**Введение**

На курсовое проектирование была поставлена задача разработать программу на тему «Разработка игрового приложения в жанре платформер «PlatformGuy»

Цель курсового проекта заключается в разработке игрового приложения в жанре платформер «PlatformGuy» для развития пространственного мышления, а так же для развлечения и отдыха. В этом игровом приложении игрок сможет сыграть в увлекательную игру и поставить свой собственный рекорд.

В игре будут задействованы различные механики, такие как:прыжок, появление платформ, подсчет очков,сохранение рекорда, остановка платформ и другие.

Поставленная задача актуальна, так как правила игры легки для понимания и не нужно тратить много времени на их осваивания.

Описание разделов пояснительной записки:

Раздел «Постановка задачи» содержит описание предметной области, инструментов разработки и требований к приложению.

В разделе «Проектирование задачи» рассмотрен выбор стратегии разработки и модели ЖЦ, описываются инструменты разработки и UML-диаграммы, а также представлены прототипы страниц сайта.

«Реализация» – это раздел, в котором описываются все функции и элементы управления, а также спецификация проекта.

В разделе «Тестирование» приведено описание тестирования данного ресурса, т.е. протестирована каждая операция, которая выполняется приложением, а также пользовательский интерфейс.

В разделе «Руководство пользователя» приведена инструкция по использованию игрового приложения.

В разделе «Список использованных источников» будет приведён список используемых при разработке источников.

В «Приложении А» будет приведена диаграмма вариантов использования.

В «Приложении Б» будет приведена модель данных.

В «Приложении В» будет приведена диаграмма классов.

В «Приложении Г» будет приведена диаграмма обьектов.

В «Приложении Д» будут приведен UI интерфейс.

В «Приложении Е» будут приведен UX интерфейс.

В «Приложении Ж» будет приведена диаграмма Ганта.

В «Приложении З» будет приведен листинг программы.

**1 Анализ задачи**

**1.1 Постановка задачи**

**1.1.1 Организационно-экономическая сущность задачи:**

1. Наименование задачи: Разработка игрового приложения в жанре платформер “PlatformGuy”

2. Цель разработки: игрового приложения в жанре платформер «PlatformGuy» для развития пространственного мышления, а так же для развлечения и отдыха

3. Назначение: Игровое приложение предназначено для любых пользователей желающих использовать данное приложение.

4. Периодичность использования ПП: ПП может использоваться ежедневно, в зависимости от потребностей пользователя.

5. Источники и способы получения данных: Будут получаться с помощью файлов приложения.

6. Информационная связь с другими задачами: будет реализовываться внутри скриптов.

7. Обзор существующих аналогичных ПП: Существуют различные

Игры в жанре платформера ,такие как: Doodle Jump ,Super meat boy,mario и другие. Каждый из них имеет свои особенности и функциональность, которая может быть использована в качестве примера при разработке собственного игрового приложения.

**1.1.2 Функциональные требования**

Функциональные требования игрового приложения должны удовлетворять потребности и требования пользователей.

1. Функции пользователей:

* Запуск игры
* Управление персонажем
* Просмотр очков
* Сохранение рекорда
* Просмотр правил игры
* Повтор игры

**1.1.3 Описание исходной (входной) информации:**

Для создания игрового приложения необходима следующая исходная

информация:

1.Очки за преодоленные платформы

2.Управление персонажем в игре

**1.1.4 Описание результатной (выходной) информации:**

Правила игры

Спрайты элементов игры

Состояние игровых предметов, такие как: скорость движения платформ, сила притяжения персонажа, расстояние межу для платформ и другие.

**1.1.5 Описание используемой условно-постоянной информации:**

Условно-постоянная информация - это информация, которая не изменяется в процессе использования программного продукта и является необходимой для его работы. К такой информации можно отнести различные классификаторы, коды, справочники и т.д.

Примеры условно-постоянной информации для игрового приложения:

1, Скрипты с кодом

2. ”Хитбоксы” элементов

**1.1.6 Нефункциональные (эксплуатационные) требования**

1.Требования к применению: пользовательский интерфейс должен быть

интуитивно понятным и удобным в использовании для всех категорий пользователей, документация должна быть подробной и доступной.

2.Требования к производительности: система должна обеспечивать высокую производительность и быстродействие.

3.Требования к реализации: система должна быть разработана с использованием современных технологий и стандартов, язык программирования – C#, движок-Unity, операционная система -Windows.

4.Требования к надежности: система должна быть высоконадежной, устойчивой к сбоям и отказам.

5.Требования к интерфейсу: система должна иметь удобный и интуитивно понятный интерфейс, обеспечивающий эффективное твзаимодействие пользователей с системой.

**1.2 Инструменты разработки**

Для разработки данного проекта будет выбрана среда разработки Unity, которая является одной из наиболее актуальных сред для создания приложений данного типа.

Разработка будет производится на языке программирования C#. Иные инструменты, используемые при разработке и написании сопутствующей документации:

1)WEB-ресурс DRAW.IO – будет использоваться для создания графической части и разработки UML-диаграмм;

2) Microsoft Office Word – для написания документации к программному продукту;

3)Craftpix.net-для поиска и использования бесплатных спрайтов для игрового приложения.

Разработка проекта будет происходить на компьютере со следующими параметрами:

* процессор  AMD Ryzen 5 5500
* объем оперативной памяти 16 GB;
* объем места на SSD: 1000 GB;
* видеокарта GeForce RTX 3060 with 12 GB VRAM;
* ОС Windows 10 Pro.

Для разработки плана над проектом был использован Microsoft Excel.

**1.3 Выбор стратегии разработки и модели жизненного цикла**

Для разработки игрового приложения «PlatformGuy» следует выбрать стратегию разработки и модель жизненного цикла. Осуществляем выбор посредством составления таблиц:

Таблица 1 – Выбор модели жизненного цикла на основе характеристик требований

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Являются ли требования к проекту легко определимыми и реализуемыми? | Да | Да | Да | Нет | Нет | Нет |
| 2. | Могут ли требования быть сформулированы в начале ЖЦ? | Да | Да | Да | Да | Нет | Нет |
| 3. | Часто ли будут изменяться требования на протяжении ЖЦ? | Нет | Нет | Нет | Нет | Да | Да |
| 4. | Нужно ли демонстрировать требования с целью их определения? | Нет | Нет | Да | Нет | Да | Да |
| 5. | Требуется ли проверка концепции программного средства или системы? | Нет | Нет | Да | Нет | Да | Да |
| 6. | Будут ли требования изменяться или уточняться с ростом сложности системы (программного средства) в ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Нужно ли реализовать основные требования на ранних этапах разработки? | Нет | Нет | Да | Да | Да | Да |

Вычисления: 4 за каскадную, 4 за V- образную, 5 за RAD, 5 за инкрементную, 3 за быстрого прототипирования и 3 за эволюционную.

Итог: На основе результатов заполнения табл. 3 подходящей является RAD модель и инкрементная модель.

Таблица 2 – Выбор модели жизненного цикла на основе характеристик команды разработчиков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории команды разработчиков  проекта | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Являются ли проблемы предметной области проекта новыми для большинства разработчиков? | Нет | Нет | Нет | Нет | Да | Да |
| 2. | Являются ли инструментальные средства, используемые в проекте, новыми для большинства разработчиков? | Да | Да | Нет | Нет | Нет | Да |
| № критерия | Критерии категории команды разработчиков  проекта | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 3. | Изменяются ли роли участников проекта на протяжении ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Является ли структура процесса разработки более значимой для разработчиков, чем гибкость? | Да | Да | Нет | Да | Нет | Нет |
| 5. | Важна ли легкость распределения человеческих ресурсов проекта? | Да | Да | Да | Да | Нет | Нет |
| 6. | Приемлет ли команда разработчиков оценки, проверки, стадии разработки? | Да | Да | Нет | Да | Да | Да |

Вычисления: 5 за каскадную, 5 за V-образную, 4 за RAD, 5 за инкрементную, 2 за быстрого прототипирования и 3 за эволюционную.

Итог: На основе результатов заполнения табл. 4 подходящими являются каскадная, V-образная и инкрементная модели.

Таблица 3 – Выбор модели жизненного цикла на основе характеристик коллектива пользователей

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории коллектива пользователей | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Будет ли присутствие пользователей ограничено в ЖЦ разработки? | Да | Да | Нет | Да | Нет | Да |
| 2 | Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки? | Нет | Нет | Нет | Да | Да | Да |
| 3 | Будут ли пользователи вовлечены во все фазы ЖЦ разработки? | Нет | Нет | Да | Нет | Да | Нет |
| 4 | Будет ли заказчик отслеживать ход выполнения проекта? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 0 за каскадную, 0 за V-образную, 2 за RAD, 1 за инкрементную, 4 за быстрого прототипирования и 2 за эволюционную.

Итог: На основе результатов заполнения табл. 5 подходящей является модель быстрого проектирования.

Таблица 4 – Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории типов проекта и рисков | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Разрабатывается ли в проекте продукт нового для организации направления? | Нет | Нет | Нет | Да | Да | Да |
| 2. | Будет ли проект являться расширением существующей системы? | Да | Да | Да | Да | Нет | Нет |
| 3. | Будет ли проект крупно- или среднемасштабным? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Ожидается ли длительная эксплуатация продукта? | Да | Да | Нет | Да | Нет | Да |
| 5. | Необходим ли высокий уровень надежности продукта проекта? | Нет | Да | Нет | Да | Нет | Да |
| 6. | Предполагается ли эволюция продукта проекта в течение ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Велика ли вероятность изменения системы (продукта) на этапе сопровождения? | Нет | Нет | Нет | Да | Да | Да |
| 8. | Является ли график сжатым? | Нет | Нет | Да | Да | Да | Да |
| 9. | Предполагается ли повторное использование компонентов? | Нет | Нет | Да | Да | Да | Да |
| 10. | Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 2 за каскадную, 3 за V-образную, 3 за RAD, 7 за инкрементную, 7 за быстрого прототипирования и 9 за эволюционную.

Итог: На основе результатов заполнения табл. 6 подходящей является эволюционная модель.

Вычисления: 11 за каскадную, 12 за V-образную, 12 за RAD, 18 за

инкрементную, 16 за быстрого прототипирования и 17 за эволюционную.

Общий итог: в итоге заполнения табл. 3 – 6 наиболее подходящей является инкрементная модель

**2 Проектирование**

Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case diagram) служит для описания функциональности системы с точки зрения ее пользователей. Она позволяет идентифицировать и описать основные сценарии использования системы.

**2.1 Диаграмма вариантов использования**

Диаграмма вариантов использования используется для описания функциональности, которую система должна предоставлять для пользователей. Она позволяет определить, какие действия могут выполнять пользователи и как система должна реагировать на эти действия.

С диаграммой вариантов использования для игрового приложения «PlatformGuy» можно ознакомиться в Приложении А.

**2.2 Модель данных**

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных.

Модели данных служат для проектирования структуры постоянных хранилищ данных, используемых системой. Профайл на языке UML для проектирования базы данных предоставляет разработчикам базы данных набор элементов моделирования, позволяющих разрабатывать подробный макет таблиц в базе данных и моделировать макет физической памяти базы данных. Профайл базы данных на языке UML также предоставляет конструкции для моделирования целостности по ссылкам (ограничений и триггеров), а также хранимых процедур, предназначенных для управления доступом к базе данных.

Модели данных могут создаваться на уровне предприятия, отдела или отдельного приложения. Модели данных на уровне предприятия или отдела могут использоваться для предоставления стандартных определений для ключевых бизнес-сущностей (таких как клиент и сотрудник), которые будут применяться всеми приложениями всего бизнес-процесса или его части. С помощью этих типов Моделей данных можно также определить, какая система предприятия будет "владельцем" данных для конкретной бизнес-сущности и какие другие системы будут пользователями (подписчиками) данных.

С моделью данных для игрового приложения «PlatformGuy» можно ознакомиться в Приложение Б.

**2.3 Разработка UML-диаграмм**

**2.3.1 Диаграмма классов**

Диаграмма классов - это графическое представление классов, интерфейсов и их отношений в объектно-ориентированной системе. Она показывает, как классы связаны друг с другом и как они взаимодействуют.

С диаграммой классов для игрового приложения «PlatformGuy» можно ознакомиться в Приложение В.

**2.3.1 Диаграмма обьектов**

Диаграмма объектов y является графическим представлением объектов, которые существуют в системе и их связей друг с другом. Она позволяет увидеть структуру объектов в системе и их взаимодействие.

На диаграмме объектов обычно присутствуют следующие элементы:

1. Объекты – это конкретные экземпляры классов, которые существуют в системе.

2. Атрибуты – это свойства объектов, которые могут быть использованы для описания их состояния.

3. Связи – это отношения между объектами, которые могут быть использованы для описания их взаимодействия.

С диаграммой обьектов для игрового приложения «PlatformGuy» можно ознакомиться в Приложение Г.

**2.4 Разработка пользовательского интерфейса**

Важным элементом проектирования программного продукта является описание пользовательского интерфейса разрабатываемого приложения. Для разработки визуального дизайна использовались мягкие цвета для удобства использования программного продукта.

При разработке был разработ UI интерфейс, который можно просмотреть в Прилодение Д.

Прототип – это наглядная модель пользовательского интерфейса. В сущности, это «черновик» созданный на основе представления разработчика о потребностях пользователя. Итоговое отображение программы может отличаться от прототипа. С прототипом главной страницы(UX) вы можете ознакомиться в Приложении Е.

**2.5 Разработка плана работы над проектом**

Для разработки плана над проектом был использован draw.io.

Диаграммы Ганта – это инструмент управления проектами, иллюстрирующий то, как выполняется запланированная работа с течением времени. Обычно она состоит из двух частей: в левой части приведен список заданий, а в правой – временная шкала с полосами, которые изображают работу. Диаграмма Ганта также может включать даты начала и завершения заданий, контрольные точки, зависимости между заданиями и исполнителей. С данной диаграммой можно ознакомиться в Приложении Ж.

**3Реализация**

**3.1 Руководство программиста**

Данное игровое приложение было разработано на движке Unity, скрипты написаны на C#. Unity предоставляет обширный инструментарий для создания игровых приложений.

Создание игрового приложения

Нужно зарегистрироваться , и скачать Unityhub.

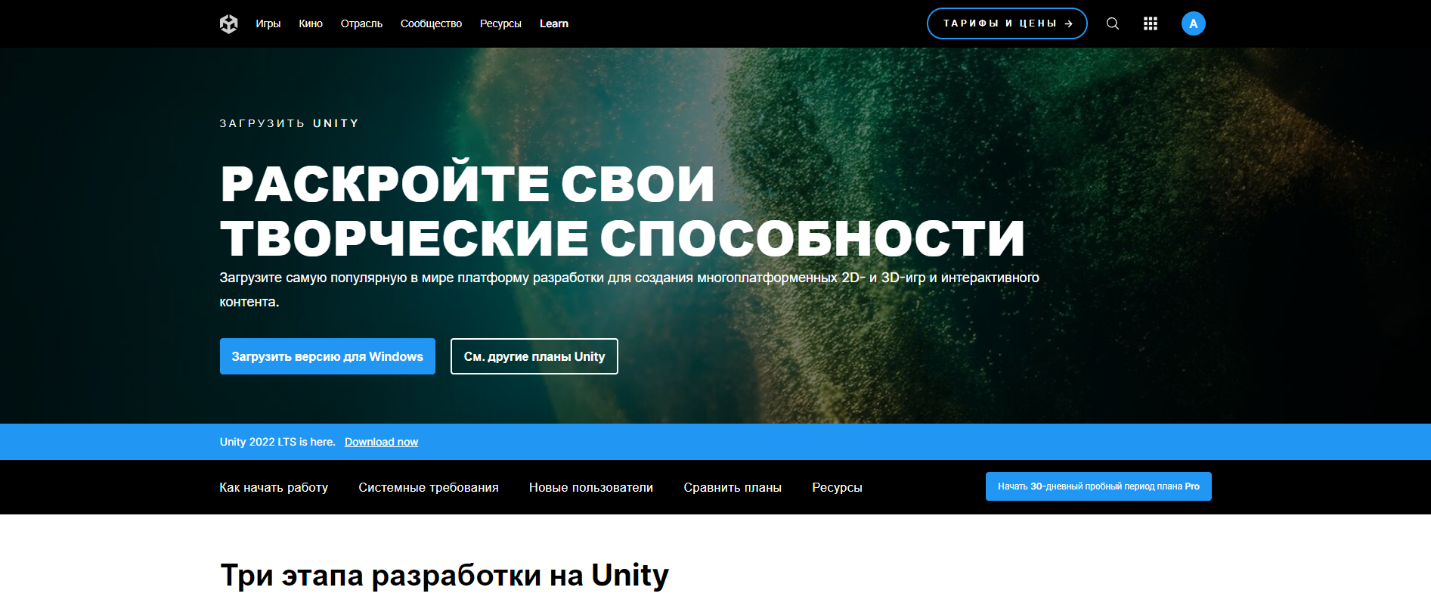


Рисунок 1-Сайт Unity

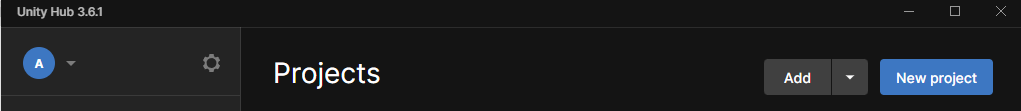
Зайдя в Unityhub нужно создать новый проект.

Рисунок 2- Добавление проекта

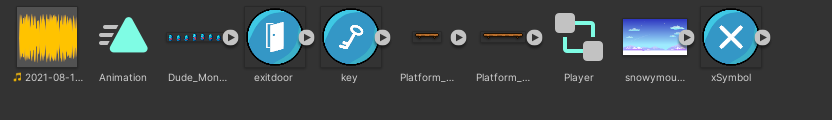
Загружаем все нужные для разработки спрайты

Рисунок 3 – Добавление спрайтов

Для добавления различных элементов нажимаем правую кнопку мыши и выбираем нужный.

Рисунок 4 – Добавление UI элементов

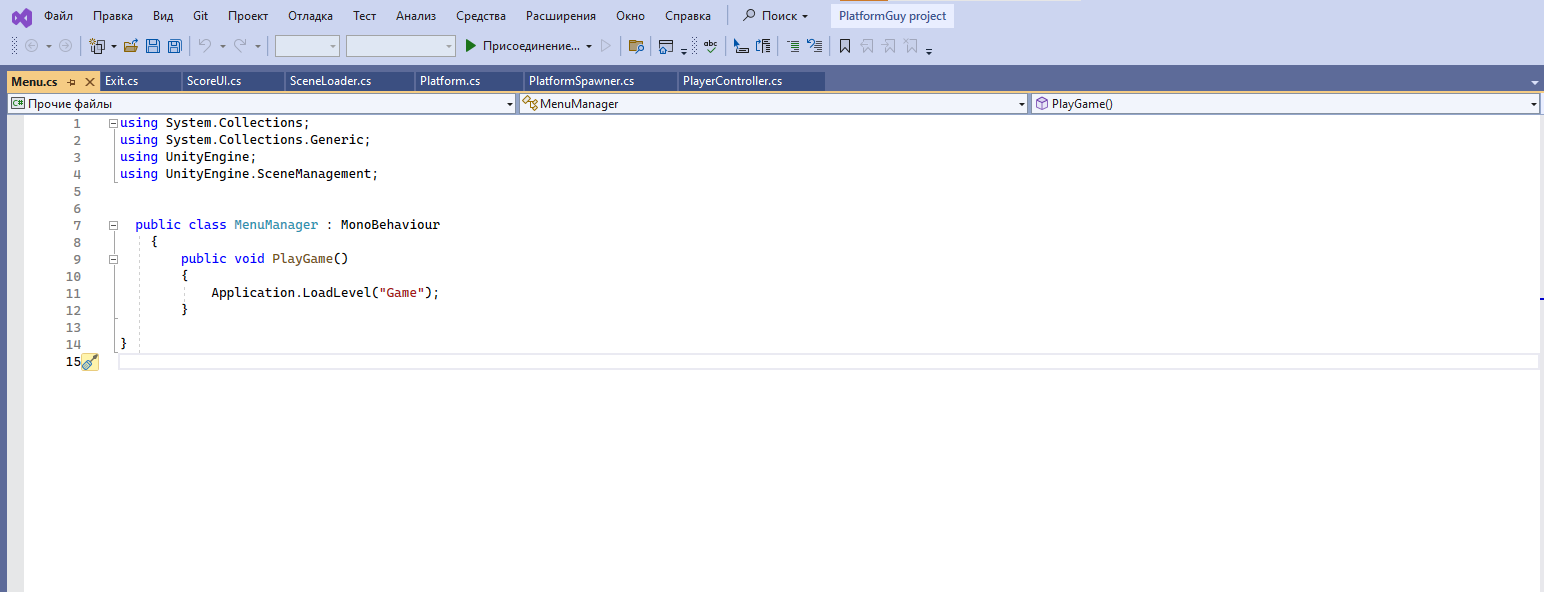
Для добавления кода выбираем в этом же меню создания скрипта. Открываем скрипт через Microsoft visual studio и пишем код.

Рисунок 5 - Microsoft visual studio

Действия на элемент добавляем через скрипты, и перетаскиваем скрипт на нужный элемент. Выбираем нужное действие.

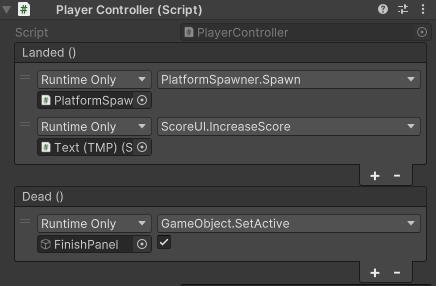


Рисунок 6 - Связь скрипта с UI элементом

Для добавления меню, нужно создать новую сцену. Для этого через правую кнопку мыши выбираем создание сцены.

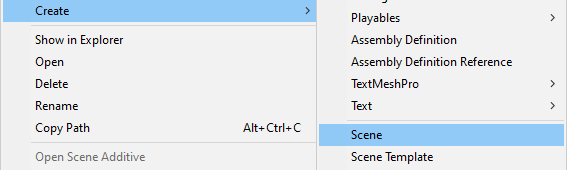


Рисунок 7 - создание сцены

Анимация для персонажа добавляется в одноименном меню.

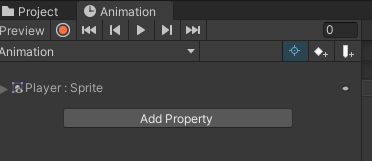


Рисунок 8 - анимация

Для создания работоспособного exe.файла, выбираем в верхнем левом углу “Build Settings”. Настраиваем и добавляем иконку для ярлыка , запускаем.

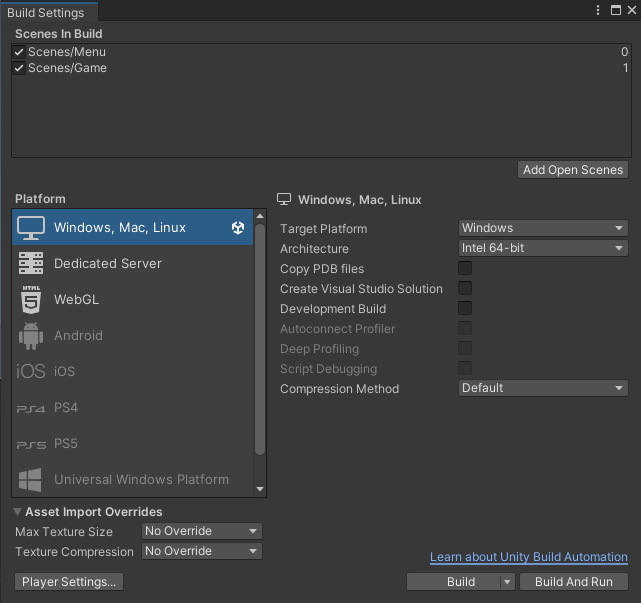


Рисунок 9- Build меню

**4 Тестирование**

При разработке проекта возникающие ошибки и недоработки были исправлены на этапе реализации проекта. После завершения этапа написания программы было проведено тщательное функциональное и нефункциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программы в автономном режиме.

Таблица 5 -отчет о результатах тестирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Название теста** | **Действие** | **Исходная информация** | **Ожидаемый результат** | **Физический результат** | **Результат тестирования** |
| Запуск программы | Открытие ярлыка приложения | Отсутствует | Открытие программы | Открытие программы | Выполнено |
| Выход из приложения | Нажатие на кнопку выхода, на крестик | Отсутствует | Выход из программы | Выход из программы | Выполнено |
| Открытие правил | Нажатие на кнопку правил,на иконку ключа | Отсутствует | Открытие правил | Открытие правил | Выполнено |
| Переход к выборам уровней | Нажатие на кнопку начать игру | Отсутствует | Открытие списка уровней | Открытие списка уровней | Выполнено |
| Запуск уровня | Нажатие на уровень из списка | Отсутствует | Запуск выбранного уровня | Запуск выбранного уровня | Выполнено |
| Управление персонажем | Нажатие на левую кнопку мыши | Отсутствует | Персонаж прыгает | Персонаж прыгает | Выполнено |
| Выход в главное меню | Нажатие на иконку двери | Отсутствует | Выход в главное меню | Выход в главное меню | Выполнено |
| Остановка платформ | Приземление на платформу | Отсутствует | Остановка платформы после приземления на нее персонажем | Остановка платформы после приземления на нее персонажем | Выполнено |
| Подсчет очков | Проверка счёта | Отсутствует | Начисление очков каждую секунду | Начисление очков каждую секунду | Выполнено |
| **Название теста** | **Действие** | **Исходная информация** | **Ожидаемый результат** | **Физический результат** | **Результат тестирования** |
| Проверка рекорда | Проверка рекорда после перезапуска уровня,приложения | Отсутствует | Рекорд начисляется | Рекорд начисляется | Выполнено |
| Проверка оконьчания игры | Столкновение персонажа с платформой | Отсутствует | Остановка игры, отображение окна проигрыша | Остановка игры, отображение окна проигрыша | Выполнено |
| Перезапуск уровня после проигрыша | Нажатие на кнопку “Повторить” после проигрыша | Отсутствует | Уровень запускается заново,а счёт обнуляется | Уровень запускается заново,а счёт обнуляется | Выполнено |

В результате проведения тестирования выяснилось, что все требования, предъявляемые к ПП, были разработаны и протестированы. Тесты показали, что все функции работают правильно.

**5 Руководство пользователя**

**5.1 Общие сведения о программном продукте**

Разрабатываемое игровое приложение “PlatformGuy” служит для развития пространственного мышления, а так же для развлечения и отдыха.

**5.2 Инсталляция**

Данный программный продукт нужно разархивировать в удобное место на компьютере.

**5.3 Выполнение программы**

**5.3.1 Запуск программы**

Для запуска приложения нажмите на ярлык.

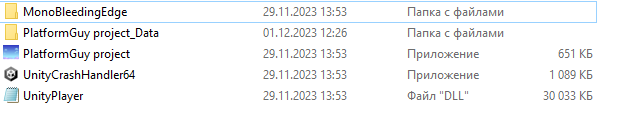


Рисунок 10- запуск приложения

**5.3.2 Инструкция по работе с программой**

После заставки появится главное меню. Для просмотра правил, выберите кнопку с одноименным названием, для запуска игры- соотвеетствующую кнопку.

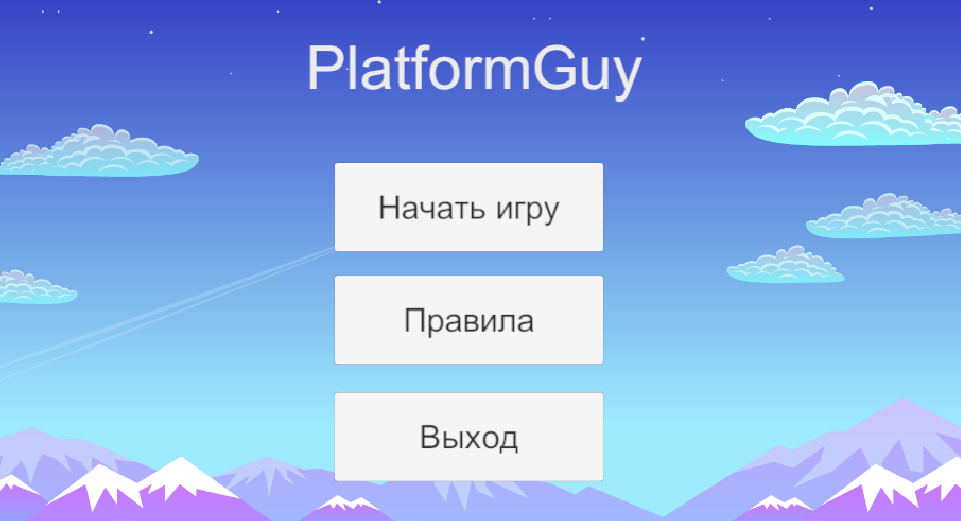


Рисунок 11 - Главное меню

Для выхода из правил, нажмите на крест.

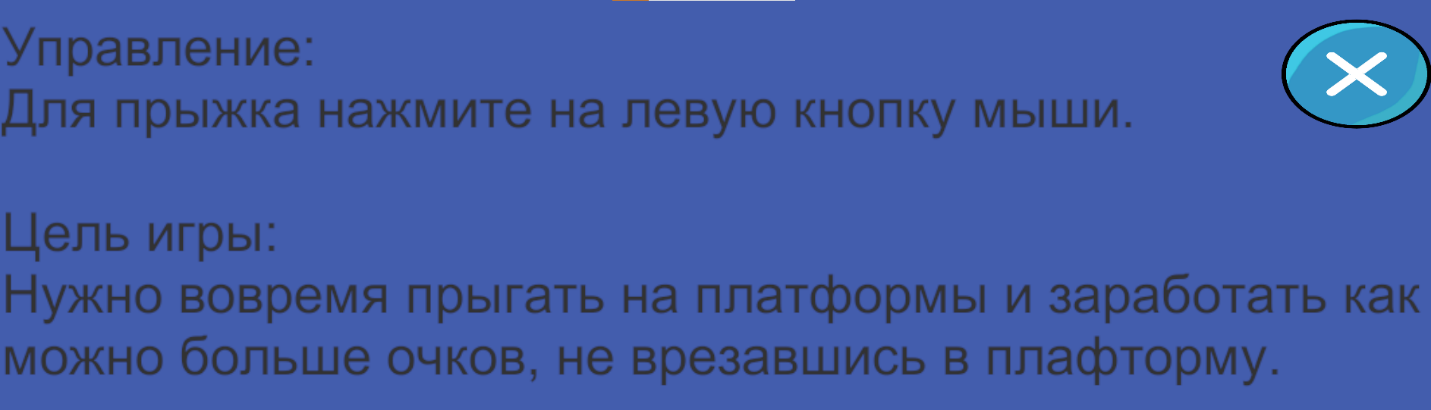


Рисунок 12- Правила

Нажав на “Начать игру”, начнется процесс игры. Управление персонажем осуществляется левой кнопкой мыши. Для просмотра правил-выберите значок ключа , для выхода-значок крестика , для выхода в главное меню-значок двери.

Рисунок 13- Игровой процесс

**5.3.3 Завершение работы с программным продуктом**

Завершить работу можно при нажатии на крест во время игрового процесса. Либо через кнопку “Выход” в главном меню.

**Заключение**

Целью данного проекта на практику являлась разработка игрового приложения в жанре платформер “PlatformGuy”. В программном продукте был реализован простой и интуитивно понятный интерфейс. Благодаря реализации механики счёта и рекорда, пользователь может усовершенствовать свой результат из раза в раз.

После и тщательного тестирования приложения были выявлены небольшие ошибки и недоработки, которые в последствии были исправлены.

При реализации программного продукта, были выполнены все условия, поставленные на начальном этапе разработки.

Так же в процессе создания программного продукта была подготовлена программная документация

**Список использованных источников**

1 Поиск бесплатных спрайтов - <https://craftpix.net/freebies/>

2 Руководства по Unity - <https://docs.unity3d.com/ru/530/Manual/UnityManual.html>

3 Создание диаграмм - <https://app.diagrams.net/>

**Приложение А**

**Диаграмма вариантов использования(Рисунок 14)**

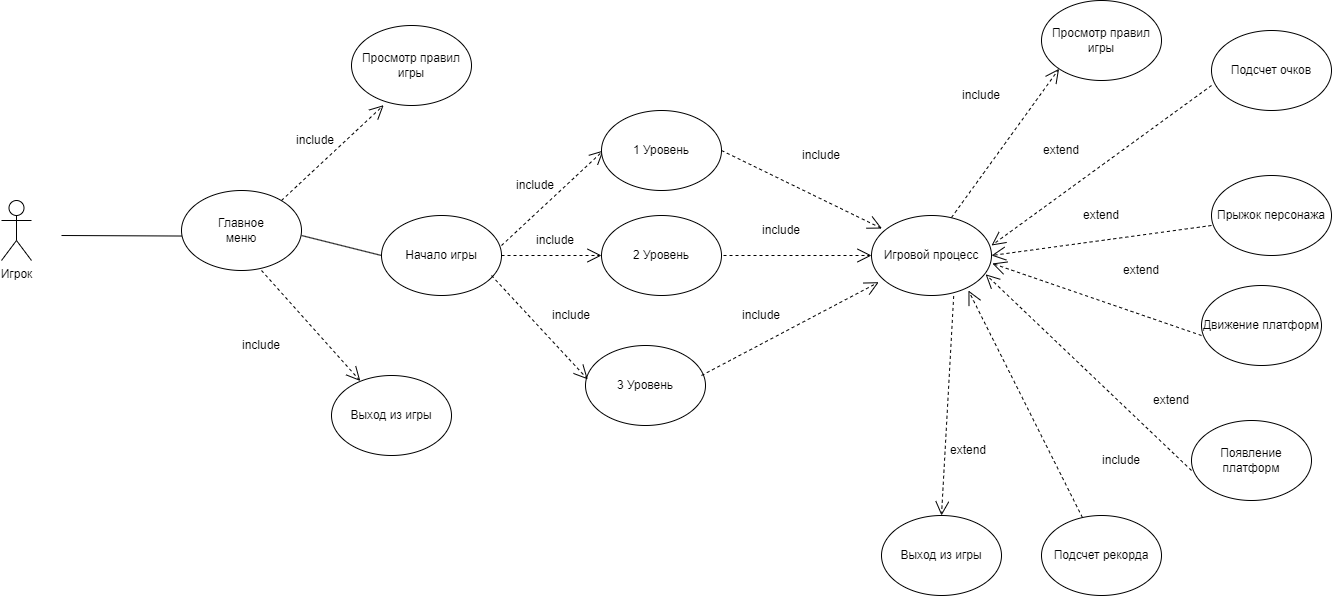


Рисунок 14-диаграмма вариантов использования

**Приложение Б**

**Модель данных(Рисунок 15)**

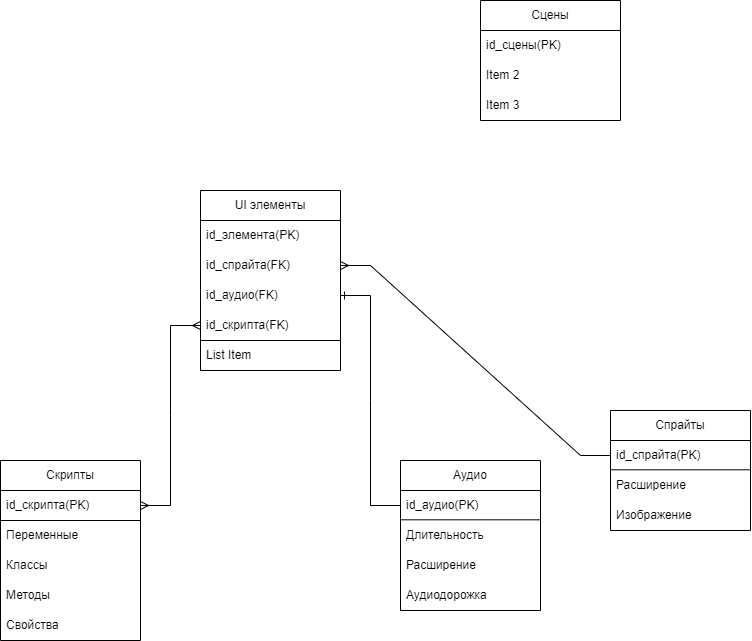


Рисунок 15 – Модель данных

**Приложение В**

**Диаграмма классов (Рисунок 16)**

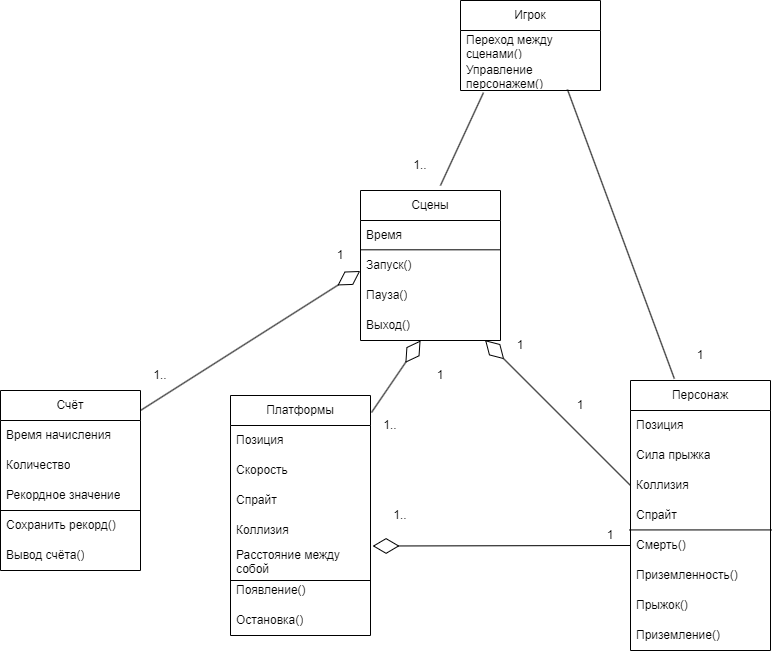


Рисунок 16 – Диаграмма классов

**Приложение Г**

**Диаграмма обьектов (Рисунок 17)**

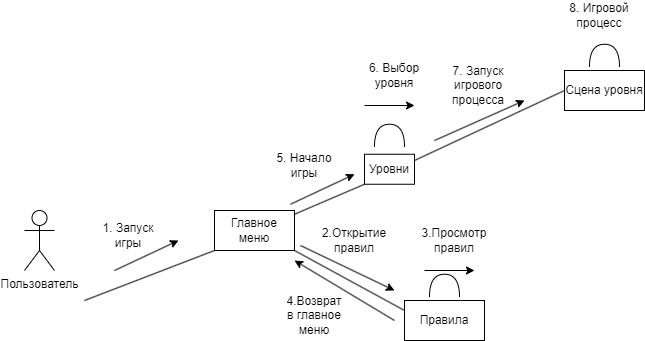
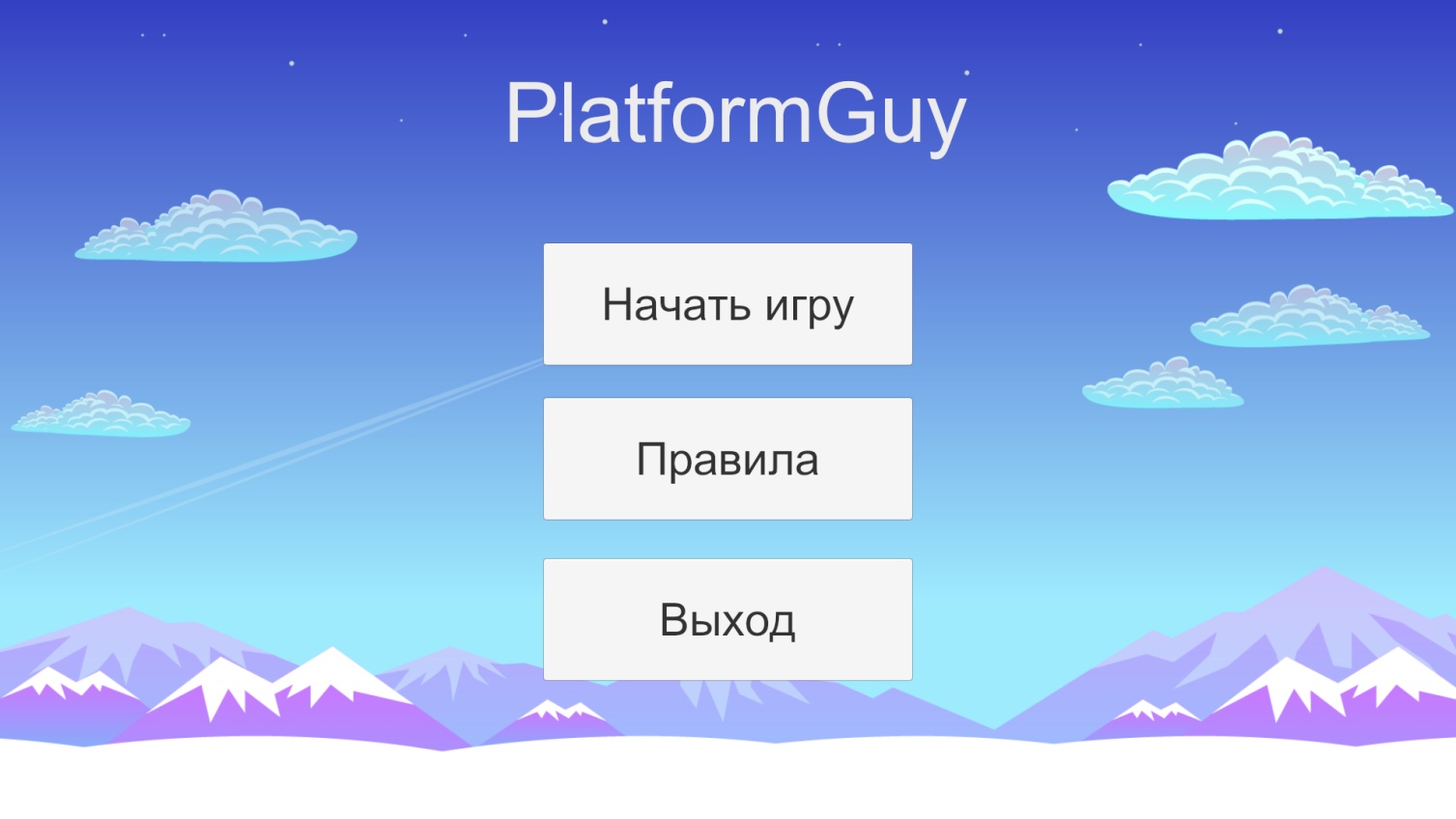


Рисунок 17- диаграмма обьектов

**Приложение Д**

**UI интерфейс (Рисунок 18-23)**

Рисунок 18- Главное меню

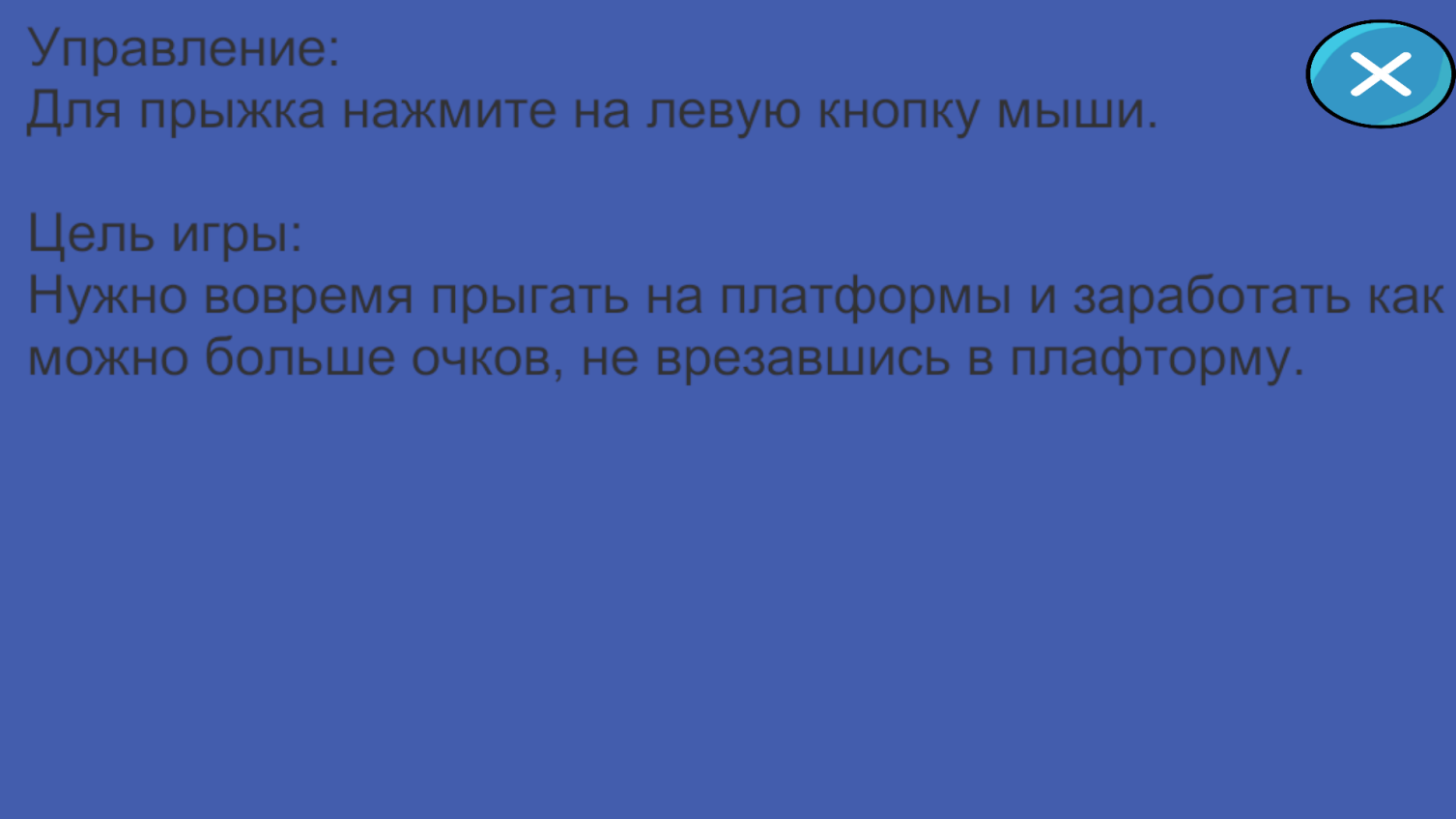


Рисунок 19 - Правила

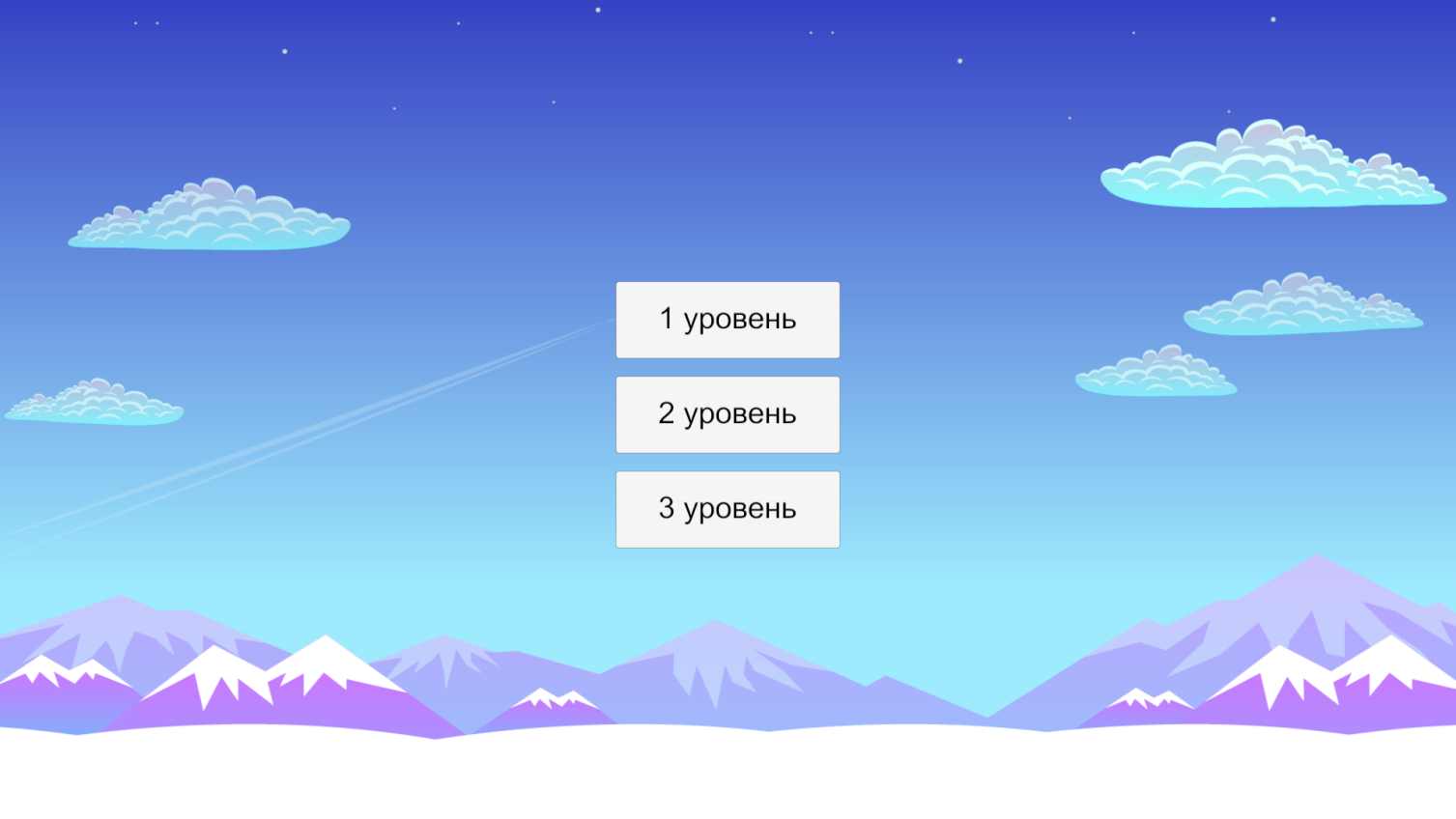


Рисунок 20 – Уровни

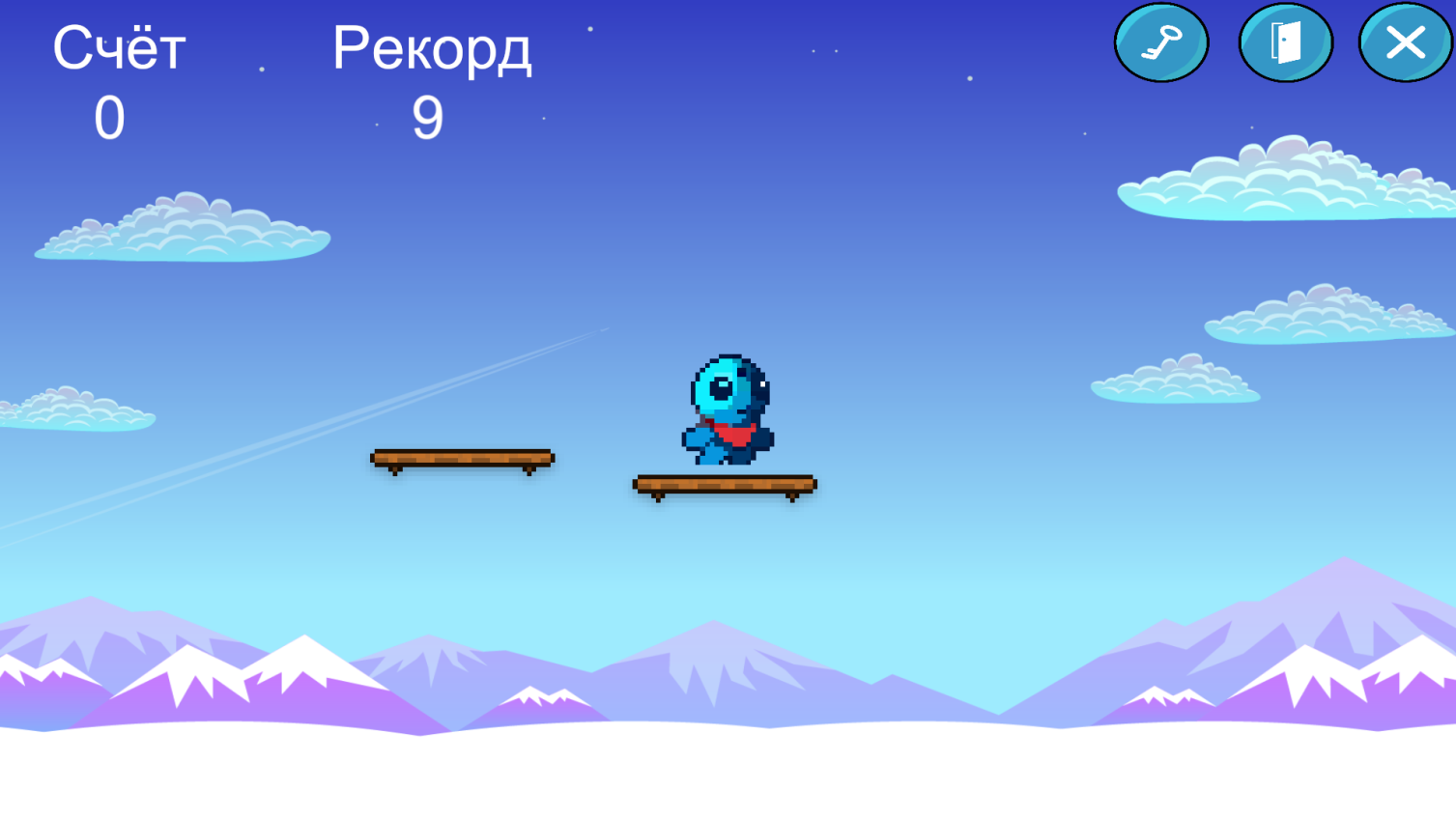


Рисунок 21 – 1 уровень

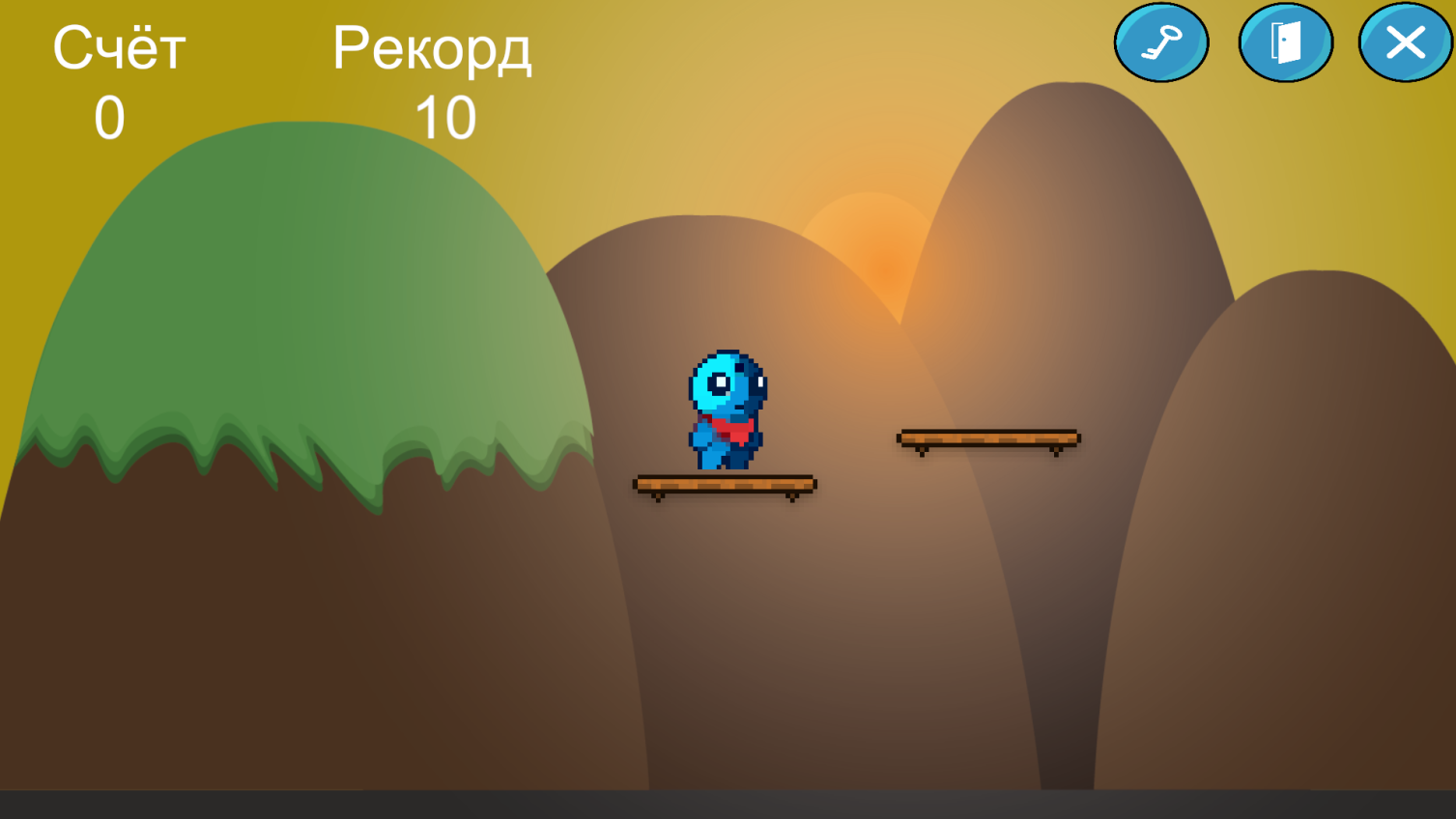


Рисунок 22- 2 уровень



Рисунок 23 – 3 уровень

**Приложение E**

**UX интерфейс (Рисунок 24-27)**

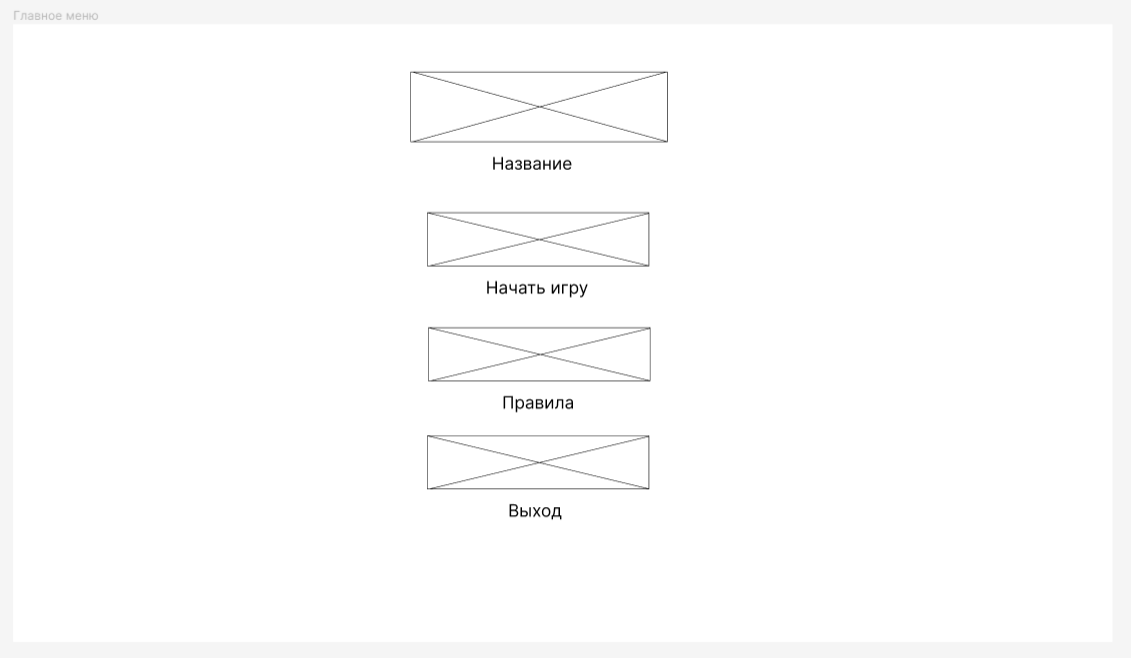


Рисунок 24 – Главное меню

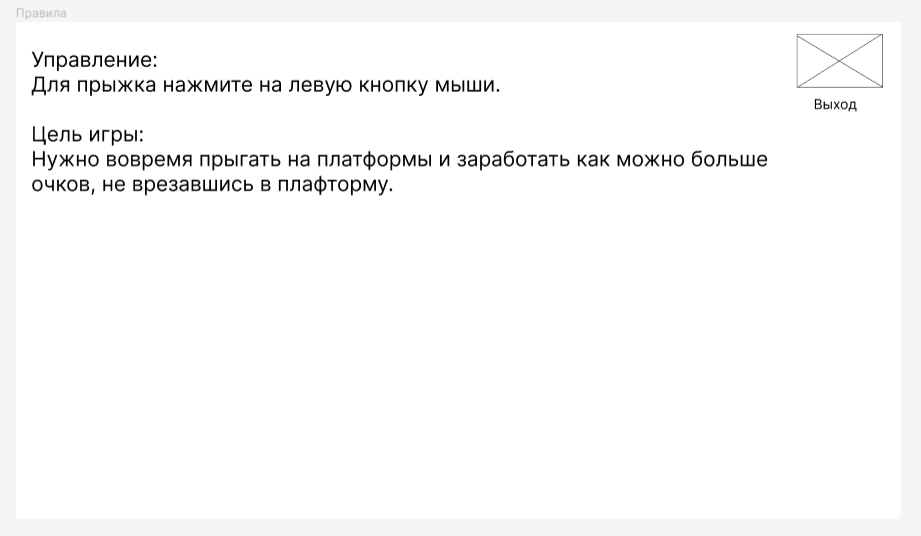


Рисунок 25 – Правила

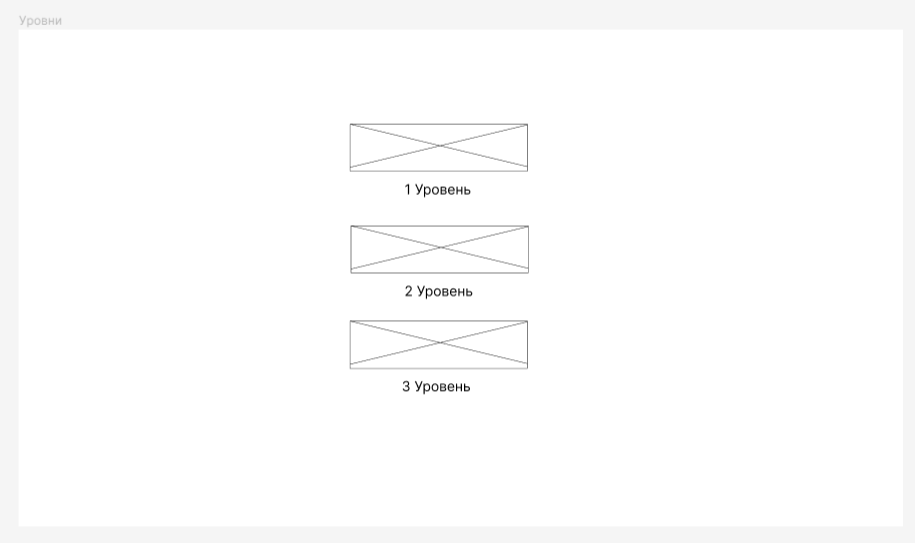


Рисунок 26 – Уровни

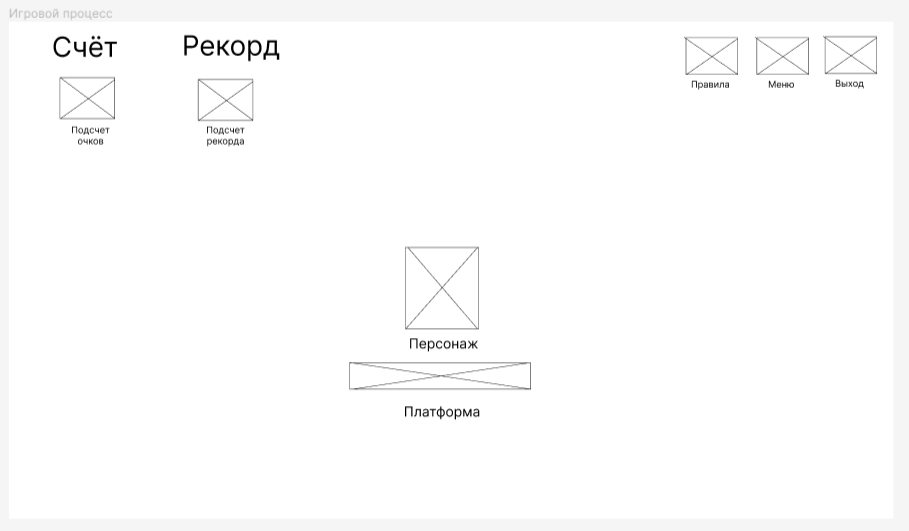


Рисунок 27 – Игровой процесс

**Приложение Ж**

**Диаграмма Ганта (Рисунок 28)**

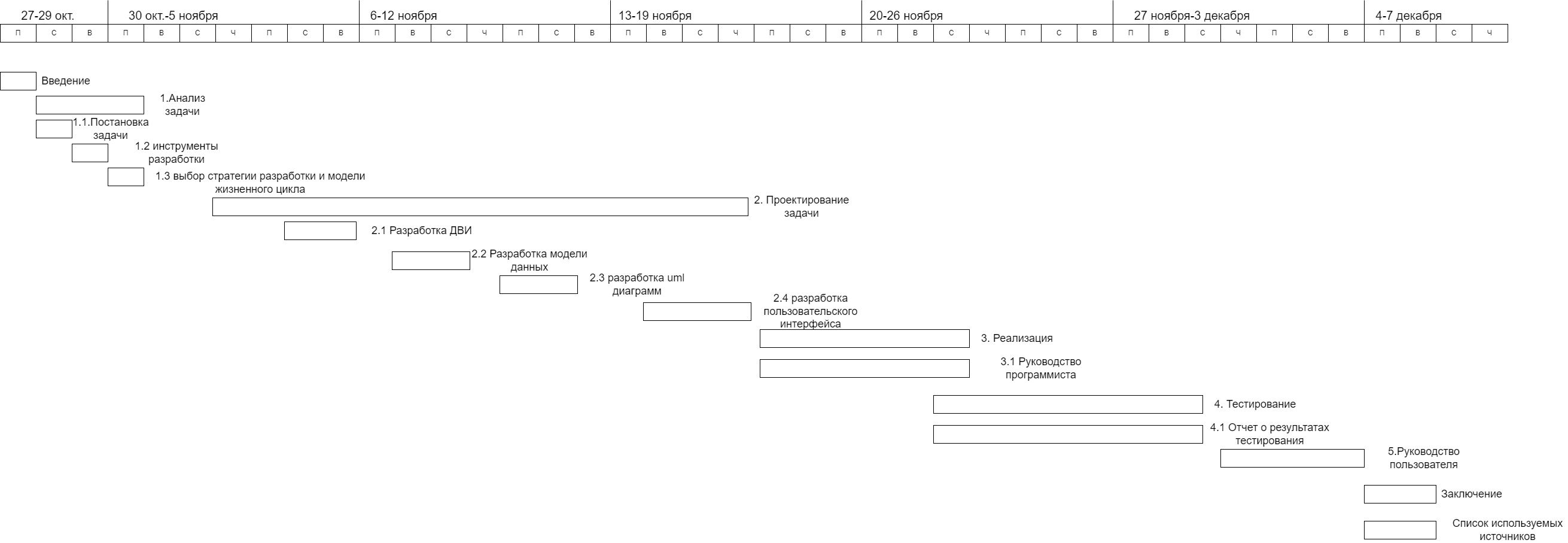


Рисунок 28-Диаграмма Ганта

**Приложение З**

**Листинг программы**

public class Exit : MonoBehaviour

{

public void ExitGame()

{

Debug.Log("Игра закрылась");

Application.Quit();

}

}

public class Control2 : MonoBehaviour

{

public Text scoreText;

public int score2;

public float time2;

private float timeStart2;

void Start()

{

ScoreManager2.score2 = 0;

timeStart2 = time2;

}

void Update()

{

time2 -= Time.deltaTime;

if (time2 <= 0)

{

ScoreManager2.score2 += 1;

time2 = timeStart2;

}

}

}

public class Control1 : MonoBehaviour

{

public Text scoreText;

public int score1;

public float time1;

private float timeStart1;

void Start()

{

ScoreManager1.score1 = 0;

timeStart1 = time1;

}

void Update()

{

time1 -= Time.deltaTime;

if (time1 <= 0)

{

ScoreManager1.score1 += 1;

time1 = timeStart1;

}

}

public class Control : MonoBehaviour

{

public Text scoreText;

public int score ;

public float time;

private float timeStart;

void Start()

{

ScoreManager.score = 0;

timeStart = time;

}

void Update()

{

time -= Time.deltaTime;

if (time <= 0)

{

ScoreManager.score += 1;

time = timeStart;

}

}

}

public class exitmenu: MonoBehaviour

{

public void PlayGame()

{

Time.timeScale = 1;

Application.LoadLevel("Menu");

}

}

public class level : MonoBehaviour

{

public void PlayGame()

{

Application.LoadLevel("Game");

}

}

public class level2 : MonoBehaviour

{

public void PlayGame()

{

Application.LoadLevel("Game 1");

}

}

{

public void PlayGame()

{

Application.LoadLevel("Game 2");

}

public class MenuManager : MonoBehaviour

{

public void PlayGame()

{

Application.LoadLevel("Levels");

}

}

public class Pause : MonoBehaviour

{

public void pas()

{

Time.timeScale = 0 ;

}

}

public class Platform : MonoBehaviour

{

[SerializeField] private float \_moveSpeed;

private GameObject \_player;

private int \_moveDirection;

private bool \_hasToMove = true;

private void Awake()

{

\_player = GameObject.FindGameObjectWithTag("Player");

\_moveDirection = transform.position.x < \_player.transform.position.x ? 1 : -1;

}

private void Update()

{

if (\_hasToMove == true)

transform.position += Vector3.right \* \_moveDirection \* \_moveSpeed \* Time.deltaTime;

}

public void StopMovement() => \_hasToMove = false;

private void OnBecameInvisible()

{

Destroy(gameObject);

}

}

public class PlatformSpawner : MonoBehaviour

{

[SerializeField] private Transform[] \_spawnPoints;

[SerializeField] private GameObject \_platformPrefab;

[SerializeField] private float \_verticalOffset = 0.5f;

private float? \_lastPointPositionY = null;

private void Start()

{

Spawn();

}

public void Spawn()

{

Transform randomSpawnPoint = \_spawnPoints[Random.Range(0, \_spawnPoints.Length)];

float spawnPointPositionY = \_lastPointPositionY == null ? randomSpawnPoint.position.y : (float)\_lastPointPositionY + \_verticalOffset;

randomSpawnPoint.position = new Vector3(randomSpawnPoint.position.x, spawnPointPositionY, randomSpawnPoint.position.z);

\_lastPointPositionY = spawnPointPositionY;

Instantiate(\_platformPrefab, randomSpawnPoint.position, Quaternion.identity);

}

}

public class PlayerController : MonoBehaviour

{

public UnityEvent Landed;

public UnityEvent Dead;

[SerializeField] private float \_jumpForce;

[SerializeField] private ContactFilter2D \_platform;

private Rigidbody2D \_rigidbody;

private bool \_isOnPlatform => \_rigidbody.IsTouching(\_platform);

private void Awake()

{

\_rigidbody = GetComponent<Rigidbody2D>();

}

public void Jump()

{

if (\_isOnPlatform == true)

\_rigidbody.AddForce(Vector2.up \* \_jumpForce, ForceMode2D.Impulse);

}

private void OnCollisionEnter2D(Collision2D collision)

{

GameObject collisionObject = collision.gameObject;

if (collisionObject.transform.parent != null)

{

if (collisionObject.transform.parent.TryGetComponent(out Platform platform))

platform.StopMovement();

}

if (collisionObject.CompareTag("PlatformWall"))

Dead?.Invoke();

else if (collisionObject.CompareTag("Platform"))

{

collisionObject.tag = "Untagged";

Landed?.Invoke();

}

}

}

public class SceneLoader : MonoBehaviour

{

public void ReloadScene()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

Time.timeScale = 1 ;

}

}

public class ScoreManager : MonoBehaviour

{

[SerializeField] Text HighscoretText;

[SerializeField] Text Score;

public static float score;

int hightscore;

void Update()

{

hightscore = (int)score;

Score.text = "" + hightscore.ToString();

if (PlayerPrefs.GetInt("score") <= hightscore)

{

PlayerPrefs.SetInt("score", hightscore);

}

HighscoretText.text = "" + PlayerPrefs.GetInt("score").ToString();

}

}

public class ScoreManager1 : MonoBehaviour

{

[SerializeField] Text HighscoretText1;

[SerializeField] Text Score1;

public static float score1;

int hightscore;

void Update()

{

hightscore = (int)score1;

Score1.text = "" + hightscore.ToString();

if (PlayerPrefs.GetInt("score1") <= hightscore)

{

PlayerPrefs.SetInt("score1", hightscore);

}

HighscoretText1.text = "" + PlayerPrefs.GetInt("score1").ToString();

}

}

public class ScoreManager2 : MonoBehaviour

{

[SerializeField] Text HighscoretText2;

[SerializeField] Text Score2;

public static float score2;

int hightscore;

void Update()

{

hightscore = (int)score2;

Score2.text = "" + hightscore.ToString();

if (PlayerPrefs.GetInt("score2") <= hightscore)

{

PlayerPrefs.SetInt("score2", hightscore);

}

HighscoretText2.text = "" + PlayerPrefs.GetInt("score2").ToString();

}

}