|  |  |  |
| --- | --- | --- |
| Пермский филиал федерального государственного автономного  образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики» | | |
| *Факультет экономики, менеджмента и бизнес-информатики* | | |
| Шевчук Михаил Романович | | |
| **ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ИГРЫ С РАЗРУШАЕМЫМ ЛАНДШАФТОМ** | | |
| *Выпускная квалификационная работа* | | |
| по направлению подготовки *09.03.04 Программная инженерия* образовательная программа «Программная инженерия» | | |
| Рецензент |  | Научный руководитель  старший преподаватель  кафедры  информационных  технологий в бизнесе  Г. И. Рустамханова    Консультант  старший преподаватель  кафедры  информационных  технологий в бизнесе  В. В. Лебедев |
| Пермь, 2020 год | | |

Аннотация

Выпускная квалификационная работа «Проектирование и разработка игры с разрушаемым ландшафтом» подготовлена студентом НИУ ВШЭ – Пермь Шевчуком Михаилом. В работе рассматриваются теоретические и практические вопросы, касающиеся подхода к деформированию ландшафта в играх и разработки 3D игры в жанре шутер с механикой деформации ландшафта. В первой главе приводится анализ использования механики деформации ландшафта в играх и анализ актуальных исследований на эту тему. Во второй главе описывается проектирование разрабатываемой игры и её основных геймплейных механик. Далее описывается разработка и тестирование приложения.

Работа содержит \_ страниц формата А4 основного текста, включая 4 главы: анализ использования механики деформации ландшафта в играх, проектирование игровых компонентов и механик, их реализация и тестирование.

В основной части содержится \_ рисунков и \_ диаграмм, описывающих процессы, выполняемые системой.

Библиографический список состоит из \_ источников.

Оглавление

[Введение 5](#_Toc67774157)

[Глава 1. Анализ использования механики деформации ландшафта в играх 8](#_Toc67774158)

[1.1. Анализ существующих игр в жанре шутер с механикой деформации ландшафта 8](#_Toc67774159)

[1.1.1. Worms 3D (2003) 9](#_Toc67774160)

[1.1.2. Scorched 3D (2001) 10](#_Toc67774161)

[1.1.3. Battlefield 3 (2011) 11](#_Toc67774162)

[1.1.4. Сравнительная характеристика игр с механикой деформации ландшафта 12](#_Toc67774163)

[1.2. Анализ существующих методов генерации деформируемого ландшафта 13](#_Toc67774164)

[1.2.1. Генерация ландшафта с помощью полигональной сетки 13](#_Toc67774165)

[1.2.2. Генерация ландшафта с помощью вокселей 17](#_Toc67774166)

[1.3. Выявление требований к разрабатываемой игре 21](#_Toc67774167)

[1.3.1. Описание функциональных и нефункциональных требований к игре 22](#_Toc67774168)

[Глава 2. Проектирование игровых компонентов и механик 24](#_Toc67774169)

[2.1. Конструирование архитектуры разрабатываемой игры 24](#_Toc67774170)

[2.2. Проектирование механики генерации TD 31](#_Toc67774171)

[2.3. Проектирование геймплея игрока 33](#_Toc67774172)

[Глава 3. Реализация игровых компонентов и механик 36](#_Toc67774173)

[Глава 4. Тестирование игрового процесса 37](#_Toc67774174)

[Заключение 38](#_Toc67774175)

[Библиографический список 39](#_Toc67774176)

[Приложение А. Техническое задание 41](#_Toc67774177)

[Приложение Б. Технико-экономическое обоснование проекта 52](#_Toc67774178)

Введение

Рынок компьютерных игр с каждым годом набирает новые обороты. Всё больше крупных компаний, таких как BMW, Intel, Nike, Xiaomi, инвестируют в киберспорт миллионы долларов. В период пандемии COVID-19 индустрия компьютерных игр стала одной из немногих отраслей, которая не только существенно не пострадала от карантинных ограничений, но и показала рост прибыли и количества клиентов. К примеру, в Китае выручка рынка видеоигр за первые месяцы 2020 года выросла на 25,2% по сравнению с концом 2019, а прибыль с мобильных игр увеличилась на 37,6%, что говорит о том, что индустрия в ближайшее время будет только расширяться [].

Особенностью компьютерных игр является разнообразие жанров и механик игрового процесса. Одной из таких механик является механика «разрушаемого окружения» (destructible environment, далее DE). Под DE подразумеваются игровые объекты, которые могут быть разрушены или деформированы в результате действий игрока. Это могут быть строения, предметы интерьера, растительность и другие искусственные, либо природные структуры. Способность игрока разрушить стену или препятствие в лабиринте и таким образом создать себе новый вариант пути является примером механики DE.

Существует отдельный подвид DE – «деформируемый ландшафт» (terrain deformation, далее TD), имеющий отличия и сложности в плане реализации. Если при разрушении объекта, находящего на ландшафте, на программном уровне достаточно просто заменить его текстуру или удалить из памяти, то при деформации самого ландшафта меняется его геометрия. К примеру, деформация исходного плоского полигона поверхности приводит к разбиению его на множество полигонов, больше не лежащих в его плоскости. При этом постоянное динамическое изменение ландшафта создаёт большую нагрузку на аппаратные ресурсы компьютера. И по этой причине механика TD меньше всего используется в самом динамичном жанре игр – «шутер».

В данной работе представлена разработка и реализация игры в жанре «шутер» с использованием механики TD.

**Объектом исследования** является игра в жанре «шутер» с механикой деформации ландшафта, а **предметом исследования** является алгоритм генерации деформируемого ландшафта для игр жанра «шутер».

**Цель** выпускной квалификационной работы – разработка исследовательского прототипа 3D игры в жанре «шутер» с разрушаемым ландшафтом. Для достижения поставленной цели нужно выполнить следующие задачи:

1. Проведение анализа предметной области:
   1. Анализ существующих игр, использующих механику TD.
   2. Анализ существующих методов генерации TD.
   3. Описание механик разрабатываемой игры.
   4. Выявление функциональных требований.
2. Проектирование игровых механик и игрового интерфейса:
   1. Проектирование механики генерации TD.
   2. Проектирование механики передвижения игрока.
   3. Проектирование механики взаимодействия игрока с окружением.
3. Реализация игровых механик и игрового интерфейса:
   1. Выбор игрового движка для реализации.
   2. Реализация игровых механик.
   3. Разработка игровой графики и пользовательского интерфейса.
4. Тестирование игровых механик.

Степень разработанности проблемы способствует выполнению поставленной цели, так как существуют актуальные исследования на тему динамически изменяемых ландшафтов, а также существуют примеры игр, использующих данную механику []. Практической значимостью игры является реализация метода с использованием нового для проблемы инструментального средства, что может привести к расширению спектра игровых механик и жанров на рынке в случае успеха продукта. Теоретической значимостью является доказательство сходимости алгоритмов и методов при решении проблемы генерации TD для динамических игр жанра «шутер».

В процессе работы предполагается использование следующих методов исследования:

1. Сравнение по аналогам и требованиям – рассмотрение существующих игр с подобными механиками и рассмотрение существующих способов реализации генерации TD.
2. Дискретизация – один из методов генерации TD – представление его не как одной поверхности, а в виде совокупности примитивов – вокселей.
3. Визуализация – рендеринг полученного результата на мониторе игроков.

# Анализ использования механики деформации ландшафта в играх

В первой главе рассматривается следующий список исследовательских вопросов:

1. Какие существуют игры, использующие механику TD? В каком жанре они созданы и как именно используют TD? Какие методы генерации TD при этом применяются?
2. Что из себя представляют методы TD? Как они реализуются? Какие имеют плюсы и минусы?
3. Какие обязательные механики должна включать в себя игра? Какие дополнительные механики дали бы ей преимущество перед другими играми?

Вопросы первого блока поднимаются с целью выявления основных паттернов шутер жанра, выявления требований к разрабатываемому прототипу и рассмотрения способов создания динамического ландшафта. Затем проводится сравнительный анализ выявленных методов на предмет их релевантности по отношению к жанру шутер. Наконец, приводится список функциональных и нефункциональных требований, составленный на основе рассмотрения основных механик игр жанра шутер и дополнения их новыми, согласующимися с выбранным методом деформации ландшафта.

## 1.1. Анализ существующих игр в жанре шутер с механикой деформации ландшафта

Для разработки 3D игры в жанре «шутер», использующей механику TD, требуется рассмотрение существующих решений с целью выделения:

* используемых способов создания разрушаемого ландшафта;
* общих паттернов жанра;

Проводить анализ существующих решений будем по следующим параметрам:

* Степень разрушаемости окружения.
* Способы взаимодействия игрока с окружением.
* Положение игровой камеры.
* Геймплей игрока.
* Особенности игрового ландшафта.

Существует сравнительно небольшое количество игр с DE, а тем более в жанре шутер []. Для анализа были выбраны 3 игры на основе их популярности и влияния на жанр в целом.

### 1.1.1. Worms 3D (2003)

Первые игры серии Worms уже поддерживали TD, и, бесспорно, дали толчок к развитию жанра шутер и механики TD в целом. В 2003 году выходит первая трёхмерная игра серии, которая не только сохранила предыдущие особенности, но и внесла новые. Переход в трёхмерное пространство дал игроку больше свободы и возможностей.

Игрок должен создать свою команду персонажей – червей, и, используя различное оружие, уничтожить команду противника. Персонажи ходят по очереди, сначала один червь из одной команды, затем следующий из другой, и так далее. Таким образом, за ход передвигается и стреляет только один персонаж, что позволит не нагружать систему большим количеством событий. Выбрав позицию, игрок выбирает оружие, которым хочет сделать выстрел, прицеливается, и выпускает снаряд. Каждый снаряд имеет свои особенности баллистики и поведения, что задаёт тактический аспект игры, а также свой наносимый урон. Цель игры – уничтожить всю команду соперника.

При движении игрока по карте, игровая камера находится в режиме «от третьего лица» для того, чтобы полностью было видно местность вокруг. При прицеливании камера переходит в режим «от первого лица» для удобства задания траектории полёта снаряда. Некоторые снаряды имеют свою камеру, например вид на карту сверху (см. рис 1.1).



Рисунок . – Два режима камеры в игре Worms 3D. Камера от третьего лица при передвижении слева и камера от первого лица при стрельбе справа

Игровой ландшафт разрушается от взрывов, вызванных выпущенными снарядами. Игроку доступен любой уровень разрушения, от рытья таким образом туннелей и пещер, до полного разрушения отдельных участков местности. Чтобы игрок не уничтожил всю карту, в игру добавлена вода, находящаяся в самом низу игровой карты и поднимающаяся с каждым ходом. При попадании в воду персонаж погибает. Игровая карта может иметь любую форму, имеются уровни как в виде обычных островов, так и средневековых замков. Такая мобильность создана при помощи использования вокселей. Воксели составляют игровой ландшафт как кирпичики, и при взрыве их часть удаляется, тем самым деформируя местность.

### 1.1.2. Scorched 3D (2001)

Другой «классикой» является игра Scorched 3D, которая является пошаговой артиллерийской стратегией. Игрок управляет танком, который размещается на игровой карте случайным образом. За один ход игрок может либо сделать выстрел, либо изменить позицию. Цель – уничтожить танки противников. При выстреле игрок выбирает угол вращения и силу выстрела, выбирает боеприпасы и инвентарь.

Игра имеет некоторые отличия от Worms 3D. Во-первых, игрок может видеть сразу обе камеры. Главная демонстрирует вид на карту сверху, её можно перемещать и рассматривать каждый кусок местности. Камера сбоку находится в режиме «от первого лица», и показывает, куда смотрит дуло танка (см. рис. 1.2).



Рисунок . – Игровой интерфейс игры Scorched 3D

Вторым отличием является модель разрушения ландшафта. В игре реализован динамический ландшафт через полигоны []. Это значит, что от взрывов ландшафт как бы прогибается, но при этом невозможно создание различных подповерхностных структур, таких как туннели и пещеры. Однако, это не исключает возможность полного уничтожения игровой карты.

### 1.1.3. Battlefield 3 (2011)

В отличие от предыдущих игр, Battlefield является более динамичной игрой, так как действие происходит в реальном времени. Игра является шутером от первого лица, и в неё можно играть как одному, проходя сюжет, так и в многопользовательском режиме. Проходя сюжет, требуется выполнить определённое задание, и уничтожение всех противников не является главной целью. В многопользовательском режиме игроки также действуют одновременно, играя за разные классы персонажей.

Battlefield 3 предоставляет большой уровень разрушаемости зданий, природных объектов и тому подобного. Использовался специальный физический движок Havok Physics, а создатели утверждали, что с его помощью возможно моделирование разрушение модели небоскрёба Бурдж-Халифа []. Однако уровень деформации ландшафта довольно поверхностный – в игре возможно создание кратеров от взрывов, но не более (см. рис. 1.3). Метод реализован с помощью генерации полигональной карты высот []. Однако, такой подход задал игре высокий уровень графики и детализации.



Рисунок . – Демонстрация разрушения ландшафта в игре Battlefield 3 [4]

### 1.1.4. Сравнительная характеристика игр с механикой деформации ландшафта

По итогам анализа вышеперечисленных игр составлена следующая таблица (табл. 1.1):

**Таблица 1.1** – **Сравнительная таблица существующих игр с механикой TD**

| **Критерий** | **Worms 3D** | **Scorched 3D** | **Battlefield 3** |
| --- | --- | --- | --- |
| Игровой темп | Пошаговая стратегия | Пошаговая стратегия | Шутер в реальном времени |
| Камера игрока | Переключение между камерами от первого и третьего лица | Камера от третьего лица с ограниченной камерой от первого | Камера от первого лица |
| Масштабы разрушаемости | Полное свобода разрушения | Практически полная разрушаемость с ограничениями | Практически полная разрушаемость зданий, поверхностная разрушаемость ландшафта |
| Мультиплеерный режим | Присутствует, но игроки ходят по очереди | Присутствует, но игроки ходят по очереди | Присутствует, игроки действуют в реальном времени |
| Способ реализации TD | Воксели | Полигональная сетка | Полигональная сетка |

Как мы видим, существующие игры используют TD по-разному. Однако не были найдены решения, которые бы совмещали многогранность, тактический аспект этой механики и командную игру в реальном времени. Worms использует TD в геймплейный целях, позволяя разрушать ландшафт под игроками, чтобы уничтожить их, или рыть туннели, но является пошаговой игрой. В Battlefield же играет роль разрушение окружения в виде построек, а не конкретно ландшафта.

## 1.2. Анализ существующих методов генерации деформируемого ландшафта

Как мы можем видеть, игры, в которых игрок может так или иначе деформировать ландшафт, используют для этого в основном два подхода: изменение полигональной сетки ландшафта и генерация ландшафта с помощью вокселей. Будем сравнивать эти способы, основываясь на следующих критериях:

1. Как работают эти подходы, и как это может повлиять на геймплей?
2. По какому алгоритму эти подходы деформируют ландшафт и какие проблемы этот алгоритм может создать?
3. Каким образом эти методы могут быть оптимизированы и может ли эта оптимизация быть реализована в нашей работе?

### 1.2.1. Генерация ландшафта с помощью полигональной сетки

Чаще всего в игре нам не важна внутренность игрового объекта, игрок взаимодействует только с поверхностями. В таком случае используется тесселяция поверхности. Тесселяция – это процесс разбиения поверхности на набор многоугольников - полигонов []. Традиционно разработчики делят поверхности на треугольники, что называется триангуляцией, по следующим причинам:

* треугольник – простейший тип многоугольника;
* треугольник всегда плоский, так как его вершины эту плоскость и определяют;
* в результате практически любых преобразований треугольник останется треугольником;
* большинство современного оборудования создано на основе треугольной растеризации [].

Таким образом треугольники становятся кусочно-линейной аппроксимацией поверхности (рис 1.4).

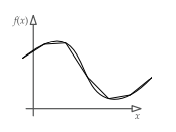


Рисунок . – Демонстрация триангуляции как приближения к поверхности []

Так как полигональная сетка является только «обволакивающей» поверхностью пустого трёхмерного пространства, это накладывает некоторые ограничения. Например, сильно затруднится генерация таких вещей как пещеры или мосты. Другой проблемой будет моделирование динамических объектов. Если нам нужно показать, например, волны на поверхности моря, то использование полигонов в этой ситуации потребует создание дополнительных объектов (брызги воды), изменение старых (деформированная волна), и так далее.

С тесселяцией неразрывно связано такое понятие как уровень детализации (level of detail, далее LOD). Фиксированная тесселяция может привести к тому, что на поверхности фигуры будут видны блоки, как на рисунке выше (рис. 1.4). Это решается увеличением количества полигонов, но, если каждый объект в игровом мире будет постоянно отрисовываться подробно, это сильно ударит по скорости компьютера, так как фигур на экране слишком много. Для решения этой проблемы подготавливают несколько уровней тесселяции поверхности объекта. Если камера близко к объекту – детализируем его максимально, если далеко – количество полигонов можно в разы уменьшить. На рисунке (см. рис. 1.5) левая поверхность разбита на 5000 треугольников, в то время как правая всего на 200 []. Таким образом, мы можем получить детализированное изображение, при этом сохранив большое количество памяти GPU.

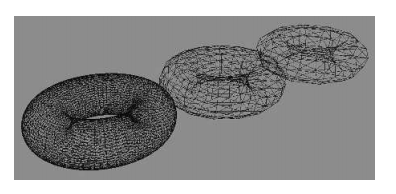


Рисунок . – Три уровня детализации объекта []

Ландшафт в современных играх генерируется уже с расчётом LOD’ов, которые меняются по мере передвижения игровой камеры. К примеру, одно из решений создания полигональной сетки для ландшафта имеет следующий алгоритм []:

1. Определяется позиция и направление камеры игрока. Во входные данные поступают все видимые регионы ландшафта.
2. Запускается цикл, одна итерация соответствует одному региону. Если регион не попадает в поле зрение камеры, то он пропускается.
3. Затем проходит проверка, нужно ли детализировать регион. Чем ближе регион к игроку, тем больше он должен быть детализирован. Детализация проводится разбиением региона на 4 части. Затем для каждой части рекурсивно проходит такая же проверка. Выходными данными является детализированный по разной степени LOD’ов регион (см. рис. 1.6).

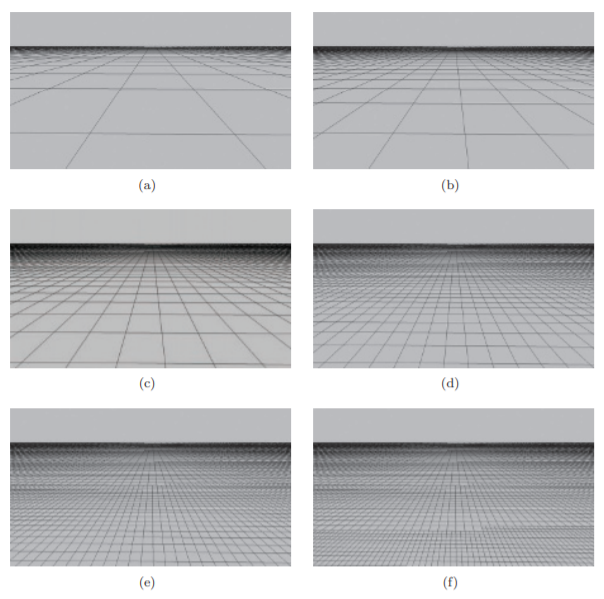


Рисунок . – Вид полигональной сетки ландшафта на итерациях алгоритма N, N+1, N+2, N+3, N+4 и N+5 соответственно []

Полигональный ландшафт, деформированный каким-либо воздействием, изменяется подобным образом. В месте деформации происходит детализация сетки, и, в зависимости от силы деформирующего воздействия, разные LOD’ы меняют свои координаты, и полигоны в данном месте перерисовываются. На рисунке (см. рис 1.7) ниже видно, как в местах, деформированных грузовым транспортом, LOD’ы намного больше, чем на ровной поверхности []. Однако такой метод требует много расчётов, и может замедлить процесс рендеринга изображения.

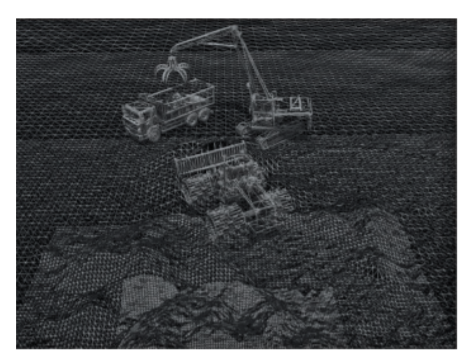


Рисунок . – Уровень детализации ландшафта в местах деформации []

После проведения тесселяции, координаты вершин треугольников объекта записываются в массив. Так как одна вершина может принадлежать сразу нескольким треугольникам, обычно разделяют массивы вершин и массивы индексов, комбинации которых уже и составляют треугольник. Делается это чтобы не отрисовывать и не освещать одну и ту же вершину несколько раз. Есть и другие структуры данных, направленные на экономию памяти компьютера, такие как «полоса» (Strips) и «веер» (Fans), использующие свои закономерности порядка записи индексов. Такой порядок также способствует удобному доступу к памяти для GPU []. Например, в структуре данных Strips, две вершины одного треугольника являются и двумя вершинами соседнего треугольника, записанного в массиве. Такой подход позволяет сильно сократить размерность массива.

Таким образом, полигональная сетка позволит генерировать хорошо детализированный ландшафт, а учитывая, что полигоны хранятся на GPU, процесс генерации будет происходить быстро. Однако, при деформации ландшафта, полигоны требуют слишком большого количества расчётов. Если в игре будет происходить слишком много событий, например, сразу несколько игроков будут стрелять по ландшафту, это может вызвать лаги и задержки.

### 1.2.2. Генерация ландшафта с помощью вокселей

Если пиксель является «элементом изображения» (picture element), то воксел является «элементом объёма» (volume element). Воксели можно сравнить с атомами, из которых состоит трёхмерное пространство, и их совокупность представляется в виде трёхмерной сетки. Каждый воксел является отдельным объектом и поэтому может хранить в себе необходимую для задачи информацию, такую как цвет, прозрачность (для симуляции таких вещей как дым, вода), плотность и тому подобное. Информация о местоположении вокселя не требуется, так как это определяется его индексом в сгенерированной сетке [].

Основное отличие между полигонами и вокселями в том, что в то время, как полигоны покрывают пустое трёхмерное пространство, создавая полую фигуру, вокселями составляют объём этой фигуры. Другими словами, объём фигуры равен сумме вокселей внутри неё. Благодаря этому, решение проблемы деформации какого-либо объекта, состоящего из вокселей, сводится к задаче прибавления и вычитания этих вокселей в нужных местах []. Такая гибкость вокселей даёт им большую вариативность в использовании, чем у полигонов. Для нашей игры это значит, что мы сможем создавать деформацию любых масштабов, от глобального изменения формы объекта до полного его уничтожения.

Проблемой вокселей является больше потребление памяти, имеющую величину . То есть локация, размерностью в 1000 вокселей по каждой оси уже имеет размер в 1 000 000 000 блоков, что создаёт огромную нагрузку на память процессора. Однако, соседние воксели часто имеют схожие значения, а некоторые части сетки вообще могут быть пустыми. Поэтому одним из основных методов оптимизации воксельной сетки и построения воксельного мира является построение разряжённого воксельного октодерева (Sparse voxel octree, далее SVO). Такой подход использовали авторы игры Teardown [], и такой подход использует физический движок PhysX, в данный момент принадлежащий компании NVIDIA [].

Суть SVO состоит в следующем. За условный объект возьмём дерево, состоящее из ствола, какого-то количества веток и листьев. Окружим наш объект идентичной по размеру областью в форме куба, и начнём делить эти область на части. Каждая часть может быть рекурсивно детализирована в дальнейшем. Проводим эту детализацию до тех пор, пока каждой области не соответствует часть нашего объекта, имеющая один набор свойств, одинаковый для каждой точки в этой области. То есть в нашем примере одной области может соответствовать либо весь ствол, либо одна из веток, либо лист. Таким образом, получится граф или дерево областей, которые содержат в себе части нашего объекта. Эти области и являются наборами вокселей, характеризующими конкретную часть объекта. На рисунке ниже (рис. 1.8) показано, как мы проделываем процедуру до тех пор, пока конечная область не будет содержать полностью один элемент или будет пустой []. Первоначальный куб является корнем дерева, конечные области-воксели являются листьями. Если же мы как-либо изменим часть объекта, то соответствующая деформируемой части область детализируется далее. То есть пока объект не тронут, он представляет собой один большой «куб», который меньше нагружает систему. Такой подход позволит сильно сэкономить память процесса, превращая миллиард блоков в несколько сотен или тысяч (в зависимости от объекта).

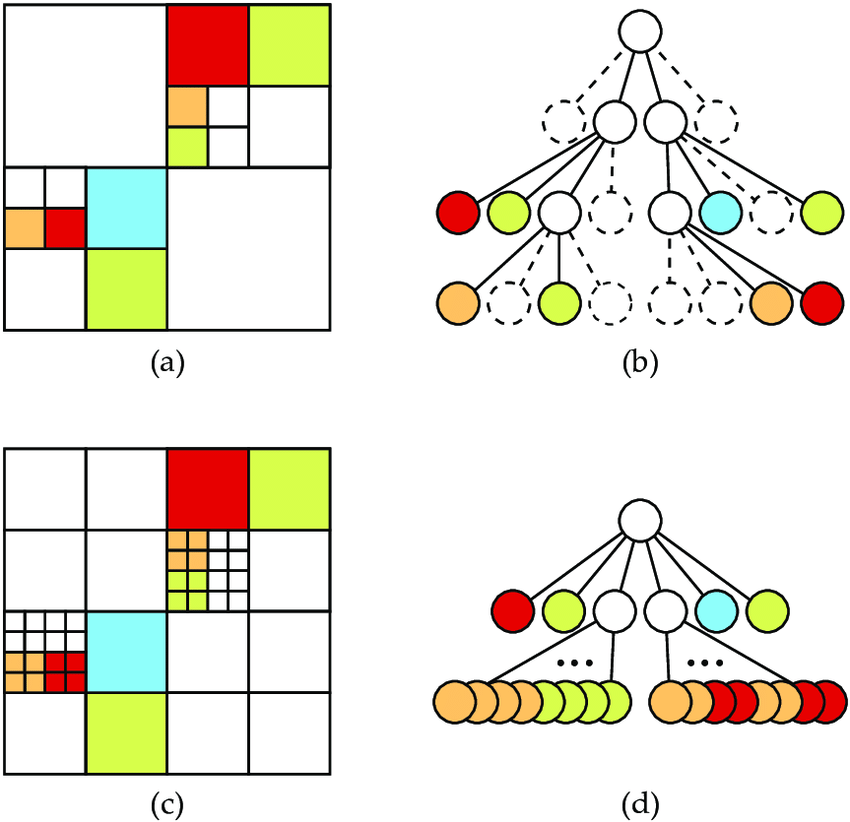


Рисунок . – Примеры построения октодерева []

Для создания воксельного объекта используются вокселизация -преобразование объекта, состоящего из полигонов, в набор вокселей []. При создании воксельной сетки для каждого вокселя происходит проверка: лежит он внутри объекта или снаружи, затем эта информация записывается в воксель. Современные методы вокселизации позволяют генерировать воксельные объекты уже с построенным SVO [].

Таким образом, процедура создания и деформирования воксельного объекта следующая:

1. Создание объекта в игровом пространстве, определение его мировых координат.
2. Вокселизация объекта. Воксели, находящиеся внутри объекта (формирующие его объём), содержат соответствующую информацию.
3. При событии, как-либо деформирующем объект, проводим следующий алгоритм:
   1. Определяется позиция и масштаб деформации.
   2. Пока соответствующий лист SVO больше масштаба деформации – детализируем данную ветвь SVO.
   3. Когда лист, соответствующий масштабу деформации получен, проводим с этим набором вокселей соответствующую операцию: 1) если причиной деформации было разрушение объекта, то принадлежность к объекту у соответствующего набора удаляется; 2) если причиной деформации было увеличение объекта, то соответствующему набору вокселей присваивается принадлежность к объекту.

Как мы можем видеть, воксельный подход обладает большими затратами памяти, а также углы кубов, из которых состоят ландшафт и объекты, скорее всего будут видны. Однако, алгоритм построения SVO сильно сэкономит память компьютера, а специфичное изображение может свести на игровой стиль. Самым существенным плюсом воксельного подхода становится простота работы с деформацией ландшафта, являющаяся лишь вопросом удаления или добавления вокселей в нужном месте. Такая эффективность позволит быстро обрабатывать все события деформации и позволит сразу нескольким игрокам принимать участие в игре.

Составим таблицу сравнения для обоих подходов (табл. 1.2):

Таблица . – Сравнительная таблица методов деформации ландшафта

| **Критерий** | **Полигональная сетка** | **Воксели** |
| --- | --- | --- |
| Влияние на геймплей | Внутренность объекта не важна | Составляют полный объём объекта |
| Аппаратная часть | GPU | CPU |
| Как деформирует ландшафт? | Сетка детализируется в месте деформации, для каждого треугольника рассчитывается новая позиция | Добавление и удаление вокселей в месте деформации |
| Основная проблема | Большое количество расчетов при деформации | Нагрузка на память при большом разрешении игрового мира |
| Насколько эффективны методы оптимизации | Количество полигонов со структурами данных = , вместо (где – количество всех вершин треугольников) [6] | Количество вокселей после наложения SVO уменьшается в 3.5–5.5 раз (зависит от вокселизируемого объекта) [12] |

Для данной работу выбран метод вокселей с применением SVO. Решающими критериями являются простота процесса деформации объектов и более эффективный метод оптимизации. Такие плюсы позволят иметь в игре высокую производительность и позволит нескольким игрокам одновременно изменять игровой ландшафт.

## 1.3. Выявление требований к разрабатываемой игре

На основе рассмотренных ранее игровых аналогов можно выделить следующие общие паттерны жанра шутер:

1. Игрок использует оружие дальнего боя, чтобы наносить урон противникам и деформировать ландшафт.
2. Игрок может использовать разное оружие дальнего боя, может менять его во время игры.
3. Используется камера от первого лица.

В нашей версии будут добавлены следующие механики:

1. Игра происходит в реальном времени.
2. В игре может участвовать несколько игроков одновременно.
3. Все игроки могут деформировать ландшафт.
4. Разные оружия наносят разный урон противникам и по-разному деформируют ландшафт (изменяется вид кратера от взрыва, его размеры и тому подобное).
5. Игроки могут деформировать любое игровое окружение, включая какие-либо статические объекты на карте.
6. Использование воксельного метода генерации ландшафта позволит для каждого вокселя задавать дополнительную информацию. На основе этого, возможна генерация объектов и локаций из различного материала, и изменение уровня их деформируемости.

### 1.3.1. Описание функциональных и нефункциональных требований к игре

**Назначение приложения**: решение проблемы оптимизации отрисовки изображения при деформации ландшафта в игре жанра шутер.

На основе проанализированных существующих игровых решений и методов генерации деформируемых ландшафтов, выявим следующие требования.

**Функциональные требования**:

1. Функциональные требования к игре:
   1. Возможность изменить настройки игры.
      1. Возможность изменить уровень звука в игре.
      2. Возможность изменить уровень разрешения экрана в игре.
      3. Возможность изменить качество текстур в игровой миссии.
      4. Возможность изменить в настройках управление персонажем.
2. Функциональные требования к игровому процессу:
   1. Персонаж игрока может перемещаться по игровой карте.
   2. Персонаж игрока может производить стрельбу из своего оружия.
   3. При получении определенного количества урона персонаж игрока погибает и теряет способность к передвижению и ведению стрельбы.
   4. Игрок может производить перезарядку своего оружия.
   5. Игрок может менять оружие, из которого производится стрельба

**Нефункциональные требования**:

1. Игровая камера находится в режиме от первого лица.
2. Минимальное количество кадров в секунду – 30.

**Требования к пользовательскому интерфейсу**:

1. Пользовательский интерфейс представлен на английском языке.

# Проектирование игровых компонентов и механик

Результатами главы являются:

* модель поведения приложения, описанная с использованием диаграмм прецедентов и последовательностей;
* описание работы подсистемы жизненного цикла TD, алгоритм генерации разрушаемого ландшафта и обработка событий его деформации;
* описание работы подсистемы жизненного цикла игрового персонажа;
* проектирование пользовательского интерфейса игры.

## 2.1. Конструирование архитектуры разрабатываемой игры

Выделив основные паттерны и основные игровые механики жанра «шутер», приведём их декомпозицию с целью выявления связей между компонентами и схемы взаимодействия подсистем. Для этого составим диаграммы прецедентов и опишем сценарии взаимодействия компонентов (см. рис. 2.1).

1. Название: Изменение пользователем игровых настроек (табл. 2.1).

Акторы: игрок.

Краткое описание: игрок хочет изменить игровые настройки, чтобы игровая производительность соответствовала возможностям компьютера.

Триггер: запуск приложения.

Таблица 2.1 – Описание прецедента "Изменение пользователем игровых настроек"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок запускает систему. | 2) Система запускается и открывает главное меню. |
| 3) Игрок выбирает пункт меню «Настройки». | 4) Система открывает окно «Настройки» |
| 5) Игрок меняет игровые настройки и нажимает кнопку «Принять». | 5) Система обрабатывает введённые данные, изменяя настройки в соответствии с введёнными (Е1). |

Альтернативные потоки:

E1: в случае если настройки не были изменены, их перенастройка не производится.

2. Название: Перемещение персонажа по игровой карте (см. табл. 2.2).

Акторы: игрок.

Краткое описание: игрок хочет изменить своё местоположение на игровой карте.

Триггер: запуск игровой миссии.

Таблица 2.2 – Описание прецедента "Перемещение персонажа по игровой карте"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок запускает игровую миссию. | 2) Система запускает игровую миссию, генерирует ландшафт и создаёт на ландшафте персонажей. |
| 3) Игрок нажимает клавиши, отвечающий за передвижение персонажа. | 4) Система расчитывает координаты, в которые должен переместиться персонаж, и персонаж начинает перемещаться туда по проложенной траектории (Е1). |

Альтернативные потоки:

E1: в случае если рассчитанная координата недосягаема для персонажа, персонаж проходит по траектории до конечной досягаемой координаты и останавливается.

3. Название: Смена оружия (табл. 2.3).

Акторы: игрок.

Краткое описание: игрок может сменить оружие, чтобы изменить тип снарядов.

Триггер: запуск игровой миссии.

Таблица 2.3 – Описание прецедента "Смена оружия"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок нажимает клавишу для смены оружия. | 2) Система меняет оружие, принадлежащее персонажу в данный момент. |

4. Название: Ведение стрельбы персонажем (табл. 2.4).

Акторы: игрок.

Краткое описание: игрок хочет начать вести стрельбу в игровой миссии, по ландшафту или по противнику.

Триггер: прецедент «Смена оружия».

Таблица 2.4 – Описание прецедента "Ведение стрельбы персонажем"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок запускает игровую миссию. | 2) Система запускает игровую миссию, генерирует ландшафт и создаёт на ландшафте персонажей. |
| 3) Игрок наводит камеру в нужное ему место и нажимает клавишу, отвечающую за стрельбу. | 4) Система расчитывает координаты, куда должен полететь снаряд и запускает его по траектории. Если при достижении одной из точек траектории там находился персонаж противника, ему наносится урон в соответствии с типом снаряда (Е1). |

Альтернативные потоки:

E1: в случае если при достижении одной из точек траектории там находился ландшафт, он деформируется в соответствии с типом снаряда.

5. Название: Смерть игрового персонажа (табл. 2.5).

Акторы: игрок.

Краткое описание: когда персонаж получает урона больше, чем он имеет здоровья, он погибает и теряет способность к ведению любой деятельности.

Триггер: прецедент «Ведение стрельбы персонажем».

Таблица 2.5 – Описание прецедента "Смерть игрового персонажа"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок наводит камеру под таким углом, чтобы попасть в противника и нажимает клавишу стрельбы. | 2) Система расчитывает координаты, куда должен полететь снаряд и запускает его по траектории. Снаряд попадает в персонажа противника и наносит ему урон. Если у противника заканчиваются очки здоровья, система перестаёт реагировать на его действия и считает его «мёртвым». |

6. Название: Перезарядка оружия (табл. 2.6).

Акторы: игрок.

Краткое описание: когда у игрока кончаются снаряды, доступные для стрельбы, он хочет провести перезарядку для увеличения количества снарядов для использования.

Триггер: прецедент «Ведение стрельбы персонажем».

Таблица 2.6 – Описание прецедента "Перезарядка оружия"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок нажимает клавишу для перезарядки. | 2) Система обновляет количество снарядов, доступных для стрельбы. |



Рисунок . – Диаграмма прецедентов

На этой основе составим диаграммы последовательностей для более детального понимания взаимодействия между подсистемами при действиях пользователя.

1) Имя: «Изменить игровые настройки» (рис. 2.2)

Обязанности: нет.

Ссылки: прецедент «Изменение пользователем игровых настроек».

Исключения: если пользователь несколько раз поменял значения, и в итоге они остались старыми, перенастройка не проводится.

Предусловия: включение приложения, переход в раздел главного меню «Настройки».

Постусловия: изменение значений настроек. Игра будет использовать новые настройки.



Рисунок . – Диаграмма последовательностей "Изменить игровые настройки"

2) Имя: «Переместить персонажа на игровой карте» (рис. 2.3)

Обязанности: персонаж должен быть «живым», то есть уровень здоровья персонажа выше нуля.

Ссылки: прецедент «Перемещение персонажа по игровой карте».

Исключения: если пользователь старается переместиться в недоступную для него координату, то он останавливается в ближайшей доступной точки на проложенной траектории.

Предусловия: включение приложения, включение игры.

Постусловия: изменённое положение персонажа на игровой карте.

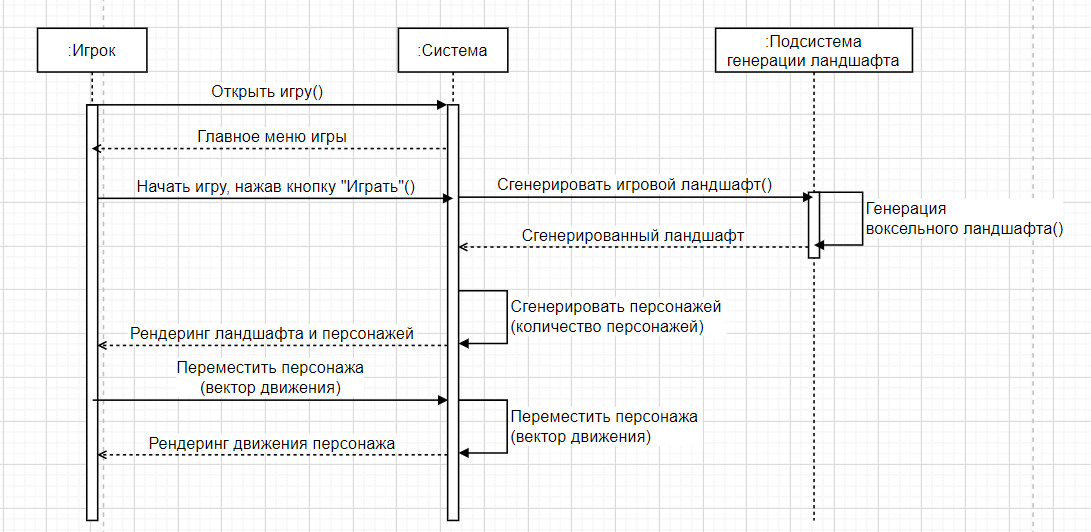


Рисунок . – Описание последовательностей "Переместить персонажа на игровой карте"

3) Имя: «Сменить оружие» (см. рис. 2.4)

Обязанности: персонаж должен быть «живым», то есть уровень здоровья персонажа выше нуля.

Ссылки: прецедент «Смена оружия».

Исключения: нет.

Предусловия: включение приложения, включение игры.

Постусловия: изменённый тип оружия у игрока.

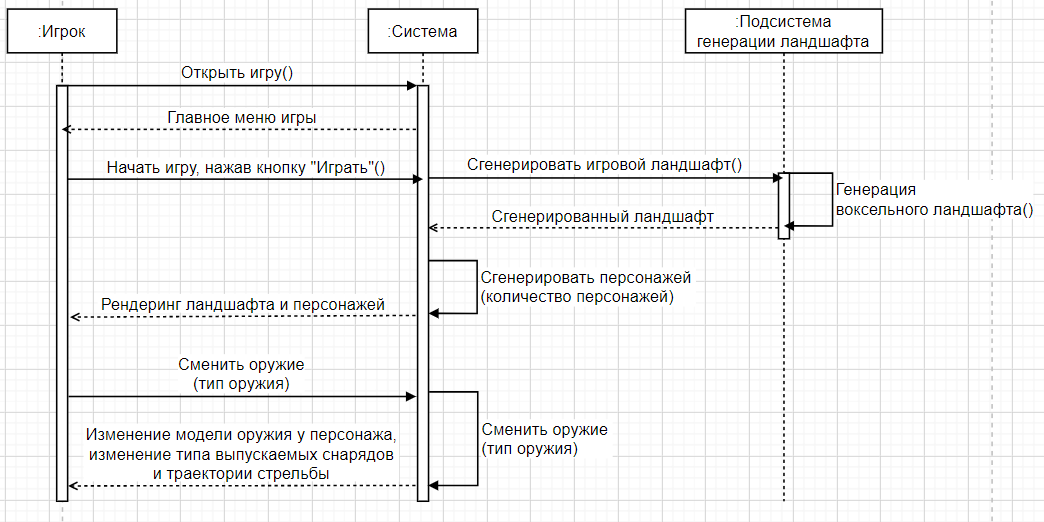


Рисунок . – Описание последовательностей "Сменить оружие"

4) Имя: «Провести стрельбу» (рис. 2.5)

Обязанности: персонаж должен быть «живым», то есть уровень здоровья персонажа выше нуля.

Ссылки: прецедент «Ведение стрельбы персонажем».

Исключения: если у игрока нет снарядов, доступных для стрельбы, стрельба не ведётся.

Предусловия: включение приложения, включение игры.

Постусловия: изменённое количество снарядов у игрока, изменённое игровое состояние в зависимости от попадания снаряда.

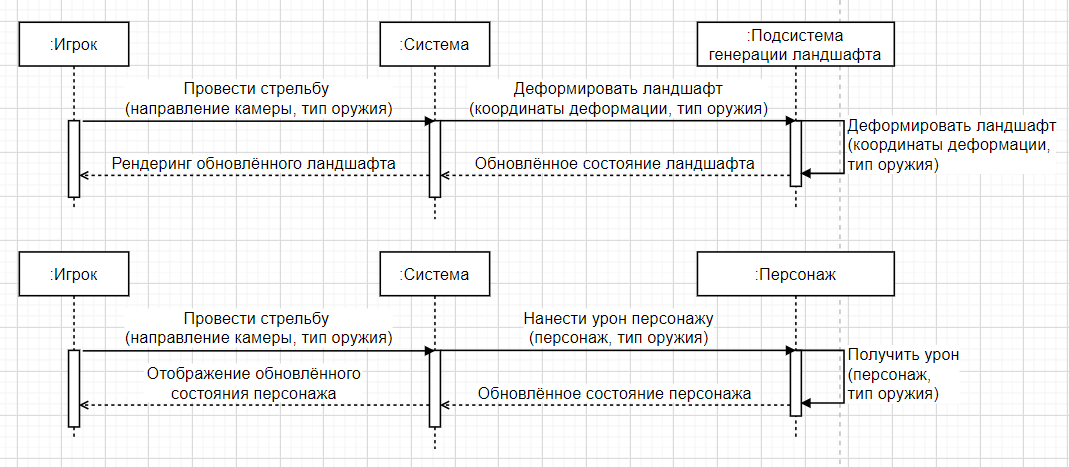


Рисунок . – Описание последовательностей "Провести стрельбу"

5) Имя: «Отключить персонажа по причине смерти» (рис. 2.6)

Обязанности: уровень здоровья персонажа должен опуститься ниже нуля.

Ссылки: прецедент «Смерть персонажа».

Исключения: нет.

Предусловия: попадание одного или более снарядов противников в персонажа.

Постусловия: изменённое состояние игры, изменённое состояние персонажа.

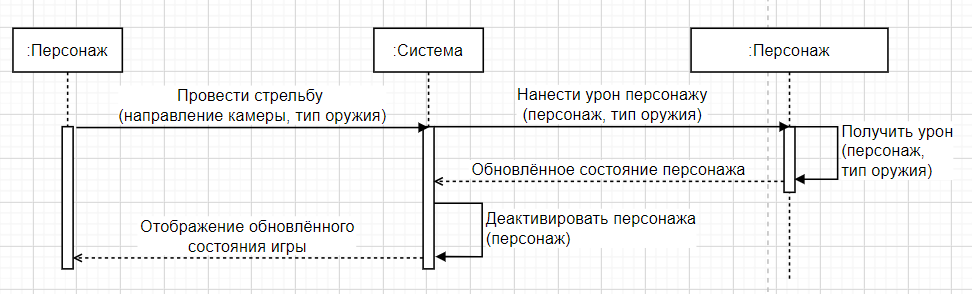


Рисунок . – Описание последовательностей "Отключить персонажа по причине смерти"

6) Имя: «Перезарядить оружие» (рис. 2.7)

Обязанности: количество доступных для стрельбы снарядов равно нулю.

Ссылки: прецедент «Перезарядка оружия».

Исключения: нет.

Предусловия: проведение серии выстрелов персонажем.

Постусловия: изменённое количество снарядов, доступных для стрельбы.

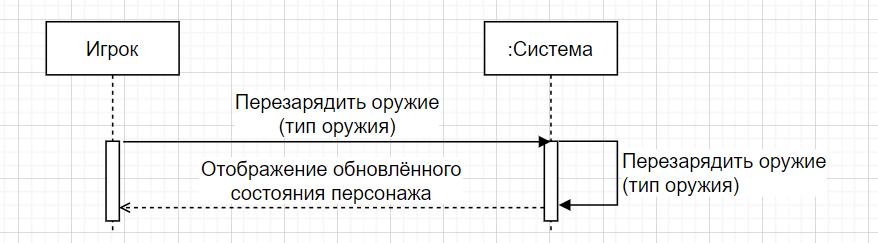


Рисунок . – Описание последовательностей "Перезарядить оружие"

Как мы видим, дополнительно требуется описать некоторые подсистемы. Требуется проектирование алгоритма работы подсистемы генерации деформируемого ландшафта и проектирование жизненного цикла персонажа. Затем, требуется описать взаимодействие этих двух подсистем через взрывы снарядов.

## 2.2. Проектирование механики генерации TD

Исходя из информации, полученной при разборе воксельного подхода к созданию объектов, алгоритм генерации деформируемого ландшафта, следующий:

1. Создание игровой карты полигональным методом. Случайная генерация ландшафта, неровностей поверхности.
2. Вокселизация созданного ландшафта.
3. Наложение структуры SVO на воксельную сетку карты.

Наличие на ландшафте впадин и холмов поспособствует тактической составляющей игры в качестве укрытий и тому подобного. Неровности поверхности можно сгенерировать путём наложения карты высот – двухмерной текстуры, содержащей информацию о внешнем виде ландшафта [16]. Мы хотим, чтобы такая структура задавала постоянно случайные значения для разнообразия созданных карт, и при этом соседние значения не различались слишком резко во избежание аномалий – не существует горы высотой несколько километров и шириной пару метров. Поэтому, для задания карт высот используют алгоритмы «шумов» - математические алгоритмы для генерации текстур псевдослучайным образом [16].

Самыми известными и проверенными на практике являются два подхода: Шум Перлина и Шум Симплекс. Оба алгоритма разработаны Кеном Перлином [17, 18], и Шум Симплекс был придуман позже для устранения ограничений и уменьшения вычислительной сложности. Однако, использование реализаций со вторым алгоритмом покрывается патентом [19], в то время как классический Шум Перлина доступен для использования в игровых движках, наподобие Unity. Чёткий выбор между подходами будет сделан во время разработки. Результаты алгоритмов практические не отличаются, и единственное осложнение, которое может быть получено – увеличенное время загрузки игрового мира перед стартом игры.

Следующий шаг – вокселизация созданного ландшафта. В работе будет использоваться алгоритм, предложенный в статье «Fast parallel surface and solid voxelizations on GPUs». Основная идея заключается в наложении воксельной сетки на объект и определении, лежит ли полигон внутри конкретного вокселя следующим образом:

* Полигон с вершинами , должен перекрывать воксель.
* Каждая из вершин , должна лежать внутри вокселя.

Наконец, используем структуру SVO для хранения вокселизированной карты. Принцип работы алгоритма был описан выше – мы делим области до определённого момента, таким образом группы вокселей с одинаковыми параметрами объединяются в один «лист» дерева и хранятся в памяти как один объект. Следует поставить несколько ограничений:

* Если мы получили «лист», в котором только один объект (в нашем случае участок ландшафта, обладающий свойствами отличными от соседних) – деление останавливается.
* Если «лист» не содержит объектов – в структуру он не добавляется.
* Необходимо задание минимального размера «листа», чтобы разделение не продолжалось до переполнения стэка. При разработке оптимальные размеры карты будут подбираться опытным путём, но за исходные величины возьмём максимальный размер карты равный 200 единиц, и минимальный размер вокселя равный 0.1 единице.

Для каждого «листа» в дереве действуют следующие правила:

* Каждый «лист» определяет включающую его область.
* Каждый «лист» имеет ссылку на родительскую «ветвь», а каждая «ветвь» на своего родителя и вплоть до «корня».
* «Ветвь» имеет восемь потомков, которые могут быть как «ветвями», так и «листьями».
* Каждый узел описывает входящие в него объекты (ландшафтные воксели).

Ландшафт в игре деформируется от взрыва снарядов. Каждый снаряд, выпущенный игроками, будет иметь параметр разрушения, показывающий, сколько минимальных вокселей уничтожает взрыв при контакте с ландшафтом. Алгоритм деформации структуры выглядит следующим образом:

1. Определяем «листья», в которых лежат уничтоженные взрывом воксели. С этим не возникнет проблем, так как местонахождение вокселя определяется его индексом в сетке.
2. Если взрыв произошёл только в одном «листе», то достаточно удалить из структуры только его.
3. Если взрыв был масштабным и задел несколько «листьев», перемещаемся по родительским узлам вверх, до тех пор, пока не достигнем «ветви», в которой лежат все выделенные «листья», и удаляем всю «ветвь».

Возможно, в игре будут оружия, способные не только разрушать ландшафт, но и деформировать его форму без изменения объёма. Это более объёмная в плане вычислений задача, так как помимо предыдущих действий, требуется:

1. установить, где именно добавились переместившиеся воксели;
2. дойти вверх по родителям до «ветви», покрывающей всё пространство с добавленными вокселями;
3. перестроить SVO для этой «ветви».

## 2.3. Проектирование геймплея игрока

Так как игра находится в трёхмерном пространстве, персонаж игрока должен уметь перемещаться по всем трём осям. Помимо стандартного передвижения вперёд, влево, вправо, назад, нужно добавить игроку возможность перепрыгнуть препятствие. Персонаж будет двигаться с постоянной равномерной скоростью, равной 2 единицы расстояния в секунду, таким образом, ему понадобится минимум 100 секунд, чтобы добежать с одного края карты до противоположного.

Основной способ взаимодействия персонажа с другими персонажами и окружающей средой заключается в проведении стрельбы. В играх существует два подхода реализации механики стрельбы: Hitscan и Баллистика летящего предмета [20]. Алгоритм работы hitscan’а можно описать следующим образом:

1. При возникновении события «стрельба», в направлении оружия выпускается луч на заданное расстояние.
2. Определяется первый объект, в который попал этот луч. Таким образом мы считаем, что этот объект находился на линии огня и в него попал выпущенный снаряд.

Основным преимуществом подхода является очень быстрая скорость вычисления коллизий, не требующая дополнительных затрат памяти. Всё, что нужно отслеживать – направление луча. С другой стороны, такой подход означает, что если луч попал в объект, то от снаряда невозможно увернуться, даже если стрелявший находится на большом расстоянии от объекта. Также, так как не тратится память на создание нового объекта – снаряда, то невозможно для него учесть влияние дополнительных факторов, таких как направление ветра. Поэтому, для задания большего реализма в игре часто используется второй подход – Баллистика летящего предмета [20].

При стрельбе снаряд создаётся как отдельный объект, обладающий характеристиками, такими как масса, скорость и так далее. Таким образом мы можем регулировать его траекторию и скорость полёта. Задавать траекторию снаряда будем по классической формуле движения снаряда под углом к горизонту, где – угол между поверхностью и горизонтом, – первоначальная скорость полёта снаряда, g – ускорение свободного падения [20]:

|  |  |
| --- | --- |
|  |  |
|  |

Создание дополнительных объектов создаёт нагрузку на систему, однако имеются способы для оптимизации. Во-первых, можно элементарно ограничить количество снарядов, которое игрок может выпустить за определённый период времени. Тогда не будет возникать непредвиденных ситуации с потерей количества кадров в секунду из-за большого количества событий. Во-вторых, все снаряды можно создать заранее и помещать их на карту по мере выстрела. А после попадания вернуть снаряд обратно в хранилище, и затем использовать повторно.

В игре будем использовать оба подхода. Некоторые оружия будут использовать hitscan, некоторые баллистику. Это создаст дополнительные тактические особенности и варианты принятия решений для игроков.

Персонаж в состоянии «живой» может передвигаться по карте, менять оружие, вести стрельбу, перезаряжаться для продолжения стрельбы. У каждого персонажа определённый уровень здоровья. Когда персонаж получает столько урона от взрывов и попаданий снарядов, что его уровень здоровья опускается до нуля – он переходит в состояние «мёртвый», в котором не может исполнить ни одного из вышеперечисленных действий.

# Реализация игровых компонентов и механик

Результатами главы являются:

* выбор игрового движка для разработки игры;
* описание процесса разработки игры, приведение программного кода и скриншотов.

## 3.1. Выбор инструмента для разработки приложения

Для разработки демоверсии игры требуется игровой движок, который обладал бы следующими характеристиками:

* низкий порог вхождения: изучение тонкостей использования движка помимо разработки системы займёт много дополнительного времени;
* наличие библиотек для поддержки физики твёрдых тел, определения коллизий между ними, а также наличие графических инструментов (создание источников света, обработка теней, поддержка LOD’ов);
* возможность импорта геометрических моделей, материалов, текстур, спрайтов и аудиодорожек со звуками.

По опросу, проведённому среди пользователей сайта Хабр, лидерами по популярности являются два игровых движка: Unity и Unreal Engine 4 [21]. Оба движка подходят по вышеперечисленным критериям, но в то же время имеют между собой пару различий в техническом плане:

* Основной язык разработки в Unity – C#.
* Основной язык разработки в Unreal Engine 4 – C++. В то же время движок располагает таким инструментом как Blueprints – визуальный язык для описания игровой логики, не требующий навыков программирования.

Unity делает больший упор на программирование, чем на дизайн, моделирование и прочее. Его порог вхождения ниже, чем у UE4, а студент знаком с языком разработки C#. Из-за этого был выбран игровой движок Unity для разработки демоверсии шутера с разрушаемым ландшафтом. При этом оба движка не имеют инструментов для работы с вокселями и воксельными объектами, что повышает актуальность данной работы.

Жизненный цикл игр, разрабатываемых в Unity, состоит из сцен. Сцены содержат в себе объекты, которые используются на данном игровом этапе, и ассеты – компоненты, которые к этим объектам прикрепляются. Это могут быть текстуры, материалы, аудио и видео файлы, и скрипты, отвечающие за описание и поведение объекта. В качестве объектов на сцене размещаются 3D и 2D объекты, такие как куб или спрайт, источники света, камера, эффекты, аудио и видео, и элементы пользовательского интерфейса. К каждому объекту прикрепляется набор ассетов и встроенных компонентов, благодаря чему объект получает описание, алгоритм поведения, характеристики и так далее (рис. 3.1).

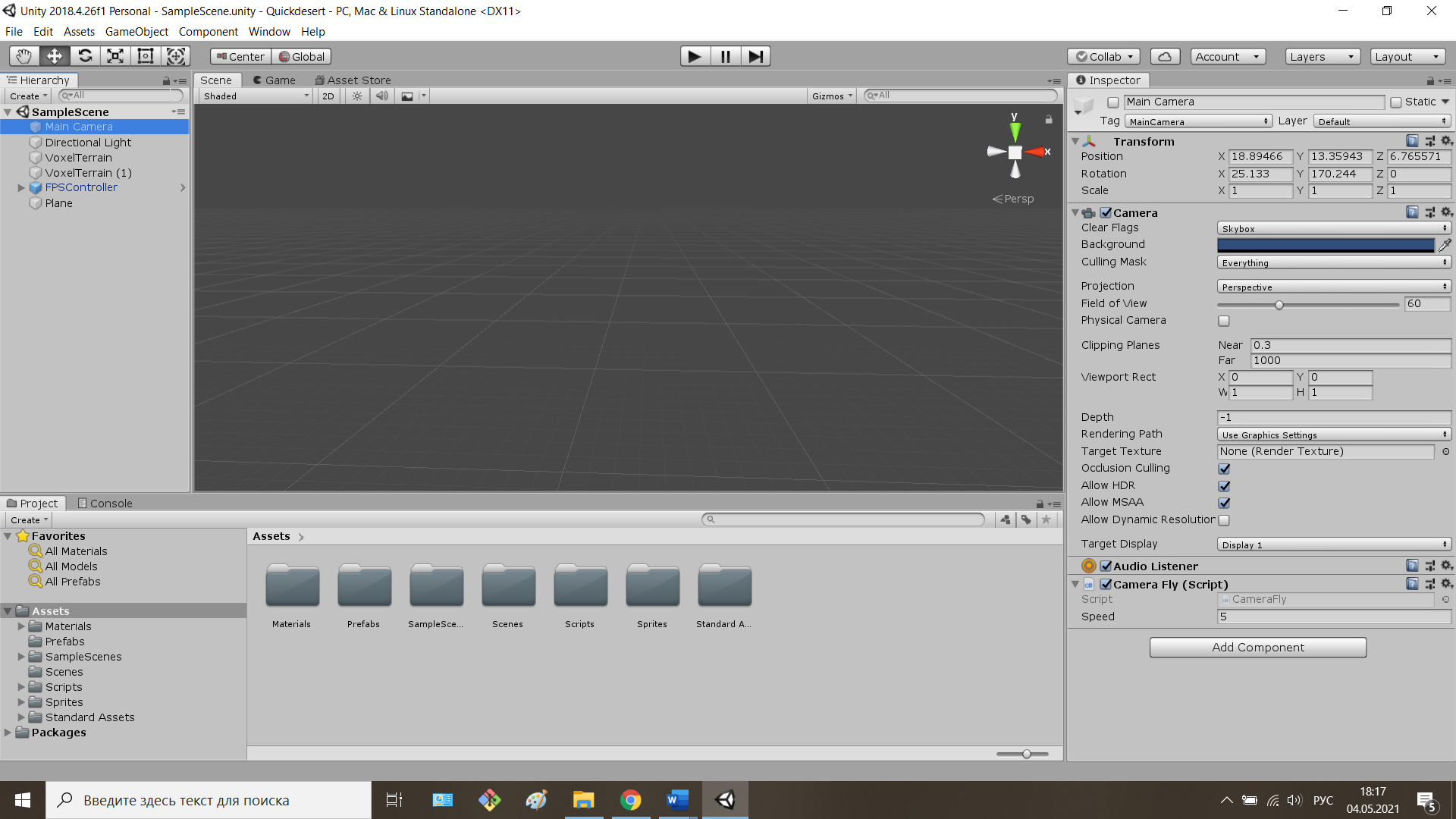


Рисунок . - Рабочее окно Unity

Основные встроенные компоненты, которые будут использованы:

* Rigidbody (твёрдое тело) – компонент, позволяющий объектам взаимодействовать с помощью физики. На тело начинает действовать гравитация, сопротивление воздуха, и им можно задать разную массу и скорость. Таким компонентом должны обладать снаряды, использующие физическую модель полёта, а также игровые персонажи.
* Mesh collider (обнаружение коллизий) – компонент, отвечающий за обнаружений коллизий между объектами. Таким компонентом должны обладать любые снаряды, для обнаружения попаданий в ландшафт и других персонажей, и персонажи, для взаимодействия с окружением.
* Transform – стандартный компонент, отвечающий за позиционирование объекта, его размер и угол поворота.

## 3.2. Разработка игры

Учитывая специфику выбранного движка, представим структуру системы в виде диаграммы компонентов (рис. 3.2):

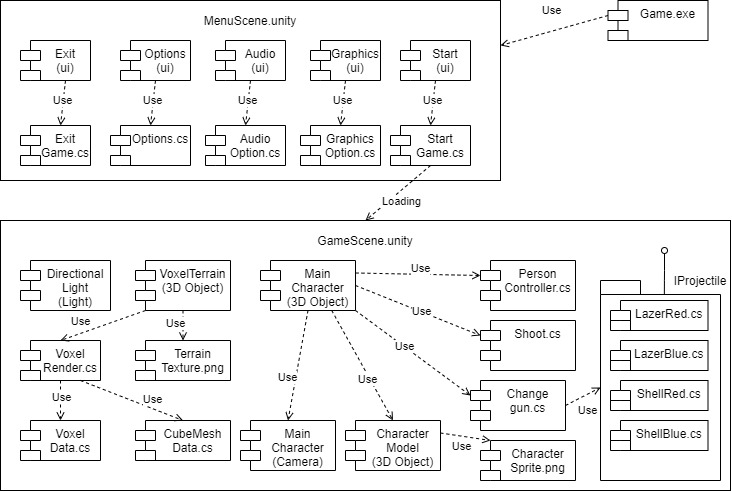


Рисунок . - Диаграмма компонентов системы

# Тестирование игрового процесса

Результатами главы является список сценариев для проверки игры и полученные ответы приложения.

Заключение

Библиографический список

1. Как растёт индустрия компьютерных игр и киберспорт в условиях пандемии [Электронный ресурс]. – Режим доступа: <https://www.kommersant.ru/doc/4325738> (дата обращения 06.12.2020).
2. Игры с лучшей разрушаемостью [Электронный ресурс]. – Режим доступа: <https://cubiq.ru/igry-s-luchshej-razrushaemostyu/> (дата обращения 01.03.2021).
3. Scorched 3D Version 43.2 beta released [Электронный ресурс]. – Режим доступа: <https://www.webcitation.org/666BwPZfZ?url=http://scorched3d.co.uk/phpBB3/viewtopic.php?t=6268> (дата обращения 13.02.2021).
4. Terrain in Battlefield 3: A modern, complete and scalable system [Электронный ресурс]. – Режим доступа: <https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/gdc12-terrain-in-battlefield3.pdf> (дата обращения 13.02.2021).
5. Battlefield 3. Планы на переворот [Электронный ресурс]. – Режим доступа: <http://mgnews.ru/read-news/battlefield-3-plany-na-perevorot> (дата обращения 13.02.2021).
6. Gregory J. Game Engine Architecture – 3rd ed. – Boca Raton.:CRC Press, 2018. – P. 525-544.
7. Mistal, B. GPU Terrain Subdivision and Tessellation // GPU Pro. – 2018. – Vol. 360. – P. 177-194.
8. Pangerl, D. Dynamic GPU Terrain // GPU Pro. – 2018. – Vol. 360. – P. 223-237.
9. Akenine-Moller, T., Haines, E. Real-time Rendering – 4th ed. – Boca Raton.:CRC Press, 2018. – P.545-589.
10. Разрушаемость в Teardown: как совместить геймплей и технологии [Электронный ресурс]. – Режим доступа: <https://media-xyz.com/ru/articles/714-razrushaemost-v-teardown-kak-sovmestit-geimple> (дата обращения 22.01.2021).
11. NVIDIA Blast [Электронный ресурс]. – Режим доступа: <https://developer.nvidia.com/blast> (дата обращения 22.01.2021).
12. Pätzold, M. Graphical model and simulation for THz-imaging: dis. Dok. der Ing. – Siegein, 2017. – S. 9.
13. Schawaz, M., Seidel, H. P. Fast Parallel Surface and Solid Voxelization on GPUs // ACM Transations on Graphics. – 2010. – Vol.29. – P.179:1-179:10.
14. Baert, J., Lagae, A., Dutre, Ph. Out-of-Core Construction of Sparse Voxel Octrees // Computer Graphics Forum. – 2013. – Vol. 33. – P.220-227.
15. Kotfis, D., Cozzi, P. Octree Mapping from a Depth Camera // GPU Pro. – 2016. – Vol. 7. – P. 257-273.
16. Создание карт из функции шума [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/430384/> (дата обращения 26.03.2021).
17. Implementing Improved Perlin Noise [Электронный ресурс]. – Режим доступа: <https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-26-implementing-improved-perlin-noise> (дата обращения 25.03.2021).
18. McEwan, I., Sheets, D., Richardson, M., Gustavson, S. Efficient Computational Noise in GLSL // journal of graphics tools. – 2012. – Vol. 16. – P. 85-94.
19. Perlin K. Standard for perlin noise // Patent USA №US6088036A. 11.07.2000.
20. Palmer, G. Physics for Game Programmers – New York.:Apress, 2018. – P. 83-138.
21. Лучший игровой движок по версии пользователей Хабра [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/307952/> (дата обращения 04.05.2021).

Приложение А. Техническое задание

**ДеМОВЕРСИЯ ИГРЫ С РАЗРУШАЕМЫМ ЛАНДШАФТОМ**

**Техническое задание**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Руководитель разработки: | |
|  |  | \_\_\_\_\_\_\_\_\_\_\_Г. И. Рустамханова | |
|  |  |  | |
|  |  | «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2021 | |
|  |  |  | |
|  |  | Исполнитель: | |
|  |  | \_\_\_\_\_\_\_\_М. Р. Шевчук | |
|  |  | «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2021 | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. Введение

1.1. Наименование программы

Наименование программы – «Демоверсия игры с разрушаемым ландшафтом» (далее по тексту – игра)

1.2. Краткая характеристика области применения

Программа предназначена к применению в любых условиях, способствующих запуску ЭВМ, пользователями-игроками.

1. Основание для разработки

2.1. Основание для проведения разработки

Основанием для проведения разработки является Приказ №8.2.6.2-13/1212-06 от 12 декабря 2021 года. Приказ утверждён старшим преподавателем кафедры информационных технологий в бизнесе «НИУ ВШЭ – Пермь» Рустамхановой Гульшат Ильдаровной, именуемым в дальнейшем Заказчиком, и утверждён Студентом «НИУ ВШЭ – Пермь» направления «Программная инженерия» Шевчуком Михаилом Романовичем, именуемым в дальнейшем Исполнителем 12 декабря 2021 года.

2.2. Наименование и условное обозначение темы разработки

Наименование темы разработки – «Проектирование и реализация игры с разрушаемым ландшафтом».

Условное обозначение темы разработки (шифр темы) – «TD-322».

1. Назначение разработки

Деформация ландшафта в играх является более трудной задачей, чем разрушение окружающего пространства, из-за большого количества требующихся вычислений и дополнительной нагрузки на память. Вследствие чего не все игровые жанры могут эффективно использовать эту механику, в частности игровой жанр «шутер», в котором уровень динамики особенно высок. Разрабатываемая демоверсия представляет собой попытку найти подход к использованию механики деформации ландшафта в играх жанра шутер. Демоверсия в дальнейшем послужит основой для полноценной игры, что может привести к расширению спектра игровых механик и жанров на рынке в случае успеха продукта.

3.1. Функциональное назначение

Функциональным назначением программы является предоставление игроку возможности поиграть в игру и опробовать новую механику среди большого спектра игр на рынке, а также развитие тактических и стратегических сторон игрока.

3.2. Эксплуатационное назначение

Программа должна эксплуатироваться в любых условиях, способствующих запуску ЭВМ, пользователями-игроками/покупателями/продавцами.

Пользователями программы должны являться пользователи-игроки в компьютерные игры.

1. Требования к программе или программному изделию

4.1. Требования к функциональным характеристикам

4.1.1. Требования к составу выполняемых функций

Программа должна обеспечивать возможность выполнения перечисленных ниже функций (табл. А.1).

Таблица А.1 – Список функций, обязательных к выполнению

| **№** | **Входная информация** | **Процесс** | **Выходная информация** |
| --- | --- | --- | --- |
| 1 | Настройки, выставленные игроком | Изменение настроек при запуске игры | Игровой процесс с изменёнными настройками |
| 2 | Сигнал о перемещении персонажа в указанном направлении | Изменение местоположения персонажа на игровой карте | Изменённое состояние персонажа, рендеринг изменённого состояния игры |
| 3 | Сигнал о смене оружия на указанное | Смена оружия | Изменённое состояние персонажа |
| 4 | Сигнал о проведении выстрела из оружия | Выстрел из оружия | Изменённое состояние игры, рендеринг изменённого состояния игры |
| 5 | Сигнал о перезарядке оружия | Перезарядка оружия | Изменённое состояние персонажа |

4.1.2. Требования к организации входных данных

Входные данные программы должны быть организованы:

* в виде строки, содержащей информацию о разрешении экрана при настройке игры;
* в виде числа, содержащего информацию о громкости игровых звуков;
* в виде строки, содержащей набор клавиш для изменения управления персонажем;
* в виде сигналов о нажатии клавиш, нажатых при передвижении игрока, смене оружия, ведения стрельбы и перезарядке оружия.

4.1.3. Требования к организации выходных данных

Выходные данные программы должны быть организованны в виде рендеринга изменённого состояния игры, а именно изменённого состояния персонажей или игрового ландшафта.

4.1.4. Требования к временным характеристикам

Программа должна стартовать не более чем 30 секунд.

Игровой процесс должен стартовать не более чем 1 минуту.

4.2. Требования к надёжности

Выделены следующие типы отказов:

* ошибки, скрытые в самой программе (далее А);
* искажение входной информации (далее Б);
* неверные действия пользователя (далее В);
* неисправность аппаратных средств ИС, на которой реализуется вычислительный процесс (далее Г).

1. Среднее время восстановления после отказа:

* А: не более 10 секунд;
* Б: не более 10 секунд;
* В: не более 5 секунд;
* Г: не более 2 минут;

1. Классификация по серьезности и времени обнаружения:

* А: высокий уровень серьезности, время обнаружения – с появлением результата;
* Б: низкий уровень серьезности, время обнаружения – с запуском обработки входной информации;
* В: низкий уровень серьезности, время обнаружения – с началом обработки действия программой;
* Г: средне - высокий уровень серьезности (зависит от типа неисправности), время обнаружения – с момента появления неисправности;

1. Последствия отказов:

* А: невыполнение определённого действия/ошибочный результат;
* Б: отвержение некорректных данных;
* В: отвержение некорректного действия;
* Г: в зависимости от типа неисправности: некорректная работа программы – полное выключение;

1. Допускаемый объем данных, утрачиваемый в случае отказа:

* А: не более 10 Мбайт;
* Б: потеря данных не предусмотрена;
* В: потеря данных не предусмотрена;
* Г: не более 100 Мбайт;

1. Функции, необходимые для обнаружения и исправления ошибок:

* А: функция фиксирования ошибки, функция исправления ошибки не предусмотрена;
* Б: функция фиксирования некорректных входных данных и сообщение об этом пользователю;
* В: функция фиксирования некорректных действий и сообщение об этом пользователю;
* Г: функция обнаружения экстренного выключения устройства, функция сохранения данных, функция восстановления части утраченных данных;

4.2.1. Требования к обеспечению надёжного (устойчивого) функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

* организацией бесперебойного питания технических средств;
* использованием лицензионного программного обеспечения;
* регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
* регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов.

4.2.2. Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать двух минут при условии соблюдения условий эксплуатации технических и программных средств. Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных систем.

4.2.3. Отказы из-за некорректных действий оператора

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу пользователя без предоставления ему административных привилегий.

4.3. Условия эксплуатации

4.3.1. Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

4.3.2 Требования к видам обслуживания

Обслуживание программы должно обеспечиваться выполнением заказчиком совокупности организационно – технических мероприятий, перечень которых приведён ниже:

* организацией бесперебойного питания технических средств;
* использованием лицензионного программного обеспечения;
* регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
* регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов.

4.3.3. Требования к численности и квалификации персонала

Минимальное количество пользователей, требуемых для работы программы, должно составлять не менее 1 штатной единицы – пользователь программы – игрок.

Пользователь программы (игрок) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

4.4. Требования к составу и параметрам технических средств

В минимальный состав технических средств должен входить:

1.IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

* процессор Inter Core i5 – 7200U с тактовой частотой 2.71 ГГц , не менее;
* оперативную память объемом 6 Гб, не менее;
* жесткий диск объемом 256 Гб, и выше;
* оптический манипулятор типа «мышь»;
* наличие 2 COM-портов;
* видеокарта с поддержкой DirectX10 и выше, OpenGL 3.2 и выше;

2.Локальная сеть. Пропускная способность ЛС до 1Гб/сек.

3.Оборудование пользователей должно иметь доступ к Интернету.

4.5. Требования к информационной и программной совместимости

4.5.1. Требования к информационным структурам и методам решения

Требования к информационным структурам и методам решения не предъявляются.

4.5.2. Требования к исходным кодам и языкам программирования

Требования к исходным кодам и языкам программирования не предъявляются.

4.5.3. Требования к программным средствам, используемым программой

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows 10.

4.5.4. Требование к защите информации и программ

Информация, содержащаяся в исходном коде программы, не подлежит несанкционированному распространению и в случае нарушения грозит уголовной ответственностью согласно статье 272 УК РФ.

4.6. Требования к маркировке и упаковке

4.6.1 Требования к маркировке

Программное изделие должно иметь маркировку с обозначением товарного знака компании-разработчика, типа (наименования), номера версии, порядкового номера, даты изготовления и номера сертификата соответствия Госстандарта России (если таковой имеется).

4.6.2. Требования к упаковке

Требования к упаковке не предъявляются.

4.7. Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

4.8. Специальные требования

Программа должна обеспечивать взаимодействие с пользователем (игроком) посредством графического пользовательского интерфейса, разработанного согласно рекомендациям компании-производителя операционной системы.

1. Требования к программной документации

В предварительный состав программной документации должны входить:

* техническое задание;
* дизайн-документ;
* технико-экономическое обоснование;
* руководство пользователя;

1. Технико-экономические показатели

6.1. Ориентировочная экономическая эффективность

Программа послужит основой для разработки новых коммерческих продуктов, наличие которых в случае успеха расширит игровой рынок и повысит интерес к приобретению у пользователей – игроков и игровых студий.

6.2. Предполагаемая годовая потребность

Предполагаемое число использования программы в год – круглосуточная работа программы на большом количестве компьютеров пользователей.

6.3. Экономические преимущества разработки

Экономическими преимуществами разработки в сравнении с лучшими отечественными и зарубежными аналогами будет являться расширение игрового рынка и привлечение новой аудитории.

1. Стадии и этапы разработки

7.1. Стадии разработки

Разработка должна быть проведена в три стадии:

* стадия разработки технического задания;
* стадия разработки технических (и рабочих) проектов;

7.2. Этапы разработки

На стадии «Техническое задание» должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии «Технический (и рабочий) проект» должны быть выполнены перечисленные ниже этапы работ:

* разработка программы;
* разработка программной документации;
* испытания программы.

7.3. Содержание работ по этапам

На этапе разработки техзадания должны быть выполнены перечисленные ниже работы:

* постановка задачи;
* определение и уточнение требований к техническим средствам;
* определение требований к программе;
* определение стадий, этапов и сроков разработки программы и документации на нее;
* выбор языков программирования;
* согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ 19.101-77.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

* разработка, согласование и утверждение программы и методики испытаний;
* проведение приемо-сдаточных испытаний;
* корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах заказчика.

1. Порядок контроля и приемки

8.1. Виды испытаний

Приемосдаточные испытания должны проводиться на объекте заказчика в сроки 09.04.21 – 17.05.21.

8.2. Общие требования к приемке работы

На основании протокола испытаний исполнитель совместно с заказчиком подписывают акт приём-сдачи программы в эксплуатацию.

Приложение Б. Технико-экономическое обоснование проекта

Согласовано

Ответственное лицо со стороны заказчика

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

утверждаю

Ответственное лицо со стороны разработчика

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

*ДЕМОВЕРСИЯ ИГРЫ С РАЗРУШАЕМЫМ ЛАНДШАФТОМ*

**ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ**

(Исследование стоимости и окупаемости разработки системы)

**Лист утверждения**

(Вид носителя данных)  
Листов 16

Согласовано

Ответственное лицо со стороны заказчика

|  |  |
| --- | --- |
| *Подп. и дата* |  |
| *Инв. № дубл.* |  |
| *Взам. инв. №* |  |
| *Подп. и дата* |  |
| *Инв. № подл.* |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

Ответственное лицо со стороны заказчика

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

Представители предприятия-разработчика

Руководитель предприятия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

Начальник отдела

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

Руководитель разработки

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

Исполнитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

Нормоконтроллер

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г

2021

Введение

**Полное наименование системы:** Демоверсия игры с разрушаемым ландшафтом.

Технико-экономическое обоснование разрабатывается на основании спецификации, полученной по итогам выполнения технического (эскизного) проекта.

Общие сведения о системе

Деформация ландшафта в играх является более трудной задачей, чем разрушение окружающего пространства, из-за большого количества требующихся вычислений и дополнительной нагрузки на память. Вследствие чего не все игровые жанры могут эффективно использовать эту механику, в частности игровой жанр «шутер», в котором уровень динамики особенно высок. Разрабатываемая демоверсия представляет собой попытку найти подход к использованию механики деформации ландшафта в играх жанра шутер. Демоверсия в дальнейшем послужит основой для полноценной игры, что может привести к расширению спектра игровых механик и жанров на рынке в случае успеха продукта.

Технико-экономическое обоснование разработки и внедрения системы

Под себестоимостью системы будем понимать общие затраты разработчика на создание программного продукта. Длительность разработки программного продукта – три месяца.

Себестоимость системы (или смета затрат на создание программного продукта) определяется по следующим экономическим элементам:

* Материальные затраты;
* Затраты на оплату труда + отчисления на социальное страхование;
* Амортизация основных фондов + аренда основного капитала;
* Прочие затраты.

Исходные данные

При расчете себестоимости системы будем принимать во внимание следующие данные:

Составить смету на разработку программного продукта;

Составить спецификацию на комплекс программно-технических средств программного продукта.

Смета на разработку программного продукта

Таблица Б.1 – Смета на разработку ПО

| **Код статьи** | **Наименование статьи расходов** | **Сумма (руб.)** |
| --- | --- | --- |
| 110100  и  110200 | Затраты на оплату труда работников, непосредственно занятых созданием научно-технической продукции, а также начисления на оплату труда | 205 841 |
| 110400 | Командировочные расходы | 0 |
| 110350 | Приобретение предметов снабжения и расходных материалов,  *в том числе НДС 18%* | 101 304 |
| 110350 | Материалы и комплектующие (системное программное обеспечение, компьютерная техника и т.п.),  *в том числе НДС 18%* | 80 640 |
| 111010 | Затраты по работам, выполняемым сторонними организациями и предприятиями | 0 |
| 111040 | Прочие текущие расходы (издание печатной продукции),  *в том числе НДС 10%* | 0 |
|  | **Итого** | **387 785** |
|  | **в том числе НДС** |  |

Спецификация расходов на материалы и комплектующие (системное программное обеспечение, компьютерная техника, аппаратура передачи данных и т. д.)

Таблица Б.2 – Спецификация расходов на материалы и комплектующие

| **Наименование предметов и материалов** | **Единица измерения** | **Кол-во** | **Цена ед. (рублей)** | **Сумма (рублей)** |
| --- | --- | --- | --- | --- |
| Unity Pro | Подписка | 1 | 11 400/мес. | 34 200 за 6 месяцев |
| Visual Studio Enterprise | Подписка | 1 | 19 000/мес. | 57 000 за 6 месяцев |
| ИТОГО: | | 2 |  | 91 200 |

Материальные затраты

Под материальными затратами понимают расходы на приобретение сырья и материалов, которые входят в состав изготавливаемой продукции и обеспечивают нормальное течение технологического процесса, на полуфабрикаты и комплектующие изделия, топливо, энергию и др. Сумма материальных затрат приведена в таблице ниже (см. табл. Б.3).

Таблица Б.3 – Таблица материальных затрат

| **Наименование материального ресурса** | **Ед. изм.** | **Цена 1 ед. потребляемого ресурса, руб.** | **Расход ресурса, ед./мес.** | **К-во месяцев потребления** | **Стоимость потребляемого ресурса, руб.** |
| --- | --- | --- | --- | --- | --- |
| Бумага | Пачка | 300 | 1 | 3 | 900 |
| Флэш накопитель | Экземпляр | 1200 | 1 | 3 | 1200 |
| Картридж | Упаковка | 600 | 1 | 3 | 1800 |
| ИТОГО: | | | | | 3900 |

Кроме указанных ресурсов к материальным затратам относят также и электроэнергию, потребляемую компьютерами и необходимую для освещения рабочих мест программистов.

, где

*KПК =2*. – количество используемых компьютеров;

*РПК* = 220 – мощность, потребляемая компьютером;

*Росв* = 15 Вт – мощность, потребляемая на освещение 1 м2 площади;

*С эл.эн.*=3 руб/кВт ч = 0.03руб/Вт ч – тариф на электроэнергию;

*S* = 20 м^2 – площадь рабочего места программистов;

*tраб.час* = 440– количество рабочих часов.

Рабочее время потребления электроэнергии рассчитано следующим образом:

Согласно исходным данным, срок создания системы равен 3 месяцам.

Количество рабочих суток *tраб.сут* равно:

 55 дн.

При учете 8-часового рабочего дня получаем следующую величину количества рабочих часов *tраб.час*:

 440 час.

Таким образом, затраты на оплату потребляемой электроэнергии составляют: 6 204 руб.

Таким образом, общая сумма материальных затрат составляет:

 91 200 + 3 900 + 6 204 = 101 304 руб.

Расчёт прямых затрат

Для расчета прямых затрат на разработку и внедрение системы необходимо определить следующие статьи расходов:

* Материалы, покупные изделия;
* Основная зарплата персонала;
* Дополнительная зарплата персонала;
* Отчисления на социальные нужды.

Зарплаты на оплату труда персонала

Персонал, необходимый для создания системы (табл. Б.4).

Таблица Б.4 – Таблица заработных плат сотрудникам

| **Категория работников** | **Кол-во, чел.** | **Заработная плата, руб./мес.** |
| --- | --- | --- |
| Технический персонал (программисты) | 1 | 40000 |

Длительность разработки программного продукта: 3 месяца.

***Затраты на оплату труда основного (технического) персонала:***

Основная заработная плата основного персонала – Cосн.зар:

, где

*tраб.* = 3 – количество месяцев, необходимых для разработки;

Cосн.з/пл.програм. = 40000 руб. – месячная зарплата программистов;

Cосн.з/пл.техн.персон = 0 руб. – месячная зарплата технического персонала;

Кпрограм. = 1 численность программистов;

Ктехн.персонал = 0 численность технического персонала;

В итоге получаем:

 120 000 руб.

***Дополнительная заработная плата основного персонала:***

Дополнительная зарплата персонала Cдоп.з/пл установлена в размере 10% премиальных выплат, базой для расчета служит основная заработная плата персонала.

0.1 = 12 000 руб.

Законодательно установлено, что в районах Крайнего Севера и иных регионах, приравненных к районам Крайнего Севера, вводится районный коэффициент, начисляемый на фактическую заработную плату. Для Уральского региона этот коэффициент определен в размере 15%.

Таким образом, на сумму основной и дополнительной заработной платы начисляем уральский коэффициент:

 0.15 = (120 000 + 12 000) \* 0.15 = 19 800 руб.

***Затраты на оплату труда основного персонала:***

 120 000 + 12 000 + 19 800 = 151 800 руб.

***Отчисления на социальное страхование***:

Заработная плата является базой для исчисления единого социального налога.

Государство должно обеспечить определенную социальную защиту граждан: медицинское обслуживание, пособие по старости, пособие по безработице, пособие по временной нетрудоспособности. Государство возлагает эти функции на предприятие, устанавливая определенные отчисления на социальное страхование во внебюджетные фонды. Единый социальный налог определен в размере 35,6%.

 0.356 = 151 800 \* 0.356 = 54 041 руб.

Затраты на оплату труда управленческого персонала:

Затраты на оплату труда управленческого персонала определяются по той же схеме, что и основного (технического) персонала.

 0 руб.

 0 = 0 руб.

 0 руб.

 0 руб.

0 руб.

Таким образом, затраты на оплату труда основного (технического) и управленческого персонала, требуемого для разработки программного продукта, и отчисления на социальное страхование составляет:



151 800 + 54 041 = 205 841 руб.

Амортизация основных фондов

В данный экономический элемент включается сумма амортизационных отчислений на полное восстановление основных производственных фондов предприятия по собственным и арендованным фондам.

***Амортизация оборудования***

В данной статье должна быть отражена сумма амортизации оборудования предприятия, используемого для производства изделий (оказания услуг, осуществлению работ). Программные и аппаратные средства, необходимые для разработки программного обеспечения системы (табл. Б. 5), находятся на балансе предприятия. Рассчитаем влияние амортизационных отчислений на себестоимость создаваемой системы.

Таблица Б.5 – Таблица программных и аппаратных средств

| **Наименование товара** | **Количество единиц** | **Цена за единицу,**  **руб.** | **Общая стоимость, руб.** |
| --- | --- | --- | --- |
| Персональный компьютер | 2 | 40000 | 80000 |
| Microsoft Visual Studio 2019 | 1 | 19 000/мес | 57 000 |
| Windows 10 professional | 2 | 4000 | 8000 |
| Unity Pro | 1 | 11 400/мес | 34 200 |

Поскольку срок службы компьютерной техники 7 лет, а программного обеспечения, продаваемого по академической лицензии 15 лет, соответственно месячная норма амортизации составит:

 = 14 %

и для программного обеспечения:

 7 %

Месячная величина амортизационных отчислений для компьютерной техники составит:

 80 000 \* 0.14 = 11 200 руб.

и для программного обеспечения:

 (57 000 + 8000+ 34 200) \* 0.7 = 69 440 руб.

Таким образом, к концу периода разработки все амортизационные отчисления используемых программных и аппаратных средств будет перенесена на себестоимость создаваемой системы и составит:

 11200 + 60039 = 80 640 руб.

***Амортизация зданий, сооружений***

Рассчитаем амортизацию здания исходя из периода жизни здания в 20 лет. Стоимость помещения, используемого для работы группы разработки программного обеспечения равна 2 000 000 руб.

Годовая норма амортизации:



Месячная величина амортизационных отчислений составляет:

 2 000 000 \*5/12 = 833 333 руб.

Величина амортизационных отчислений зданий (сооружений) за период разработки:

 833 333 \* 3 = 2 500 000 руб.

В существующей разработке программного обеспечения прочие затраты не учитываются.

Общие затраты на разработку программного продукта сведены в таблицу ниже (табл. Б. 6).

Таблица Б.6 – Общие затраты на разработку

| № п/п | Наименование экономического элемента | Сумма, руб. |
| --- | --- | --- |
| 1 | **Материальные затраты** | **3 900** |
|  | Бумага | 900 |
|  | Флэш-накопитель | 1 200 |
|  | Картридж | 1 800 |
|  |  |  |
| 2 | **Оплата труда персонала** | **54 041** |
|  | Основной персонал | 54 041 |
|  | Управленческий персонал | 0 |
|  |  |  |
|  |  |  |
| 3 | **Амортизация основных фондов** | **2 580 640** |
|  | Амортизация оборудования | 80 640 |
|  | Амортизация зданий, сооружений | 2 500 000 |
| 4 | **Прочие затраты** | 0 |
|  | **ИТОГО: затраты на разработку продукта** | **2 638 581** |

Расчёт цены на программный продукт

Произведем расчет цены на программный продукт *Цп.п.* методом полных издержек. Указанный метод применяется потому, что разрабатываемый программный продукт не предназначен для тиражирования. В силу постановки задачи программный продукт будет использоваться для внутренних потребностей промышленного предприятия.

Нормативная прибыль определена в размере 8,5 % от себестоимости продукта.

Таким образом,

, где *Сп.п.* = 2 638 581 руб. – затраты на разработку программного продукта;

 2 638 581 \* 0.085 = 224 279,385 руб. – нормативная прибыль (надбавка).

В итоге получаем:

2 862 860, 385 руб.

Округляя полученный результат, получим окончательную цену программного продукта разрабатываемой системы, равного 2 862 861 руб.

Расчёт эффективности инвестиций

Расчет инвестиционных вложений

Из вышеописанных разделов известна стоимость программного продукта, т.е. объем затрат на этапе разработки:

Цп.п. = 2 862 861 руб.

Определим некоторые требования к системе на этапах внедрения и функционирования (эксплуатации) системы.

Каждый этап требует различное количество технического и обслуживающего персонала (табл. Б.7).

Таблица Б.7 – Общие затраты на разработку

|  |  |  |
| --- | --- | --- |
| **Наименование этапа** | **Длительность этапа мес.** | **Потребность в персонале** |
| Внедрение системы | 3 | Количество тех. персонала (программисты) – 1 чел. |

Поскольку существующая инфраструктура коммуникаций, технических средств, и программного обеспечения полностью удовлетворяют требованиям системы, то установка системы производится на закупленное компьютерное оборудование и программное обеспечение.

Определим заработную плату для технического персонала на стадиях внедрения и функционирования из расчета 40000 руб./мес. на одного человека.

Общие затраты на внедрение системы определяются следующим образом:

, где

Соб. = 0 руб. – стоимость оборудования. (Оборудование заказчика)

Сз/пл – затраты на оплату труда технического персонала

 3\*1\*40000 + 40000\*0.356 = 134 240 руб.

134 240 руб.

***Общие затраты на этапе функционирования (эксплуатации) системы*** составляют расходы на оплату труда технического персонала, которые равны:

 0 руб.

Таким образом, инвестиционные (*I*) вложения составляют: 2 997 101 руб.

 2 862 861 + 134 240 + 0 = 2 997 101 руб.

При определении эффективности инвестиционного проекта важно распределить инвестиционные вложения во времени, в качестве размера шага принят период в 1 месяц.

Инвестиционные затраты в течение всего срока функционирования системы представлены в таблице ниже (см. табл. Б. 8).

Таблица Б.8 – Общие затраты на разработку

|  |  |  |
| --- | --- | --- |
| **Период** | **Составляющие инвестиционных затрат** | **Сумма инвестиционных затрат, руб.** |
| Первоначальные инвестиции | - затраты на приобретение программного продукта;  - затраты на приобретение оборудования | 2 862 861 |
| Инвестиции 1-го периода | - затраты на оплату труда технического персонала на этапе внедрения  - затраты на оплату труда технического персонала на этапе функционирования (оплата труда в течении 3 месяцев эксплуатации) | 134 240 |

Определение экономического эффекта

Экономический эффект от использования программного продукта выражается в экономии затрат на оплату труда, материалов и иных расходов.

***Эффект от экономии труда***

Средняя заработная плата программиста равна 40000 руб./мес.

Премия определена в размере 10% средней заработной платы специалиста;

Величина уральского коэффициента – 15%;

Отчисления на социальное страхование (единый социальный налог) – 35,6%.

Система рассчитана на 1 пользователя.

Месячные затраты на оплату труда одного пользователя во время внедрения системы составляет:

1\* (40000 + 40000\*0.1 + 40000 \* 0.15) = 1 \* (40000 + 4000 + 6000) = 50000 руб.

Сумма отчислений на социальное страхование составляет:

17 800 руб.

Исходя из этих данных, за счет увеличения интенсификации труда специалистов на 50%, определим месячную экономию трудовых затрат:

 (50 000 + 17 800) \* 0.5 = 33 900 руб./мес.

 33 900 \* 3 = 101 700 руб.

Оценка инвестиционного проекта

Оценка инвестиционного проекта включает в себя анализ его доходности.

Для того чтобы оценить выгодность инвестиций в разработку программной системы мониторинга производственно-инновационных процессов предприятия, необходимо рассмотреть динамические методы, которые учитывают изменение стоимости денежных средств во времени.

***К динамическим методам оценки относят следующие:***

Метод чистого дисконтированного дохода (ЧДД);

Метод индекса доходности (ИД);

Метод внутренней нормы доходности (ВНД).

Динамические методы оценки инвестиционных проектов предполагают наличие ставки дисконтирования, т.е. минимально допустимой ставки процента, при которой инвестиции могут считаться эффективными. Процесс определения ставки дисконтирования, ее объективного выражения достаточно сложен. Существует несколько методов определения ставки дисконтирования:

В качестве ставки дисконтирования берут один из следующих показателей:

* Депозитная ставка;
* Ставка кредитования;
* Ставка рефинансирования.

Рассматривается как альтернативный вариант вложения.

* В качестве ставки дисконтирования берут доходность, которую предприятие задает для себя (в стратегии);
* В качестве ставки дисконтирования принимают сумму следующих показателей:
* Ставка безрисковых вложений;
* Уровень инфляции;
* Риск от данного вида инвестиций.
* Определим ставку дисконтирования:

, где

r – ставка безрисковых вложений (в государственные ценные бумаги);

i – годовой уровень инфляции;

P –поправка на риск, %.



Соизмерение разновременных показателей осуществляется путем при­ведения (дисконтирования) их к ценности в начальном периоде.

Расчет чистого дисконтированного дохода

Под ЧДД понимается разница между дисконтированным доходом от инвестиций и суммой дисконтированных стоимостей всех затрат.



Расчет ЧДД для данного инвестиционного проекта представлен в таблице ниже (табл. Б. 9):

Таблица Б.9 – Расчёт ЧДД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Период** | **Инвестиционные вложения, руб.** | **Экономический эффект, руб.** | **Коэффициент дисконтирования** | **Текущий дисконтированный доход (ТДД), руб.** |
| 0 | 2 862 861 | 0 | 11 | -2 862 861 |
| 1 | 134 240 | 101 700 | 11 | -2 895 401 |
| **Итого** |  |  |  | **-2 895 401** |

Итак, к концу периода функционирования системы

ЧДД = -2 895 401 руб.

Критерием для принятия решения на основании метода ЧДД является максимизация значения ЧДД.

Если значение ЧДД> 0, то проект приемлем.

ЧДД -2 895 401 руб. <0 ⇒ инвестиционный проект неэффективен.

***Расчет индекса доходности (ИД)***

Индекс доходности (ИД) представляет собой отношение суммы при­веденных эффектов к величине капиталовложений

Этот показатель позволяет определить относительный рост денежных доходов и инвестиций (затрат).

 =

Если величина ИД > 1, то проект эффективен.

ИД = 1,3 > 1 ⇒ 1 рубль инвестиций приносит предприятию доход 1,3 руб.

Расчет внутренней нормы доходности инвестиций

Метод расчета ВНД позволяет оценить эффективность капитальных вложений путем сравнения внутренней (предельной) нормы окупаемости инвестиций с эффективной ставкой процента. ВНД соответствует такой ставке процента, при которой достигается нулевой чистый дисконтированный доход.

ВНД можно определить с помощью функции:

, где

k1, k2 – коэффициенты, при которых f(k) = ЧДД принимает различные знаки, т.е.

ВНД= 31%

Таким образом, при ставке дисконтирования, равной 31%, чистый дисконтированный доход будет равен нулю. Полученное значение ВНД свидетельствует об эффективности инвестиционного проекта.

Расчет периода окупаемости инвестиционного проекта

Период окупаемости инвестиционного проекта определяется по формуле:



Для расчета периода окупаемости определим инвестиции, приведенные к одному году функционирования проекта, а также годовой суммы экономии.

I – 2 862 861 + 134 240 = 2 997 101 руб.

Ээкон. год. – 101 700 руб.

 2 997 101/ 101 700 = 29, 47 лет.

Таким образом, период окупаемости проекта в виде демоверсии равен 29 годам 6 месяцев.