math.Abs(tmp);
                Value.text = "Текущее значение: " + tmp;
                ActiveTankText.text = "Резервуар №"
+ (IndexActiveTank + 1);

                if (tmp == max &
MyOilStorage.Tanks[IndexActiveTank].InputPump.IsOn)
                {
                    OnInput();
                }

                if (tmp == min &
MyOilStorage.Tanks[IndexActiveTank].OutputPump.IsOn)
                {
                    OnOutput();
                }
            }

            if (Input.GetKey("escape"))  // если нажата клавиша
Esc
            {
                Application.Quit();    // закрыть приложение
            }
        }

        /// <summary>
        /// Метод, вызываемый приложением при выходе из него
        /// </summary>
        private void OnApplicationQuit()
        {
            SaveController.SaveGame(this);
        }

        /// <summary>
        /// Метод уведомляющий основной класс
        /// модели (Model из архитектуры MVP)
        /// OilStorage нужно совершить очередную
        /// проверку на получение/трату нефти
        /// из резервуаров нефтехранилища
        /// </summary>
        private void GettingOil()
        {
```

```
            MyOilStorage.AddingOil();
        }

        /// <summary>
        /// Метод изменяющий цену в определенном диапазоне
        /// </summary>
        private void PricesChange()
        {
            PriceToBuyOil = 20;
            System.Random rnd = new System.Random();
            Int32 value = rnd.Next(-15, 15);
            PriceToBuyOil += value;
        }

        /// <summary>
        /// Метод «спасалка» для неудачливых игроков. В будущем
будет изменен на начало новой игры!!!
        /// </summary>
        private void CheckBudget()
        {
            if (Money <= PriceToBuyOil && MyOilStorage.OilCount
== 0)
            {
                Money = 100;
            }
        }

        /// <summary>
        /// Метод задающий необходимые значения для начала новой
игры
        /// </summary>
        private void StartNewGame()
        {
            MyOilStorage = new OilStorage(this);
        }

        private IEnumerator ShowMessage(String message)
        {
            NotificationText.text = message;
            Notification.SetActive(true);
            yield return new WaitForSeconds(2f);
            Notification.SetActive(false);
        }

        #endregion

        #endregion
    }
}
```

Листинг Б.2 – класс OilStorage (нефтехранилище) как часть Model

```
using System;
using System.Collections.Generic;

namespace Assets.Scripts
{
    /// <summary>
    /// Нефтехранилище. Основной класс Model (часть архитектуры
MVP)
    /// </summary>
    public class OilStorage
    {
        #region Поля класса

        List<ITank> tanks = new List<ITank>();
        Presenter myPresenter;

        public List<ITank> Tanks { get => tanks; set => tanks =
value; }

        public Double OilCount
        {
            get
            {
                Double result = 0;
                foreach (OilTank tank in tanks)
                {
                    result += tank.Value;
                }
                return result;
            }
        }

        #endregion

        #region Методы класса

        public OilStorage(Presenter my, Int32 Count = 1)
        {
            myPresenter = my;
            for (Int32 i = 0; i < Count; i++)
            {
                OilTank tmp = new OilTank();
                tanks.Add(tmp);
            }
        }

        /// <summary>
        /// Метод просчитывающий количество
        /// нефти полученной в каждый резервуар
        /// и количество нефти проданной из резервуаров
```

```
        /// </summary>
        public void AddingOil()
        {
            foreach (OilTank elem in Tanks)
            {
                if (elem.MinValue > elem.Value)
                {
                    elem.Value = elem.MinValue;
                }

                if (elem.MaxValue < elem.Value)
                {
                    elem.Value = elem.MaxValue;
                }

                if (elem.MyManager != null)
                {
                    elem.MyManager.Work(myPresenter.PriceToBuyOi
l);
                }

                if (elem.IsOn)
                {
                    if (elem.InputPump.IsOn)
                    {
                        if (myPresenter.Money -
(elem.InputPump.PumpingSpeed *
                            (myPresenter.PriceToBuyOil -
myPresenter.SideBonus)) >= 0)
                        {
                            if (elem.Value +
elem.InputPump.PumpingSpeed >= elem.MaxValue)
                            {
                                myPresenter.GettingMoney(elem.Va
lue - elem.MaxValue);
                                elem.Value = elem.MaxValue;
                            }
                            else
                            {
                                myPresenter.GettingMoney(-
elem.InputPump.PumpingSpeed);
                                elem.Value +=
elem.InputPump.PumpingSpeed;
                            }
                        }
                        else
                        {
                            //myPresenter.InputOnButton.SetActiv
e(true);
                            //myPresenter.InputOffButton.SetActi
ve(false);
                            elem.InputPump.Off();
                        }
```

```
                    }

                    if (elem.OutputPump.IsOn)
                    {

                        if (elem.Value -
elem.OutputPump.PumpingSpeed <= elem.MinValue)
                        {
                            myPresenter.GettingMoney(elem.Value
- elem.MinValue);
                            elem.Value = elem.MinValue;
                        }
                        else
                        {
                            myPresenter.GettingMoney(elem.Output
Pump.PumpingSpeed);
                            elem.Value -=
elem.OutputPump.PumpingSpeed;
                        }
                    }
                }
            }
        }

        #endregion
    }
}
```

Листинг Б.3 – класс OilTank (резервуар) как часть Model

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.UI;

namespace Assets.Scripts
{
    /// <summary>
    /// Резервуар. Класс входящий в Model
    /// </summary>
    public class OilTank : ITank
    {
        #region Поля класса

        String type = "Oil";
        Pump inputPump;
        Pump outputPump;
        Boolean isOn = true;
        Double value;
```

```
        Double maxV;
        Double minV;
        OilManager Manager;

        public String Type { get => type; }
        public Pump InputPump { get => inputPump; set =>
inputPump = value; }
        public Pump OutputPump { get => outputPump; set =>
outputPump = value; }
        public double Value { get => value; set => this.value =
value; }
        public double MaxValue { get => maxV; set => maxV =
value; }
        public double MinValue { get => minV; set => minV =
value; }
        public bool IsOn { get => isOn; set => isOn = value; }
        public OilManager MyManager { get => Manager; set =>
Manager = value; }

        #endregion

        #region Методы класса

        #region public

        public OilTank(Double startValue = 0, Double minValue =
0D, Double maxValue = 100D)
        {
            value = startValue;
            minV = minValue;
            maxV = maxValue;
            inputPump = new Pump();
            outputPump = new Pump();
        }

        public void On()
        {
            IsOn = true;
        }

        public void Off()
        {
            IsOn = false;
            inputPump.Off();
            outputPump.Off();
        }

        public void SwitchActiveInputPump()
        {
            SwitchActivePump(this.InputPump);
        }

        public void SwitchActiveOutputPump()
        {
```

```
            SwitchActivePump(this.OutputPump);
        }

        #endregion

        #region private

        /// <summary>
        /// Метод меняющий состояние переданного в качестве
   /// параметра резервуара
        /// </summary>
        /// <param name="thisPump"></param>
        private void SwitchActivePump(Pump thisPump)
        {
            if (thisPump.IsOn)
            {
                thisPump.Off();
            }
            else
            {
                thisPump.On();
            }
        }
        #endregion
        #endregion
    }
}
```

Листинг Б.4 – класс Pump (насос) как часть Model

```
using System;

namespace Assets.Scripts
{
    public class Pump
    {
        #region Поля класса

        Boolean isOn = false;
        Double pumpingSpeed;
        Int32 speedLevel = 1;

        public Double PumpingSpeed { get => pumpingSpeed; set =>
pumpingSpeed = value; }
        public Int32 SpeedLevel { get => speedLevel; set =>
speedLevel = value; }
        public Boolean IsOn { get => isOn; set => isOn = value;
}
        #endregion
        #region Методы класса
        public Pump(Double pumpingSpeed = 1D)
        {
```

```
            this.PumpingSpeed = pumpingSpeed;
        }
        public void On()
        {
            IsOn = true;
        }
        public void Off()
        {
            IsOn = false;
        }
        #endregion
    }
}
```

Листинг Б.5 – класс LoginController

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text.RegularExpressions;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

namespace Assets.Scripts
{
     public class LoginController : MonoBehaviour
     {
        #region Поля класса

        [SerializeField] private InputField userName;
          [SerializeField] private InputField userPass;
          [SerializeField] private Text messageText;
          [SerializeField] private Button register;
          [SerializeField] private Button login;
          [SerializeField]
          private String loginURL =
               "https://oil-storage-system-
001.herokuapp.com/login.php";
          [SerializeField]
          private String registerURL =
               "https://oil-storage-system-
001.herokuapp.com/register.php";
          [SerializeField] private SceneChangerController
sceneChanger;

        #endregion

        #region Методы класса

        /// <summary>
```

```
        /// Метод, который вызывается приложением
        /// еще до отрисовки первого кадра сцены
        /// </summary>
        private void Awake()
          {
               userPass.contentType =
InputField.ContentType.Password;
               register.onClick.AddListener(() => { Register();
});
               login.onClick.AddListener(() => { Login(); });
          }

          /// <summary>
          /// Валидация имени и пароля
          /// </summary>
          /// <param name="value"></param>
          /// <param name="min"></param>
          /// <param name="max"></param>
          /// <param name="field"></param>
          /// <returns></returns>
          private Boolean IsValid(String value, Int32 min, Int32
max, String field)
          {
               if (value.Length < min)
               {
                    Message("В поле [ " + field + " ]
недостаточно символов, нужно минимум [ " + min + " ]");
                    return false;
               }
               else if (value.Length > max)
               {
                    Message("В поле [ " + field + " ] допустимый
максимум символов, не более [ " + max + " ]");
                    return false;
               }
               else if (Regex.IsMatch(value, @"[^\w\.@-]"))
               {
                    Message("В поле [ " + field + " ]
содержаться недопустимые символы.");
                    return false;
               }

               return true;
          }

          /// <summary>
          /// Метод создающий сообщение для log
          /// </summary>
          /// <param name="text"></param>
          private void Message(String text)
          {
               messageText.text = text;
               Debug.Log(this + " --> " + text);
          }
```

```
        /// <summary>
        /// Метод, выполняющий вход в игру по логину и паролю
введенных пользователем в соответствующие поля
        /// </summary>
        private void Login()
        {
            if (!IsValid(userName.text, 3, 15, "Имя") ||
!IsValid(userPass.text, 6, 20, "Пароль")) return;

            WWWForm form = new WWWForm();
            form.AddField("name", userName.text);
            form.AddField("password", userPass.text);
            WWW www = new WWW(loginURL, form);
            StartCoroutine(LoginFunc(www));
        }

        /// <summary>
        /// Метод, выполняющий регистрацию нового пользователя
        /// </summary>
        private void Register()
        {
            if (!IsValid(userName.text, 3, 15, "Имя") ||
!IsValid(userPass.text, 6, 20, "Пароль")) return;

            WWWForm form = new WWWForm();
            form.AddField("name", userName.text);
            form.AddField("password", userPass.text);
            //form.AddField("email", userEmail.text);
            WWW www = new WWW(registerURL, form);
            StartCoroutine(RegisterFunc(www));
        }

        /// <summary>
        /// Метод, в котором приложение получает ответ на
запрос о входе в игру
        /// </summary>
        /// <param name="www"></param>
        /// <returns></returns>
        private IEnumerator LoginFunc(WWW www)
        {
            yield return www;

            if (www.error == null)
            {
                if (string.Compare(www.text, "Success!") ==
0) // получаем в ответе слово-ключ из файла login.php
                {
                    Message("Успешный вход!");

                    sceneChanger.ChangeScene("MainScene");
                }
                else
                {
```

```
                    Message(www.text);
                }
            }
            else
            {
                Message("Error: " + www.error);
            }
        }

        /// <summary>
        /// Метод, в котором приложение получает ответ на
запрос о регистрации нового пользователя
        /// </summary>
        /// <param name="www"></param>
        /// <returns></returns>
        private IEnumerator RegisterFunc(WWW www)
        {
            yield return www;

            if (www.error == null)
            {
                Message("Пользователь успешно добавлен в
базу.");
            }
            else
            {
                Message("Error: " + www.error);
            }
        }

    #endregion
}
}
```

Листинг Б.6 – скрипт login.php

```
<?php
    $name = $_POST['name'];
    $pass = $_POST['password'];

    require_once('index.php');

    $check = pg_query_params($db, "
    SELECT Login.\"Value\", Pass.\"Value\"
        FROM \"Objects\" O
            INNER JOIN \"Strings\" Login ON O.\"Id\" =
Login.\"ObjectId\"
            INNER JOIN \"StrDesc\" LoginType ON Login.\"TypeId\"
= LoginType.\"Id\"
            INNER JOIN \"Strings\" Pass ON O.\"Id\" =
Pass.\"ObjectId\"
```

```
            INNER JOIN \"StrDesc\" PassType ON Pass.\"TypeId\" =
PassType.\"Id\"
        WHERE LoginType.\"Code\" = 'Login' AND
            PassType.\"Code\" = 'Password' AND
            Login.\"Value\" = $1", array($name));

    $numrows = pg_num_rows($check);

    if ($numrows == 0)
    {
        die ("Пользователь не зарегистрирован!");
    }
    else
    {
        //$pass = md5($pass);
        while($row = pg_fetch_row($check))
        {
            if ($pass == $row[1])
                die("Успешный вход!");
            else
                die("Неверный пароль");
        }
    }
?>
```

Листинг Б.7 – скрипт register.php

```
<?php
    $name = $_POST['name'];
    $pass = $_POST['password'];
    //$email = $_POST['email'];

    require_once('index.php');

    $check = pg_query_params($db,
    "SELECT O.\"ItemName\" FROM \"Objects\" O
        INNER JOIN \"ObjType\" OT ON OT.\"Id\" = O.\"TypeId\"
    WHERE O.\"ItemName\" = $1 AND
        OT.\"Code\" = 'Player';", array($name));

    $numrows = pg_num_rows($check);

    if ($numrows == 0)
    {
        //$pass = md5($pass);

        $ins = pg_query_params($db,
            "INSERT INTO \"Objects\" VALUES(default, '2', $1);",
        array($name));

        if (!$ins)
            die ("Error: " . pg_error());
```

```
        $check = pg_query_params($db,
            "SELECT O.\"Id\" FROM \"Objects\" O
                INNER JOIN \"ObjType\" OT ON OT.\"Id\" =
O.\"TypeId\"
            WHERE O.\"ItemName\" = $1 AND
                OT.\"Code\" = 'Player';", array($name));
        $result = pg_fetch_row($check);
        $idNewPlayer = $result[0];

        $ins = pg_query_params($db,
            "INSERT INTO \"Strings\" VALUES(default, '2', $1,
$2);",
        array($idNewPlayer, $name));

        if (!$ins)
            die ("Error: " . pg_error());

        $ins = pg_query_params($db,
            "INSERT INTO \"Strings\" VALUES(default, '3', $1,
$2);",
        array($idNewPlayer, $pass));

        if (!$ins)
            die ("Error: " . pg_error());

        $ins = pg_query_params($db,
            "INSERT INTO \"Properties\" VALUES(default, '2', $1,
'100');",
        array($idNewPlayer));

        if (!$ins)
            die ("Error: " . pg_error());


        die("Player created!");
    }
    else
    {
        die("User allready exists!");
    }
?>
```

Приложение В

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

(рекомендуемое)

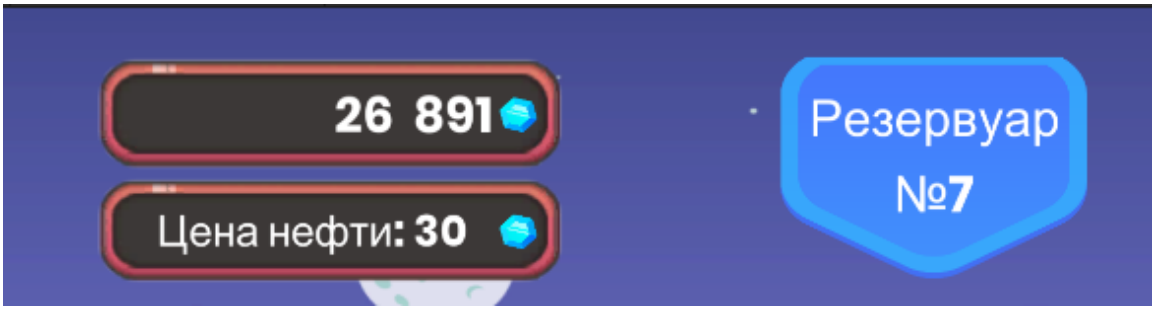

Рисунок В.1 – Отображение цены на экране (слева снизу)

Рисунок В.2 – Отображение состояния резервуара

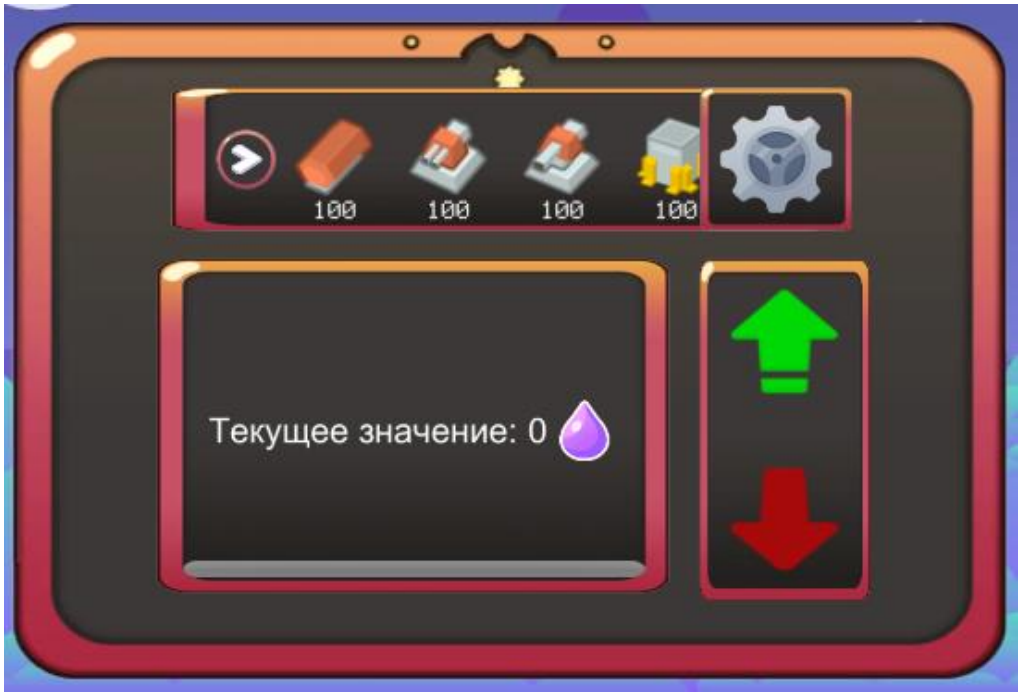

Рисунок В.3 – Активация насоса закачки (до нажатия)

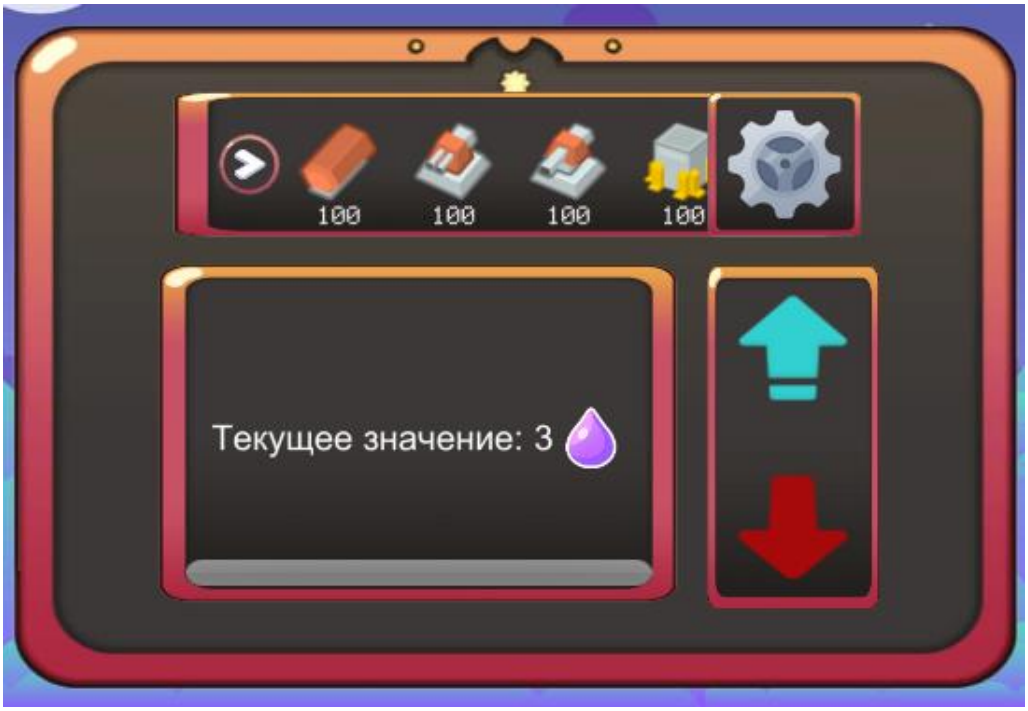

Рисунок В.4 – Активация насоса закачки (после нажатия)

Рисунок В.5 – Активация насоса откачки (до нажатия)

Рисунок В.6 – Активация насоса откачки (после нажатия)

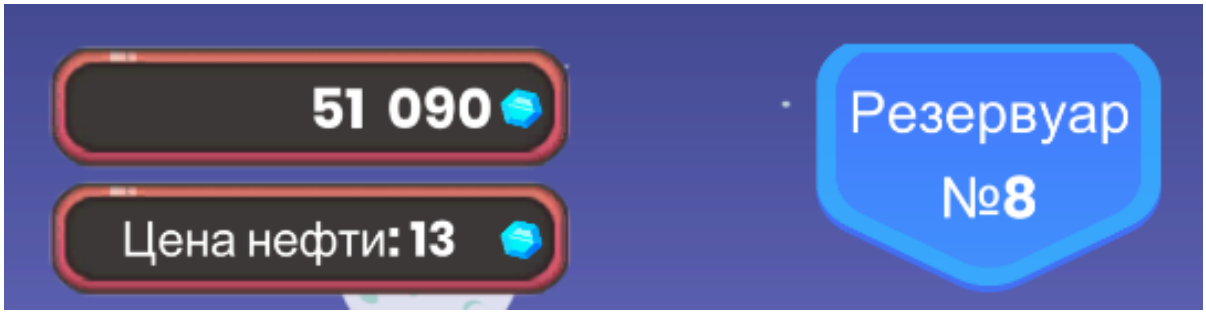

Рисунок В.7 – Отображение бюджета нефтехранилища (слева сверху)

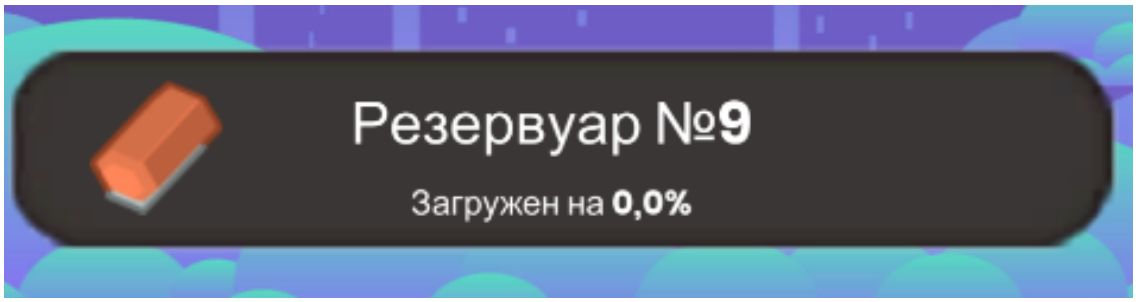

Рисунок В.8 – Отображение купленного резервуара (если средств достаточно)

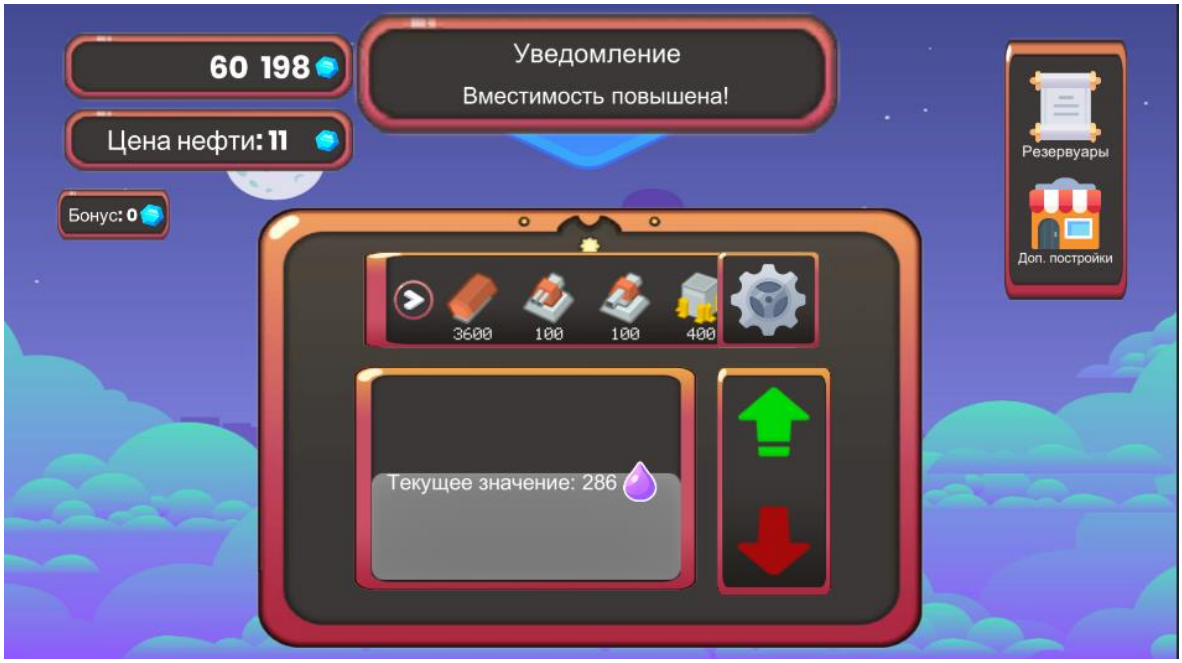

Рисунок В.9 – Увеличить объем резервуара (если средств достаточно)

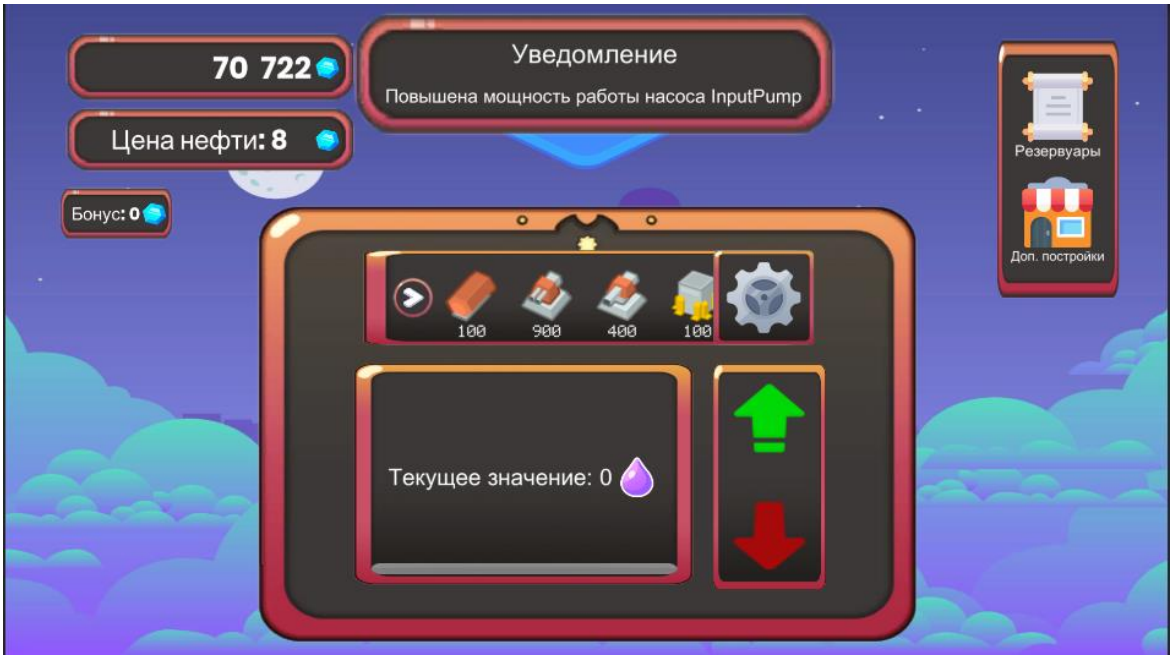

Рисунок В.10 – Улучшить насос закачки резервуара

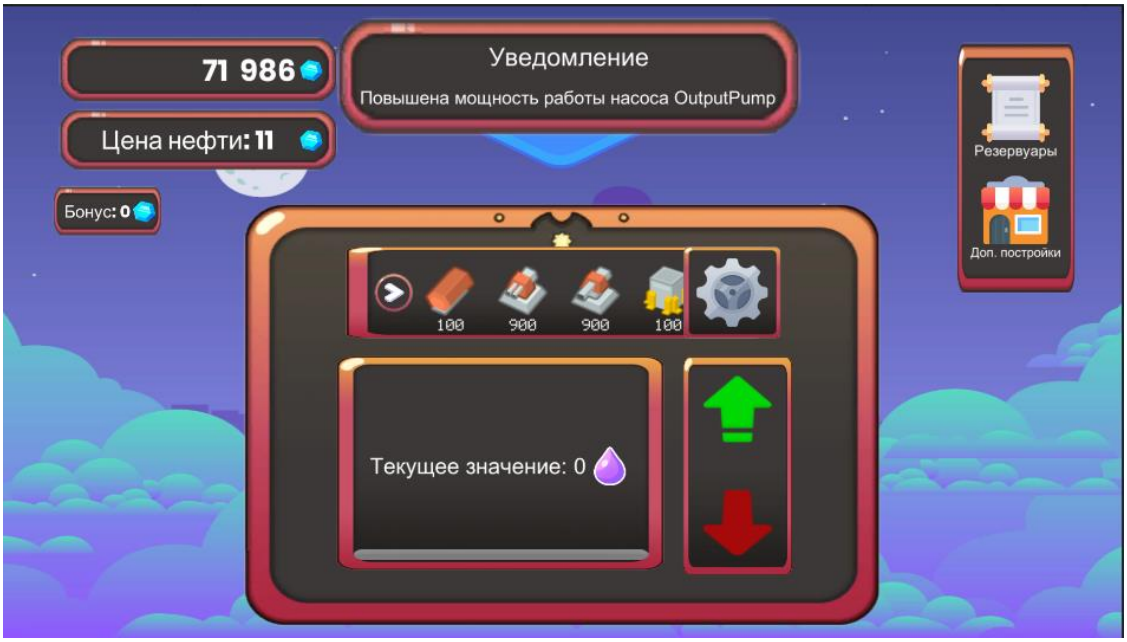

Рисунок В.11 – Улучшить насос откачки резервуара

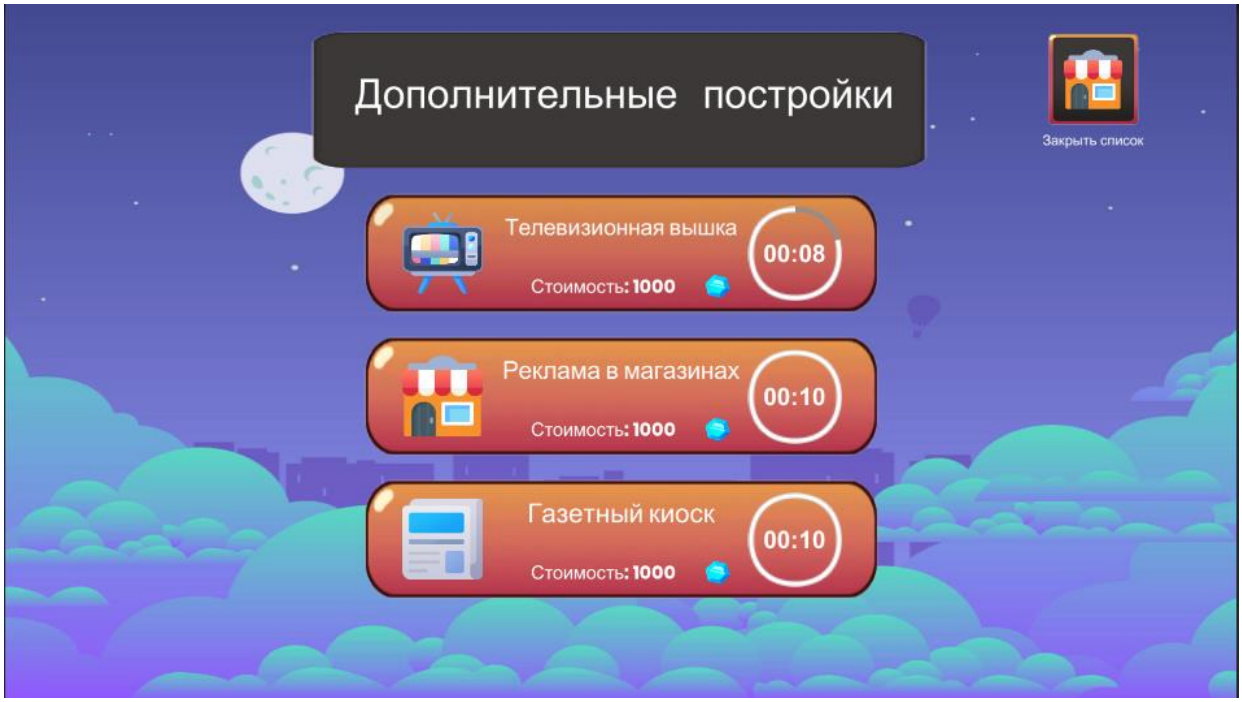

Рисунок В.12 – Покупка дополнительного здания

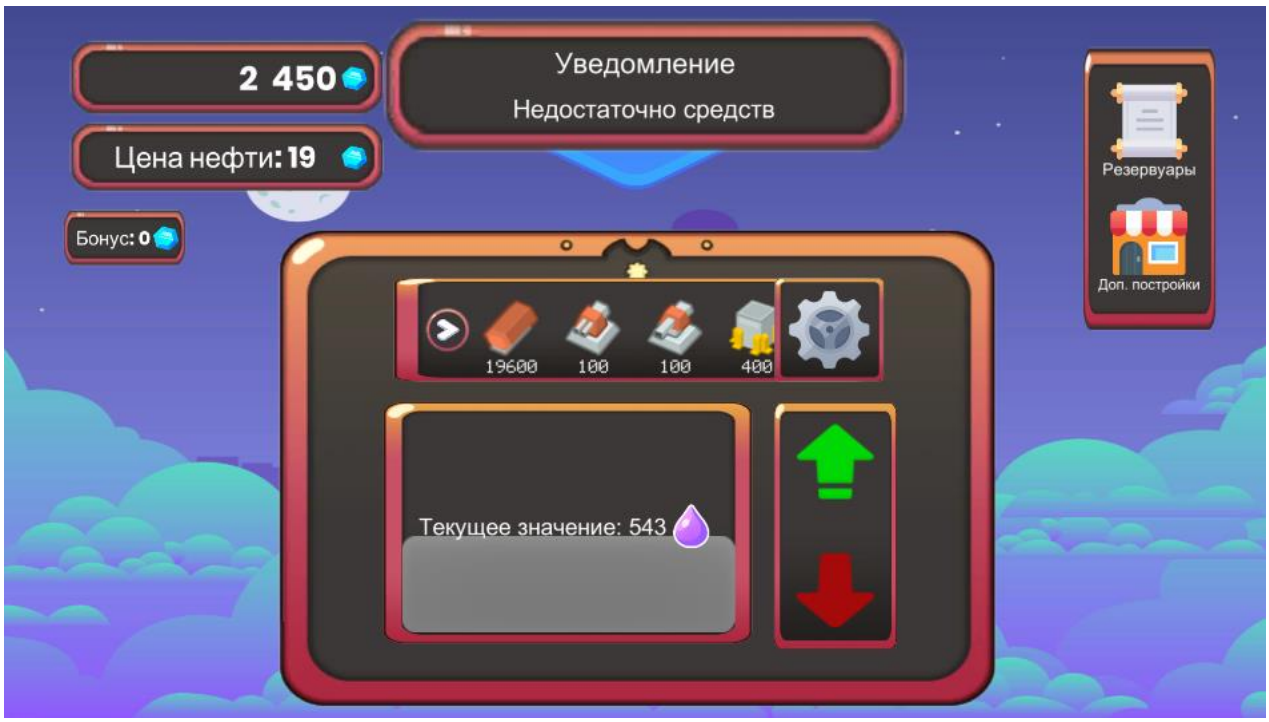

Рисунок В.13 – Недостаток средств на покупку улучшения