Учреждение образования

«Гродненский государственный политехнический колледж»

Специальность: 2–40 01 01  «Программное    обеспечение       информационных технологий»

Специализация: 2 – 40 01 01 35 «Программное обеспечение обработки

экономической и деловой информации»

Предмет: «Основы алгоритмизации и программирования»

Группа: ПЗТ – 38

**ТЕХНО-РАБОЧИЙ ПРОЕКТ**

Тема: «Разработка игрового приложение «Arena Dominus»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

Разработал Д. В. Лебедь

Руководитель проекта Е. В. Заяц

2023

**Содержание**

Введение 3

*Изм.*

*Кол*

*Лист №док*

*Подпись*

*Дата*

*Лист*

*2*

*ТРПО 2-40 01 01.35.38.13.24 ПЗ*

*Разраб.*

*Лебедь*

*Пров.*

*Заяц*

*Н. контр.*

*Утв.*

*Разработка игрового приложения ”Arena Dominus”*

*Стадия*

*Листов*

*36*

*УО ГГПК*

1 Анализ задачи 5

1.1 Постановка задачи 5

1.2 Выбор стратегии разработки и модели жизненного цикла 6

1.3 Инструменты разработки 10

2 Проектирование задачи 11

2.1 Организация данных 11

2.2 Разработка UML-диаграмм 11

2.3 Разработка пользовательского интерфейса 15

3 Реализация 18

3.1 Руководство программиста 18

4 Тестирование 22

4.1 Отчёт о результатах тестирования 22

5 Руководство пользователя 22

5.1 Общие сведенья о программном продукте 24

5.2 Инсталляция 24

5.3 Выполнение программы 24

5.3.1 Запуск программы 24

5.3.2 Инструкция по работе с программой 24

Заключение .28

Список использованных источников 29

Приложение А 30

Приложение Б 33

**Введение**

Была поставлена задача разработать проект на тему: Игровое приложение «Arena Dominus».

В настоящее время игры являются неотъемлемой частью жизни многих людей. Игры используются для развлечения и развития, а также для отдыха.

Создаваемое приложение будет рассчитано на любого рода пользователей и может использоваться в любой момент по желанию пользователя.

Далее приведём краткое описание разделов пояснительной записки.

Первый раздел носит название “Анализ задачи”. В нем вы сможете ознакомиться с постановкой задачи, которая включает в себя: исследование предметной области поставленной задачи. Также в этом разделе вы сможете узнать о том, как данная задача решается в настоящее время. Все входные и выходные данные тоже будут описаны в первом разделе. В подразделе “Выбор стратегии разработки и модели жизненного цикла” будут представлены таблицы, с помощью которых была выбрана модель жизненного цикла. В подразделе “Инструменты разработки” будет рассмотрена среда, в которой создаётся данный курсовой проект. Здесь также будут установлены минимальные и оптимальные требования к аппаратным характеристикам, обеспечивающим правильное функционирование поставленной задачей.

В разделе “Проектирование задачи” будут рассмотрены основные аспекты разработки программного продукта. Здесь можно будет узнать об организации данных в контексте среды разработки. В данном разделе будет чётко описан пользовательский интерфейс, составлены алгоритмы процесса обработки информации, описана разработка системы справочной информации. Так же тут будут представлены UML диаграммы.

“Реализация задачи” – это третий раздел пояснительной записки, в котором описываются элементы и объекты, которые будут использованы при реализации данного приложения. В этом разделе будут чётко описаны функции пользователя и их структура.

Четвёртый раздел – “Тестирование”. В нем будет описано полное и функциональное тестирование данной программы, т.е. будет оттестирован каждый пункт меню, каждая операция, которая выполняется приложением. Будут смоделированы все возможные действия пользователя при работе с программой, начиная от запуска до выхода.

В разделе “Руководство пользователя” будет описано как пользоваться программой.

“Заключение” будет содержать краткую формулировку задачи, результаты проделанной работы, описание использованных методов и средств, описание степени автоматизации процессов на различных этапах разработки, степень соответствия проектных решений поставленной задаче и возможный модификации проекта.

В “Списке используемых источников” будет приведён список используемых при разработке источников.

В приложениях к пояснительной записке будет приведена функциональная модель и листинг программы с необходимыми комментариями.

**1 Анализ задачи**

# 1.1 Постановка задачи

**Наименование задачи:** Разработка игрового приложения «Arena Dominus».

**Цель разработки:** создание игрового приложения «Arena Dominus», которое рассчитано на людей, интересующихся различными играми, мотивируя их к изучению такой сферы в IT, как разработка видеоигр.

**Назначение:** для развлечения и отдыха, развития скорости реакции, демонстрацииреализации игрового приложения с графическим представлением.

**Предметная область:** вданной предметной области существуют видеоигры с схожими особенностями и механиками. Например, Red Alert 3 и StarCraft 2.

**Периодичность использования:** может использоваться в любой момент по желанию пользователя.

**Функциональные требования:** Игрок управляет отрядом роботов в режиме реального времени. В начале каждой игры отряд игрока появляется на арене и должен победить враждебный отряд. Так же на арене находятся ловушки, от которых нужно уклоняться. После победы игрок получает валюту, количество которой зависит от сложности уровня и переходит в магазин, в котором тратит валюту на новых персонажей, улучшение имеющихся и покупку различных способностей, которые можно использовать во время боя. После этого игрок переходит на карту, на которой выбирает следующий уровень, которые отличаются типом, сложностью противников и ландшафтом. Так повторяется до тех пор, пока игрок не пройдёт финальный уровень. Каждый робот имеет различные характеристики (скорость, дальность атаки, скорость атаки, урон от атаки, здоровье и т.д.). Так же некоторые роботы имеют уникальные способности (как активный, так и пассивные). Входной информацией является валюта игрока и его набор роботов. Выходной информацией является количество полученной валюты в конце боя. Условно-постоянной информацией являются характеристики роботов, карта, ландшафт на уровнях (река, холм, мост и т.д.)

**Входная, выходная и постоянная информация:** к входной информации можно отнести нажатие игроком клавиш управления. К выходной информации можно отнести количество полученных очков. К постоянной информации можно отнести изображения.

Разрабатываемый программный продукт позволит выполнить следующие действия:

* Осуществить игровую сессию.
* Просмотреть справку с обучением.
* Посмотреть статистику

**Требования к применению:** интерфейс должен быть интуитивно понятен, правила игры должны быть объяснены

**Требования к производительности:** игра должна запускаться на большинстве компьютеров.

**Требования к реализации:** игра должна быть реализована с использованием языка программирования C#.

**Требования к надёжности:** надёжность должна быть на высоком уровне, не допуская возможность ошибок.

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

Используя таблицы определим самую подходящую стратегию разработки и модель жизненного цикла.

В таблице 1 рассмотрим выбор модели ЖЦ на основе характеристик требований

Таблица 1 **–** Выбор модели жизненного цикла на основе характеристик требований

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 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 |
| Продолжение таблицы 1 | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 7 Нужно ли реализовать основные требования на ранних этапах разработки? | Нет | Нет | Да | Да | Да | Да |

Вычисления: 0 за каскадную, 0 за V- образную, 3 за RAD, 3 за инкрементную, 7 за быстрого прототипирования и 7 за эволюционную.

На основе результатов заполнения таблицы 1 подходящими являются модели быстрого прототипирования и эволюционная.

В таблице 2 рассмотрим выбор модели ЖЦ на основе характеристик команды разработчиков

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 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 лёгкость 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: 3 за каскадную, 3 за V-образную, 3 за RAD, 1 за инкрементную, 4 за быстрого прототипирования и 5 за эволюционную.

На основе результатов заполнения таблицы 1 подходящими являются модели быстрого прототипирования и эволюционная.

В таблице 3 рассмотрим выбор модели ЖЦ на основе характеристик коллектива пользователей

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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: 1 за каскадную, 1 за V-образную, 1 за RAD, 2 за инкрементную, 3 за быстрого прототипирования и 3 за эволюционную.

На основе результатов заполнения таблицы 3 подходящими являются модели быстрого прототипирования и эволюционная.

В таблице 4 рассмотрим выбор модели ЖЦ на основе характеристик типа проектов и рисков

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 надёжности 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: 3 за каскадную, 4 за V-образную, 2 за RAD, 8 за инкрементную, 6 за быстрого прототипирования и 8 за эволюционную.

На основе результатов заполнения таблицы 4 подходящими являются инкрементная и эволюционная модели.

Итого: 7 за каскадную, 8 за V-образную, 9 за RAD, 14 за инкрементную, 20 за быстрого прототипирования и 23 за эволюционную.

На основе результатов заполнения таблиц 1-4 самой подходящей моделью жизненного цикла является эволюционная.

# 1.3 Инструменты разработки

Для разработки данного проекта выбрана игровая платформа Unity. Эта игровая платформа хорошо подходит для разработки видеоигр. В этой игровой платформе используется язык программирования C#.

Также для разработки программы необходимы:

* Google-браузер – нужен для нахождения информации.
* Word 2021 – нужен для написания пояснительной записки;
* Power Point 2021 – нужен для создания отчётной презентации;
* Aseprite – нужен для создания и редактирования пиксельных изображений;
* Blender 3.6 – нужен для создания и анимирования 3D моделей.
* Smart Install Maker – нужен для создания установщика.

При разработке данного программного продукта был использован ноутбук Lenovo со следующими характеристиками:

* Процессор: AMD Ryzen 5;
* ОЗУ: 8Gb;
* Память: Samsung SSD 256Gb;
* ОС – Windows 10.

**2 Проектирование задачи**

# 2.1 Организация данных

При разработке проекта будет использоваться база данных, модель которой представлена на рисунке 1.

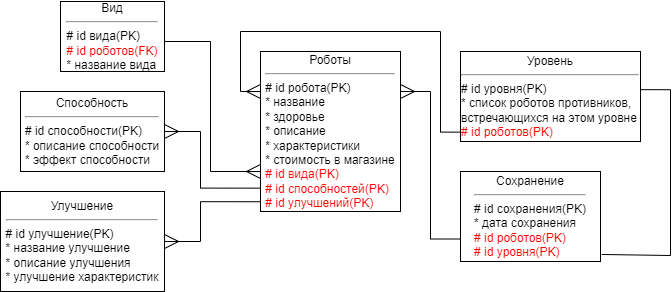


Рисунок 1 – Модель базы данных

В этой базе данных будет храниться информация об роботах, их вида, способностях и улучшения, а также о текущем уровне и сохранении.

# 2.2 Разработка UML-диаграмм

При разработке данного проекта были созданы диаграммы варианта использования, последовательности, деятельности и диаграмма классов.

Диаграмма вариантов использования представлена на рисунке 2.

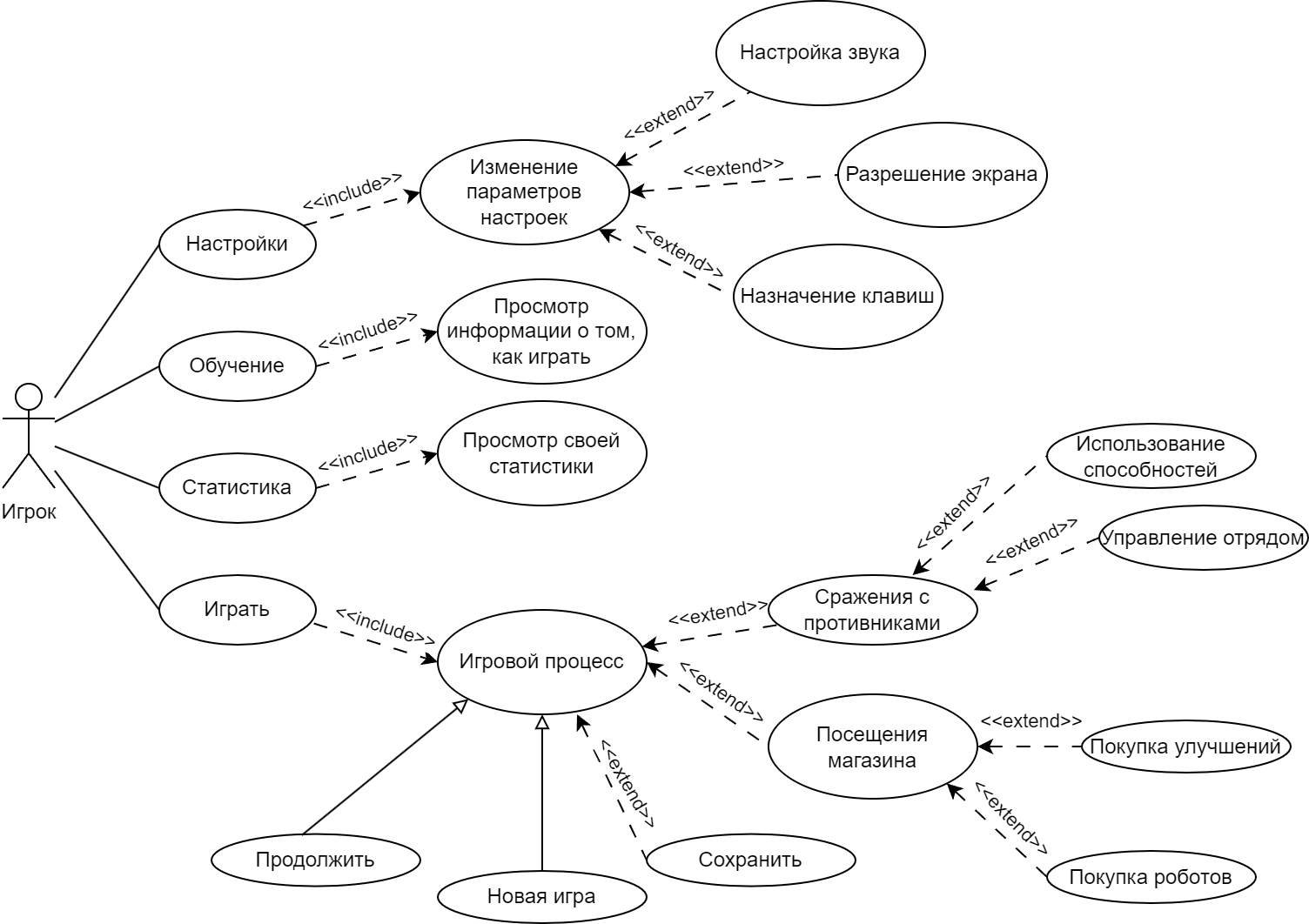


Рисунок 2 – Диаграмма вариантов использования

В этой диаграмме есть один актёр – игрок. Игрок имеет возможность открыть обучение, статистику, настройки или играть. Игрок может начать новую игру или продолжить последнюю сохранённую. Во время игры пользователь будет сражаться с врагами и посещать магазин. В магазине пользователь покупает новые улучшения и роботов. Во время битвы с противниками игрок управляет своим отрядом и использует способности. Так же игрок может сохранить игру в любой момент во время игрового процесса

На рисунке 3 представлена диаграмма последовательности.

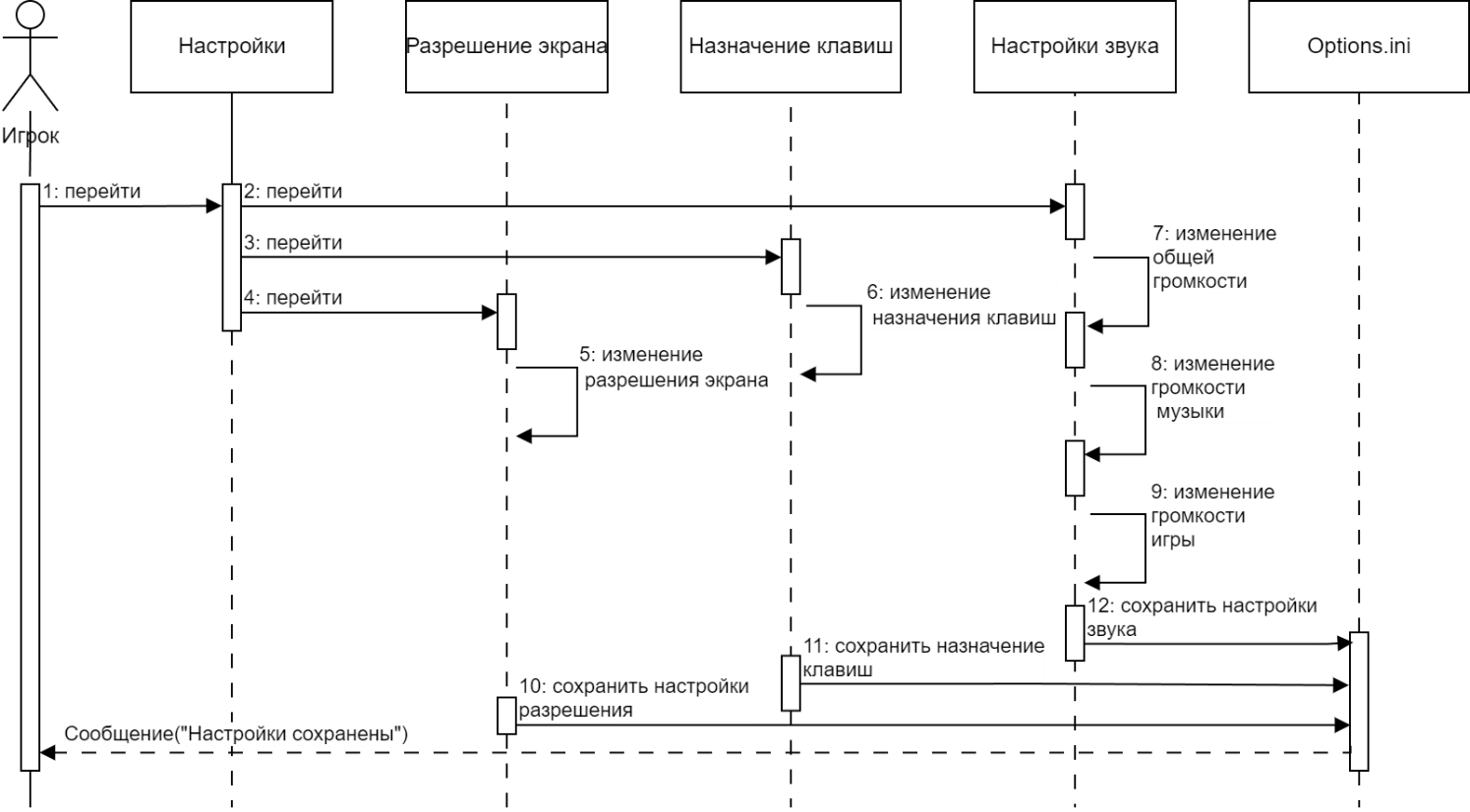


Рисунок 3 – Диаграмма последовательности

В диаграмме последовательности показано перемещение игрока по настройкам. В настройках игрок может поменять разрешение экрана, назначение клавиш и настроить звук.

На рисунке 4 представлена диаграмма деятельности.

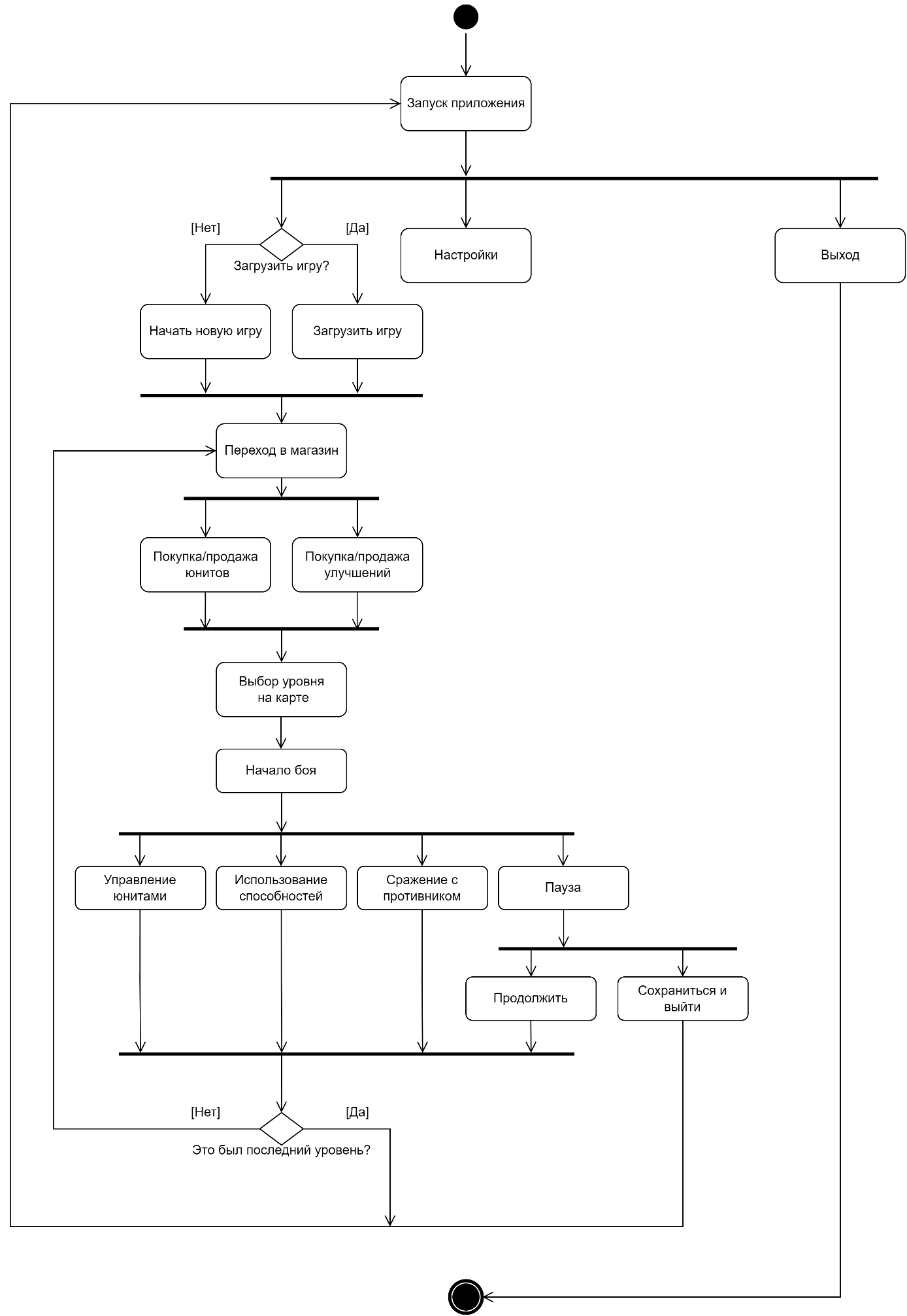


Рисунок 4 – Диаграмма деятельности

В диаграмме деятельности показан игровой процесс.

На рисунке 5 представлена диаграмма классов.

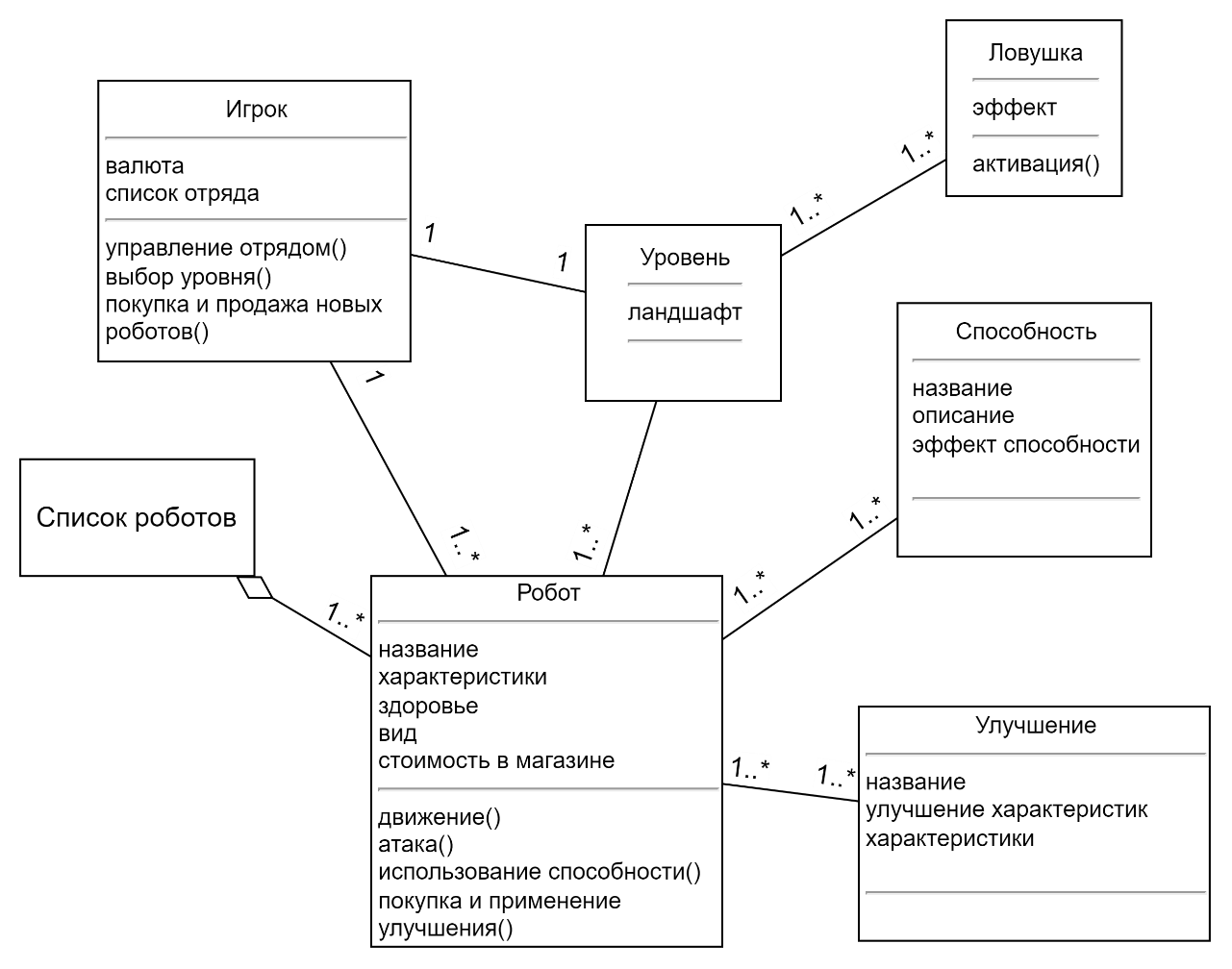


Рисунок 5 – Диаграмма классов

В диаграмме классов показаны поля и методы классов, которые будут в проекте.

**2.3 Разработка пользовательского интерфейса**

В прототипе был реализован главный игровой экран. Пользовательский интерфейс этого экрана представлен на рисунке 6.



Рисунок 6 – Интерфейс главного игрового экрана

На рисунках 7-8 представлен интерфейс магазина и карты в виде схем.

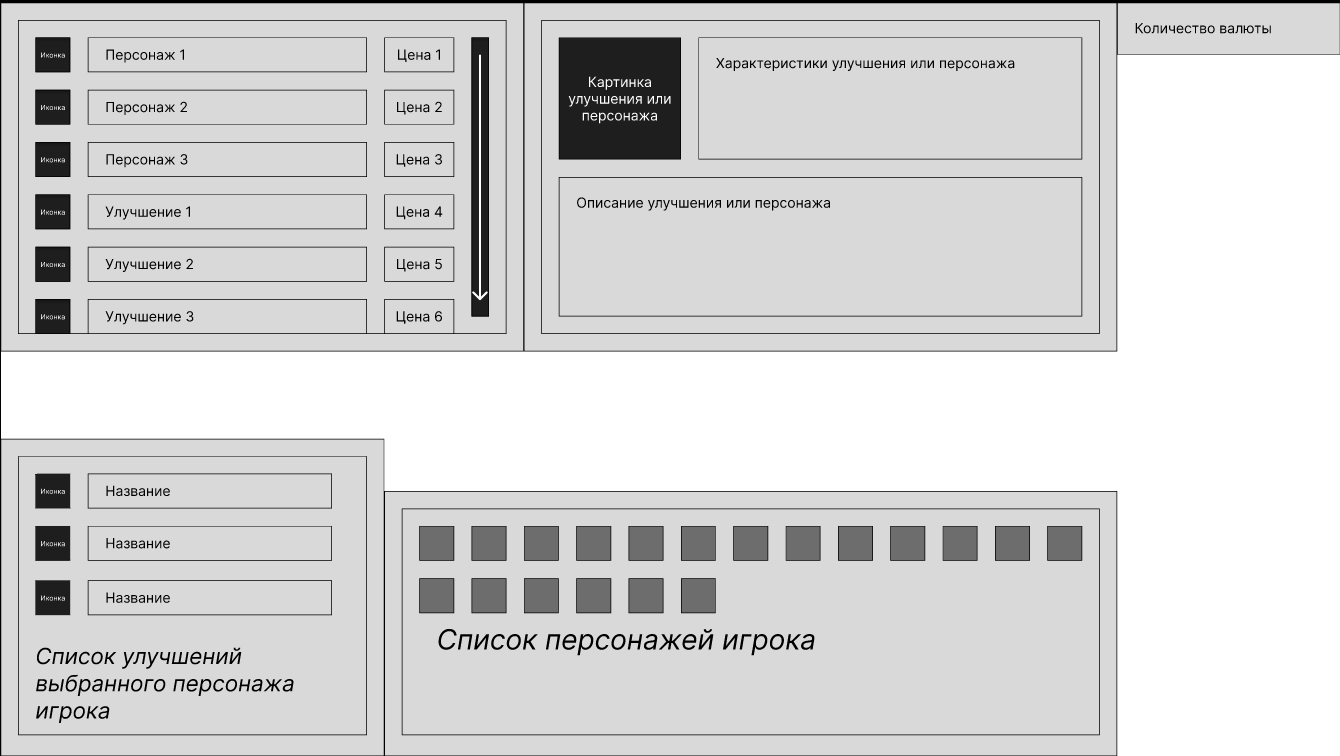


Рисунок 7 – Схема интерфейса в магазине

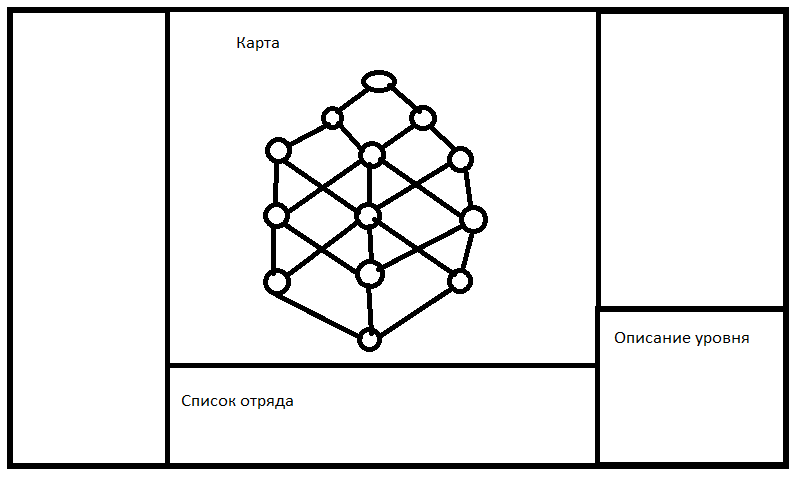
****

Рисунок 8 – Схема интерфейса на карте

Интерфейс проекта был создан в пиксельной стилистике. Для разработки интерфейса использовались различные оттенки серого, которые стилизуют интерфейс под футуристичный стиль.

Для надписей использовался шрифт OldTimer белого и чёрного цвета.

## 3 Реализация

### 3.1 Руководство программиста

### 3.1.1 Организация данных

В прототипе проекта код содержится в 5 файлах: EnemyCode, PlayerBullet, PlayerMovement, ShotButtonPress, TimerScript. Пользователь взаимодействует с этими файлами, вводя входную информацию в виде нажатий на кнопки и клавиши клавиатуры. Этот код управляет персонажам игрока и противниками. Все существующие противники сохраняются в список EnemyList.

**3.1.2 Структура программы**

Каждый файл кода прикрепляется к игровому объекту и выполняет функции. Связанные с этим объектом. Например, PlayerMovement имеет процедуру, позволяющую игроку перемещаться, а TimerScript отсчитывает время и создаёт новых противников через определённые интервалы. Эти файлы также имеют функции, которые связаны друг с другом, например, пуля при прикосновении к врагу или игроку наносит им урон.

**3.1.3 Структура и описание процедур и функций пользователя**

Описание всех разработанных функций и процедур приведено в таблице 5.

Таблица 5 ­ процедуры и функции

|  |  |  |  |
| --- | --- | --- | --- |
| Имя процедуры(функции) | В каком файле находится | За каким игровым объектом закреплена | Описание |
| 1 | 2 | 3 | 4 |
| 1 Start() | PlayerMovement | Игрок | Вызывается при появлении игрока, устанавливает его здоровье и прикрепляет аниматор |
| 2 FixedUpdate() | PlayerMovement | Игрок | Активируется 50 раз в секунду. Если игрок не целится, то вызывает процедуру движения, а если целится, то вызывает процедуру стрельбы. Если у игрока здоровье падает до 0 то заканчивает игру поражением |
| 3 PlayerMove() | PlayerMovement | Игрок | Поворачивает персонажа игрока в ту сторону, куда он нажимает и двигает его вперёд. Не позволяет выйти за границы. Перемещает камеру за персонажем игрока. |

Продолжение таблицы 5

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 4 PlayerShootMode | PlayerMovement | Игрок | Определяет куда показывает курсор и поворачивает персонажа игрока в ту сторону, создает красную линию между персонажем игрока и курсором. |
| 5. PlayerShoot | PlayerMovement | Игрок | Создаёт пулю, летящую в том направлении куда смотрит игрок |
| 6 Start() | EnemyCode | Противник | Вызывается при появлении противника, устанавливает его здоровье и запускает повторяющиеся процедуры стрельбы и перемещения |
| 7 FixedUpdate() | EnemyCode | Противник | Активируется 50 раз в секунду. Вызывает процедуру перемещения противника. Уничтожает его если здоровье падает до 0 |
| 8 EnemyMove() | EnemyCode | Противник | Перемещает противника в нужную сторону если он двигается. Поворачивает его в сторону игрока, когда стреляет |
| 9 MoveState() | EnemyCode | Противник | Каждые 2 секунды меняет состояние перемещения противника, случайно определяет напривление |
| 10 EnemyShoot() | EnemyCode | Противник | Поворачивает противника в сторону игрока и создает пулю |
| 11 OnDestroy() | EnemyCode | Противник | Вызывается при уничтожении противника. Уменьшает счётчик на 1 и убирает противника из списка |
| 12 Start() | PlayerBullet | Пуля | Вызывается при появлении пули. Вызывает функцию уничтожения |
| 13 FixedUpdate() | PlayerBullet | Пуля | Активируется 50 раз в секунду. Перемещает пулю вперёд |

Продолжение таблицы 5

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 14 OnTriggerEnter() | PlayerBullet | Пуля | Активируется при касании персонажа игрока или противника. Наносит урон тому, кого коснулся и исчезает |
| 15 BulletDestroy | PlayerBullet | Пуля | Уничтожает пулу через определённое время |
| 16 ShootingMode() | ShootButtonPress | Кнопка стрельбы | Переключает игрока в режим прицеливания |
| 17 Start() | TimerScript | Таймер | Устанавливает счётчик оставшихся противников. Вызывает процедуры отсчёта времени и обновления счётчика врагов |
| 18 TimerCount() | TimerScript | Таймер | Отсчитывает время, создаёт нового противника каждые 5 секунд |
| 19 EnemiesCountUpdate() | TimerScript | Таймер | Обновляет счётчик врагов. Если счётчик равен нулю то заканчивает игру победой. |

**3.1.4 Описание использованных компонентов**

Описание использованных при разработке игровых объектов будет приведено в таблице 6.

Таблица 6 ­ Использованные объекты

|  |  |
| --- | --- |
| Объект | Назначение |
| 1 MainCamera | Камер |
| 2 Directional Light | Свет |
| 3 Plane | Поверхность, по которой перемещаются противники и игрок |
| 4 UICanvas | Объект, содержащий элементы пользовательского интерфейса |
| 5 EventSystem | Объект, вызывающий определённые процедуры при нажатии на определённые кнопки |
| 6 ShootingLine | Красная линия между персонажем игрока и курсором |
| 7 Player | Персонаж игрока |

**3.1.5 Спецификация программы**

Все файлы, использовавшиеся в процессе разработки хранятся в папке Assets.Они перечислены в таблице 7

Таблица 7 – файлы в папке Assets

|  |  |
| --- | --- |
| Название папки или файла | Описание |
| 1 EnemyCode | Код противника |
| 2 PlayerBullet | Код пули |
| 3 PlayerMovement | Код игрока |
| 4 ShootButtonPress | Код кнопки стрельбы |
| 5 TimerScript | Код таймера |
| 6 BulletOrange | Материал пули |
| 7 EnemyRed | Материал противника |
| 8 PlaneMaterial | Материал плоскости |
| 9 SampleScene | Сцена |
| 10 UI Battle Bullet Button | Спрайт кнопки стрельбы |
| 11UI Battle Buttons Down | Спрайт нижней части пользовательского интерфейса |
| 12 UI Battle Frame sprite | Спрайт рамки пользовательского интерфейса |
| 13 UI Battle Level-Enemy count | Спрайт счётчика врагов и уровня |
| 14 UI Battle Money Count | Спрайт счётчика денег |
| 15 UI Battle Time Count | Спрайт таймера |
| 16 ACPlayer | Контроллер анимации персонажа игрока |
| 17 ColorPallet | Цветовая палитра для модели персонажа игрока |
| 18 Enemy | Заготовка противника |
| 19 ExplosionBig | Заготовка большого взрыва |
| 20 ExplosionSmall | Заготовка маленького взрыва |
| 21 Mech1 | Модель персонажа игрока |
| 22 OldTimer | Шрифт |
| 23 PlayerBullte | Заготовка пули |

**4 Тестирование**

**4.1 Отчёт о результатах тестирования**

При разработке данной программы большинство возникающих ошибок и недоработок были исправлены на этапе реализации проекта. После завершения испытания реализации программы было проведено тщательное функциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программы в автономном режиме.

Отчёт о результатах тестирования предоставлен в таблице 8.

Таблица 8 – Отчёт результатах тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Физический результат | Результат тестирования |
| Проверка запуска приложения | Приложение запустилось | Приложение запустилось | Выполнено |
| Проверка выхода | Приложение закрылось | Приложение закрылось | Выполнено |
| Проверка таймер | Время отсчитывается, противники появляются | Время отсчитывается, противники появляются | Выполнено |
| Перемещение игрока | Игрок перемещается в нужную сторону | Игрок перемещается в нужную сторону | Выполнено |
| Проверка стрельбы | Игрок переходит в режим стрельбы и стреляет | Игрок переходит в режим стрельбы и стреляет | Выполнено |
| Проверка счётчика врагов | При уничтожении врага счётчик уменьшается | При уничтожении врага счётчик уменьшается | Выполнено |
| Проверка получения урона игроком | При попадании по игроку пули он получает урон | При попадании по игроку пули он получает урон | Выполнено |
| Проверка получения урона противником | При попадании по противнику пули он получает урон | При попадании по противнику пули он получает урон | Выполнено |
| Проверка уничтожения врага | Враг уничтожился | Враг уничтожился | Выполнено |
| Проверка уничтожения игрока | Игрок уничтожился, поражение | Игрок уничтожился, поражение | Выполнено |
| Проверка победы | Когда противника закончились игра кончается победой | Когда противника закончились игра кончается победой | Выполнено |

При разработке программного продукта было решено множество проблем, например, некорректное поведение противников. После тестирования и исправления проблем пользователь не столкнётся с ими.

Элементы программы были проверены, и было установлено, что все они работают правильно и выполняют задачи, указанные в процедурах.

**5 Руководство пользователя**

# 5.1 Общие сведения о программном продукте

Цель данной игры заключается в развлечении пользователя и развитии его реакции.

Создаваемое игровое приложение будет рассчитано на любого рода пользователей и может использоваться в любой момент по желанию пользователя.

Быстродействие любой программы во многом зависит от характеристик выбранного персонального компьютера: рабочей частоты процессора, объема оперативной памяти и т.д. Несмотря на все реализованные в ней задачи, она легко запускается и функционирует на большинстве машинах.

Тестирование проводилось на разных классах ЭВМ и данное игровое приложение стабильно работало на любом персональном компьютере. Программа разработана на ноутбуке со следующими характеристиками:

− процессор AMD Ryzen 5 3500U with Vega mobile GFX, 2,1 ГГц, ядер:

4, логических процессоров: 8

− объем ОЗУ 8Гб;

− Графический процессор AMD Radeon Vega 8 Graphics;

− операционная система Windows 10.

# 5.2 Инсталляция

Для того, чтобы установить программу необходимо запустить файл Setup.exe. Появится окно установки приложения “Arena Dominus”.

Затем достаточно следовать приведённой инструкции установки приложения.

# 5.3 Выполнение программы

## 5.3.1 Запуск программы

Для запуска данного приложения необходимо запустить исполняемый файл Arena Dominus.exe, который находится в папке c игрой по пути, выбранному при установке.

## 5.3.2 Инструкции по работе с программой

После запуска приложения пользователь сразу попадает в начало игры (рисунок 9).



Рисунок 9 – Начало игры

Игрок может перемещаться, нажимая клавиши WASD (рисунок 10).

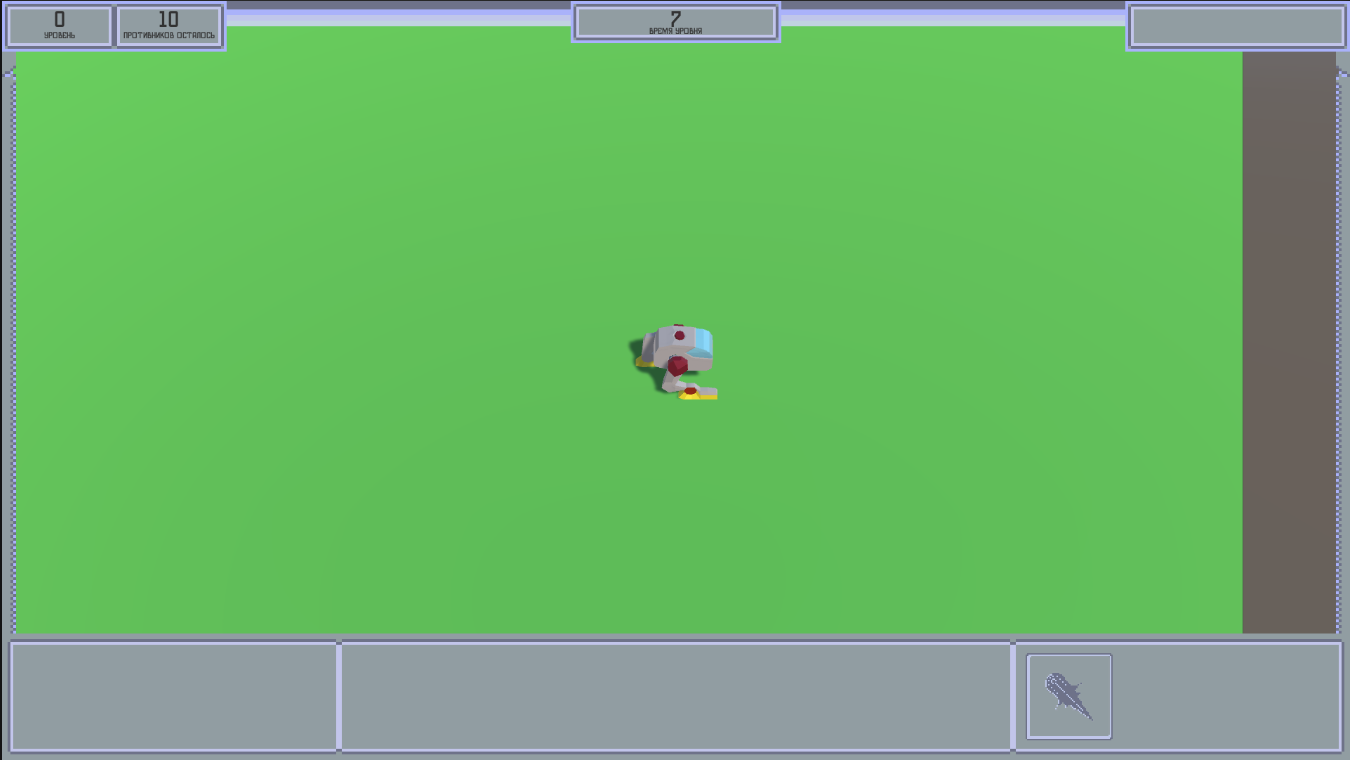


Рисунок 10 – Кнопка бросить

Так же игрок может войти в режим прицеливая. (рисунок 11). Когда он находится в этом режиме он может нажать левую кнопку мышки чтобы выстрелить

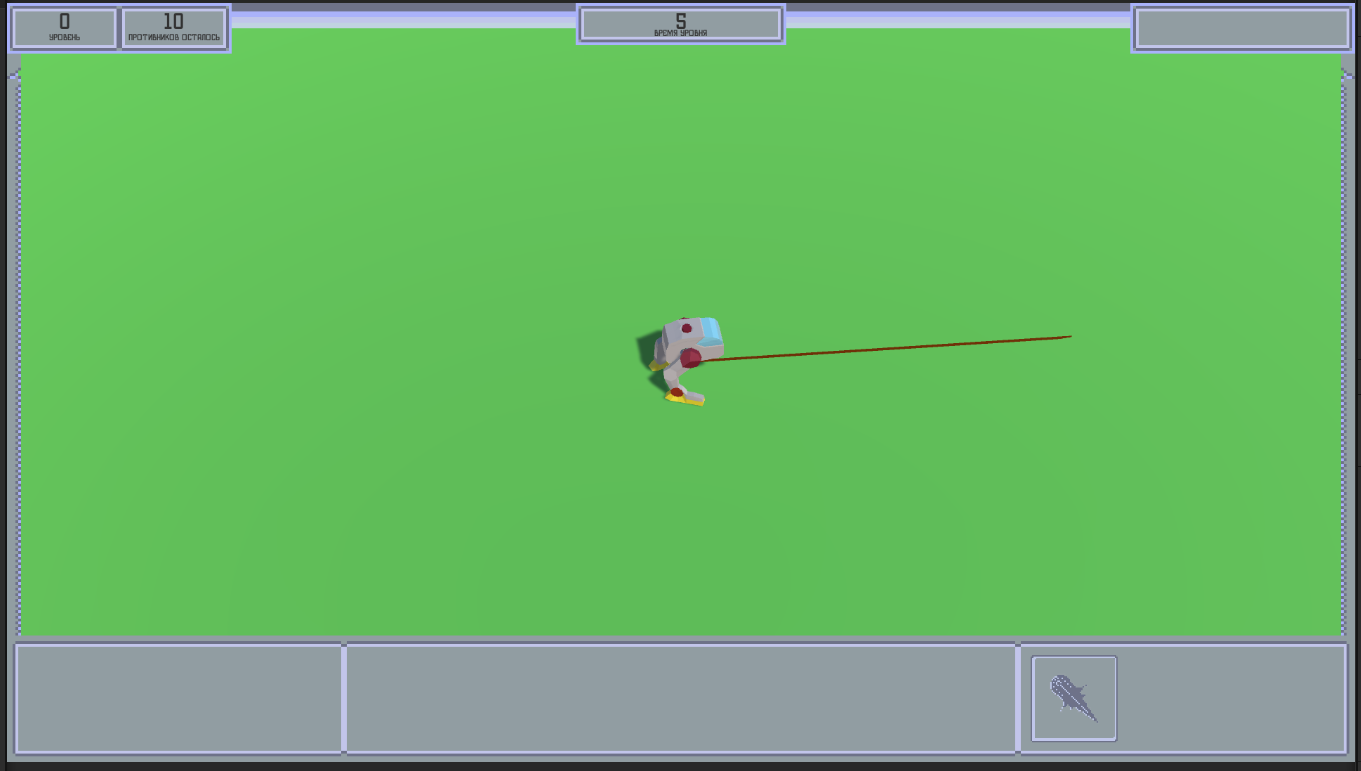


Рисунок 11 – Игрок прицеливается

Для того чтобы выиграть игрок должен уничтожить 10 противников. Противники получают урон при попадании по ним и имеют 2 очка здоровья (рисунок 12).

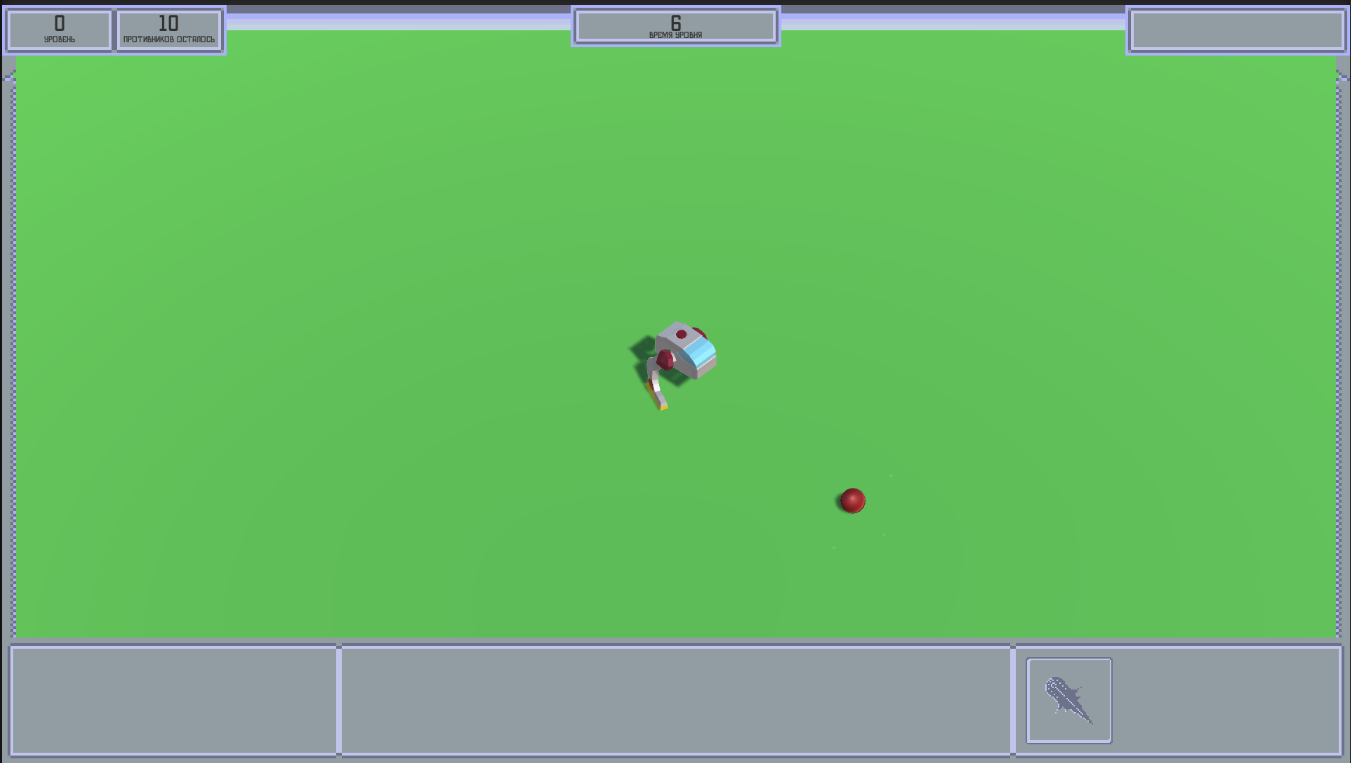


Рисунок 12 – Противник получает урон

Противники перемещаются вокруг и стреляют в игрока. После 5 попаданий игрок уничтожается и игра заканчивается поражением (рисунок 13).



Рисунок 13 – Экран поражения

**Заключение**

Для выполнение данного проекта была поставлена задача разработать игровое приложение “Arena Dominus”, целью которого является развлечение пользователя, развитие его реакции и привлечение его внимания к индустрии видеоигр.

При выполнении данного проекта возникали ошибки, все из которых были исправлены и этапе реализации или тестирования. Поставленную задачу можно считать выполненной успешно.

При разработке данного приложения использование сторонних материалов было минимальным, в результате чего все составляющие проекта, кроме компонентов и библиотек Unity, были разработаны самостоятельно.

Решения, принятые во время разработки проекта, полностью соответствуют поставленной задаче.

В документации был представлен прототип проекта, передающий только малую часть всего игрового процесса.

**Список использованных источников**

1. Сайт с документацией Unity3D DocsUnity3d [Электронный ресурс]. – Режим доступа: https:// https://docs.unity3d.com. – Дата доступа: 03.01.2024
2. Форум об Unity ForumUnity3d [Электронный ресурс]. – Режим доступа: https://forum.unity.com. – Дата доступа: 03.01.2024
3. Форум программистов StackOverflow [Электронный ресурс]. – Режим доступа: https://stackoverflow.com. – Дата доступа: 03.01.2024
4. Брауде, Технология разработки программного обеспечения / Э. Брауде – СП6: Петербург, 2004

**Приложение А**

Функциональная модель

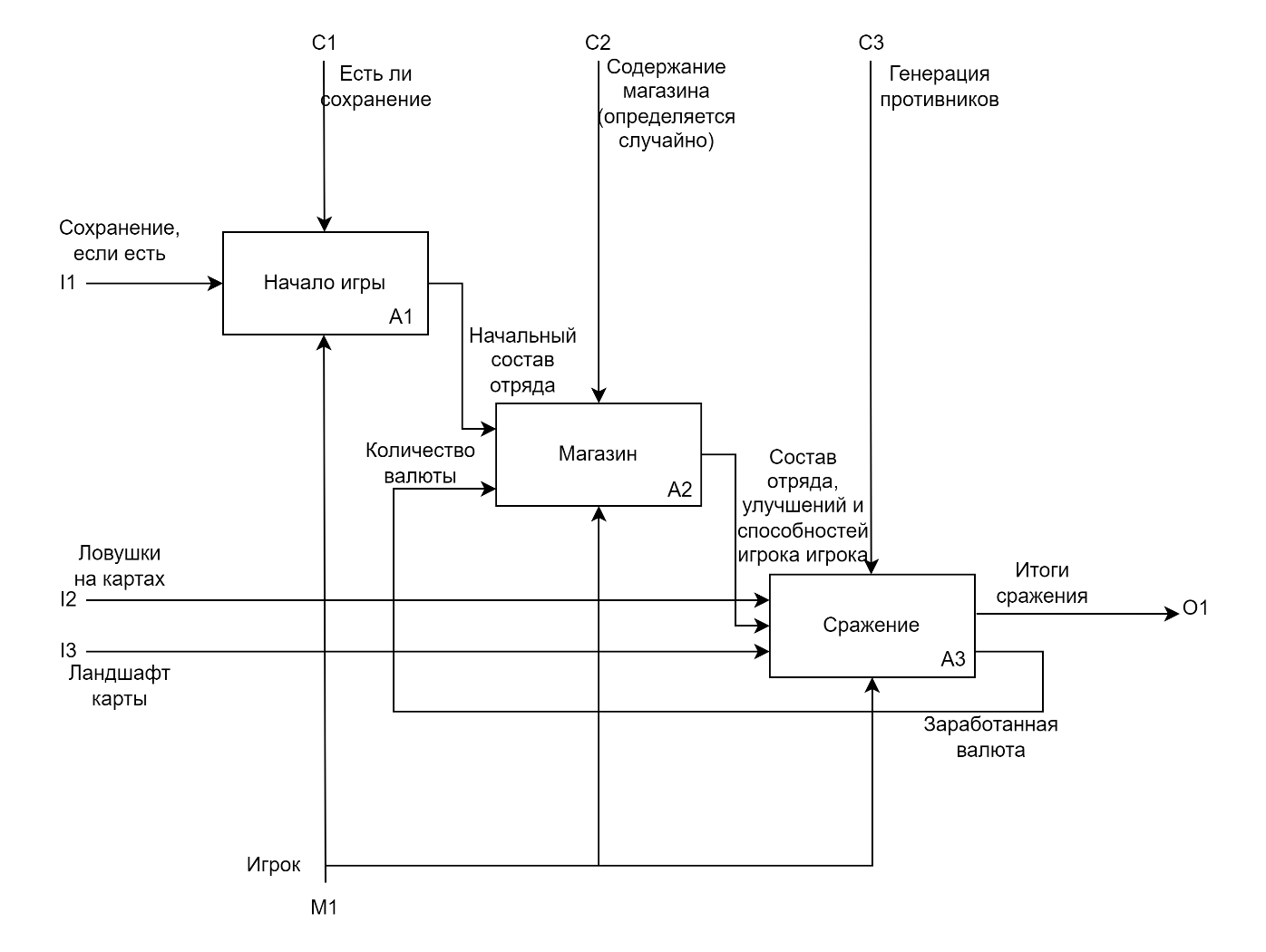


Рисунок 14 ­ Общий вид модели

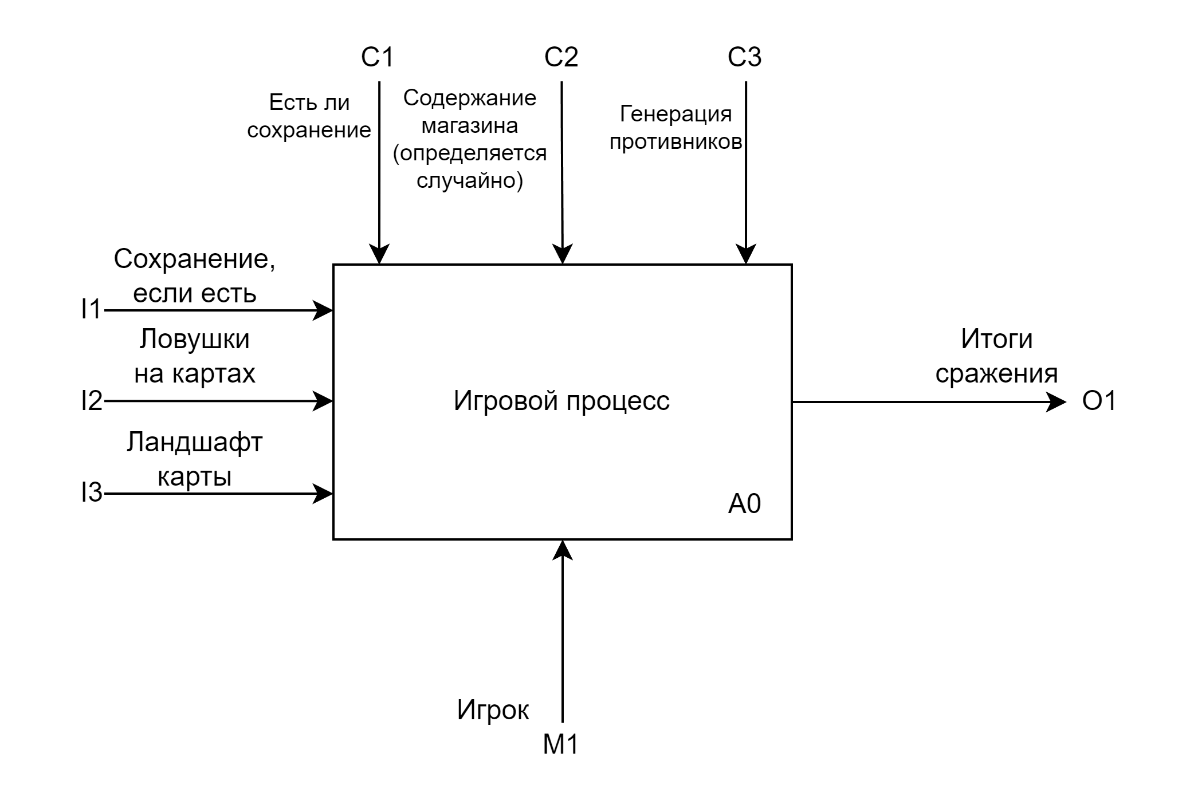


Рисунок 15 ­ Сокращённый вид функциональной модели

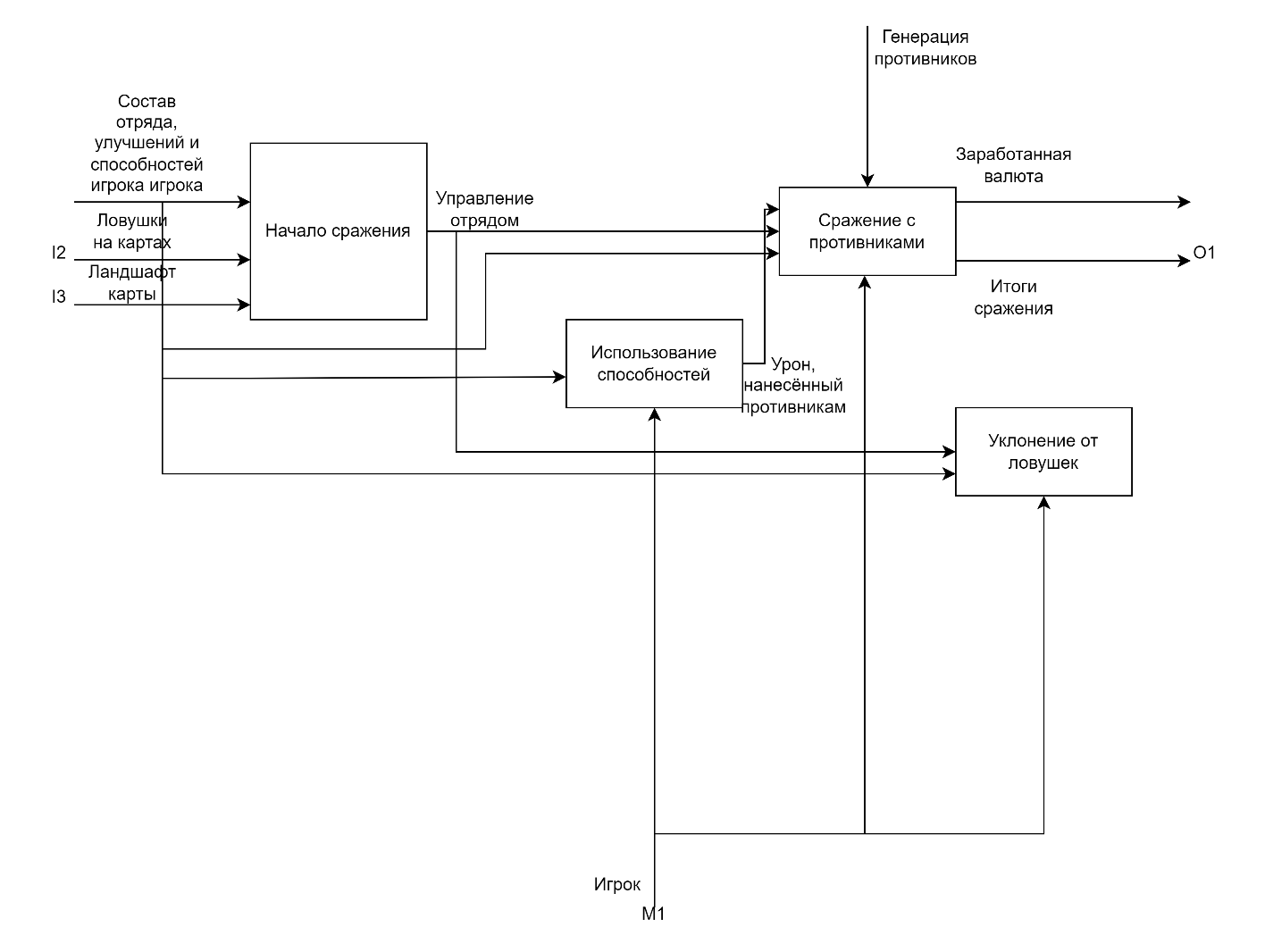


Рисунок 16 ­ Подробная функциональная модель сражения

**Приложение Б**

Листинг приложения

//Код игрока

using System;

using UnityEngine;

using UnityEngine.UI;

public class PlayerMovement : MonoBehaviour

{

[SerializeField] private float xMove;

[SerializeField] private float zMove;

[SerializeField] private float rot;

public float speed = 10f;

public float Rotatespeed = 10f;

public static bool Shooting = false;

public LayerMask PlaneLayer;

public GameObject ShootingLine;

public GameObject SB;

private GameObject ShotPlayerBullet;

public GameObject Explosion;

public GameObject EndText;

public int health = 5;

private Animator animator;

public GameObject cam;

private void Start()

{

animator = gameObject.GetComponent<Animator>();

health = 5;

}

void FixedUpdate()

{

if (Shooting == false)

{

PlayerMove();

}

else

{

PlayerShootMode();

}

if (health <= 0)

{

GameObject Expl = Instantiate(Explosion, gameObject.transform.position, gameObject.transform.rotation);

Destroy(Expl, 2f);

EndText.GetComponent<Text>().text = "Поражение";

gameObject.SetActive(false);

}

}

void PlayerMove()

{

xMove = Input.GetAxisRaw("XMove");

zMove = Input.GetAxisRaw("ZMove");

if ((xMove == 1) && (gameObject.transform.position.x >= 50))

xMove = 0;

if ((xMove == -1) && (gameObject.transform.position.x <= -50))

xMove = 0;

if ((zMove == 1) && (gameObject.transform.position.z >= 50))

zMove = 0;

if ((zMove == -1) && (gameObject.transform.position.z <= -50))

zMove = 0;

if (xMove != 0 || zMove != 0)

{

animator.SetBool("Walking", true);

transform.Translate(new Vector3(0, 0, 1) \* speed \* Time.fixedDeltaTime, Space.Self);

Quaternion ToRotate = Quaternion.LookRotation(new Vector3(xMove, 0, zMove), Vector3.up);

transform.rotation = Quaternion.RotateTowards(transform.rotation, ToRotate, Rotatespeed \* Time.deltaTime);

if (animator.GetBool("RotateBool") == true) { animator.SetBool("RotateBool", false); }

}

else { animator.SetBool("Walking", false); }

cam.GetComponent<Transform>().position = new Vector3(GetComponent<Transform>().position.x, 15, GetComponent<Transform>().position.z - 15);

}

void PlayerShootMode()

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

if (Physics.Raycast(ray, out hit, 1000f, PlaneLayer))

{

if (animator.GetBool("RotateBool") == false) { animator.SetBool("RotateBool", true); }

Vector3 NewDirection = Vector3.RotateTowards(transform.forward, hit.point - transform.position, 500 \* Time.deltaTime, 0f);

NewDirection.y = 0;

rot = (transform.rotation.y - Quaternion.LookRotation(NewDirection).y)\*10;

rot = (float)Math.Round((decimal)rot, 4, MidpointRounding.AwayFromZero);

if (rot < 0) { animator.SetBool("RotateSide", false); }

else if (rot > 0) { animator.SetBool("RotateSide", true); }

animator.SetFloat("Rotate", rot);

transform.rotation = Quaternion.LookRotation(NewDirection);

ShootingLine.GetComponent<LineRenderer>().SetPositions(new[] { transform.position, hit.point });

ShootingLine.SetActive(true);

if (Input.GetMouseButton(0) == true) PlayerShoot();

}

}

void PlayerShoot()

{

Shooting = false;

ShootingLine.SetActive(false);

ShotPlayerBullet = SB;

ShotPlayerBullet.GetComponent<PlayerBullet>().PlayerType = true;

Instantiate(ShotPlayerBullet, new Vector3(transform.position.x, 0.5f, transform.position.z), transform.rotation);

}

}

//Код противника

using System.Collections;

using UnityEngine;

public class EnemyCode : MonoBehaviour

{

[SerializeField] private float xMove;

[SerializeField] private float zMove;

public float speed = 5f;

public float Rotatespeed = 250f;

[SerializeField] bool move = false;

private GameObject ShotEnemyBullet;

public GameObject SB;

public GameObject player;

public GameObject Explosion;

public GameObject TimerCount;

public int health = 2;

void Start()

{

player = GameObject.Find("Player");

health = 2;

StartCoroutine(MoveState());

StartCoroutine(Shoot());

}

void FixedUpdate()

{

EnemyMove();

if (health <= 0)

{

GameObject Expl = Instantiate(Explosion, gameObject.transform.position, gameObject.transform.rotation);

Destroy(Expl, 2f);

Destroy(gameObject);

}

}

void EnemyMove()

{

if (move == true)

{

if ((xMove == 1) && (gameObject.transform.position.x >= 50))

xMove = 0;

if ((xMove == -1) && (gameObject.transform.position.x <= -50))

xMove = 0;

if ((zMove == 1) && (gameObject.transform.position.z >= 50))

zMove = 0;

if ((zMove == -1) && (gameObject.transform.position.z <= -50))

zMove = 0;

transform.Translate(new Vector3(xMove, 0, zMove).normalized \* speed \* Time.fixedDeltaTime, Space.World);

}

if (xMove != 0 || zMove != 0)

{

Quaternion ToRotate = Quaternion.LookRotation(new Vector3(xMove, 0, zMove), Vector3.up);

transform.rotation = Quaternion.RotateTowards(transform.rotation, ToRotate, Rotatespeed \* Time.deltaTime);

}

}

private IEnumerator MoveState()

{

while (true)

{

yield return new WaitForSeconds(2f);

move = !move;

if (move == true)

{

xMove = UnityEngine.Random.Range(-1, 2);

zMove = UnityEngine.Random.Range(-1, 2);

}

}

}

private IEnumerator Shoot()

{

while (true)

{

yield return new WaitForSeconds(UnityEngine.Random.Range(2,6));

EnemyShoot();

}

}

private void EnemyShoot()

{

Vector3 NewDirection = Vector3.RotateTowards(transform.forward, player.transform.position - transform.position, 500 \* Time.deltaTime, 0f);

NewDirection.y = 0;

transform.rotation = Quaternion.LookRotation(NewDirection);

ShotEnemyBullet = SB;

ShotEnemyBullet.GetComponent<PlayerBullet>().PlayerType = false;

Instantiate(ShotEnemyBullet, transform.position, transform.rotation);

}

private void OnDestroy()

{

GameObject TimerCount = GameObject.Find("TimerCount");

TimerCount.GetComponent<TimerScript>().EnemiesLeft -= 1;

TimerCount.GetComponent<TimerScript>().EnemiesCountUpdate();

TimerCount.GetComponent<TimerScript>().Enemies.Remove(gameObject);

}

}

//Код таймера

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class TimerScript : MonoBehaviour

{

public int Count;

public GameObject Enemy;

public GameObject EnemiesCount;

public List<GameObject> Enemies;

public int EnemiesLeft;

public GameObject EndText;

void Start()

{

EnemiesLeft = 10;

EnemiesCountUpdate();

StartCoroutine(TimerCount());

}

IEnumerator TimerCount()

{

yield return new WaitForSecondsRealtime(1);

Count += 1;

gameObject.GetComponent<Text>().text = Count.ToString();

if ((Count % 5 == 0) && (Enemies.Count < 5) && (Enemies.Count < EnemiesLeft))

{

Enemies.Add(Instantiate(Enemy, new Vector3(Random.Range(-20,20),0.5f, Random.Range(-20, 20)), new Quaternion(0,0,0,0)));

}

StartCoroutine(TimerCount());

}

public void EnemiesCountUpdate()

{

EnemiesCount.GetComponent<Text>().text = EnemiesLeft.ToString();

if (EnemiesLeft == 0)

EndText.GetComponent<Text>().text = "Победа";

}

}

//Код пули

using System.Collections;

using UnityEngine;

public class PlayerBullet : MonoBehaviour

{

public float speed = 50f;

public bool PlayerType = true;

public GameObject HitExplosion;

private void Start()

{

StartCoroutine(BulletDestroy());

}

private void FixedUpdate()

{

transform.Translate(new Vector3(0, 0, 1) \* speed \* Time.fixedDeltaTime, Space.Self);

}

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Player") && PlayerType == false)

{

other.GetComponent<PlayerMovement>().health -= 1;

GameObject Expl = Instantiate(HitExplosion, other.transform.position, other.transform.rotation);

Destroy(Expl, 1f);

Destroy(gameObject);

}

else if (other.CompareTag("Enemy") && PlayerType == true)

{

other.GetComponent<EnemyCode>().health -= 1;

other.GetComponent<Renderer>().materials[0].SetColor("\_Color", new Color(0.5f \* other.GetComponent<EnemyCode>().health, 0, 0));

GameObject Expl = Instantiate(HitExplosion, other.transform.position, other.transform.rotation);

Destroy(Expl, 1f);

Destroy(gameObject);

}

}

private IEnumerator BulletDestroy()

{

yield return new WaitForSeconds(5f);

Destroy(gameObject);

}

}

//Код нажатия кнопки стрельбы

using UnityEngine;

public class ShootButtonPress : MonoBehaviour

{

public void ShootingMode()

{

PlayerMovement.Shooting = true;

}

}