ВВЕДЕНИЕ

“Индустрия компьютерных игр зародилась в середине 1970-х. годов как движение энтузиастов и за несколько десятилетий выросла из небольшого рынка в мейнстрим\* с годовой прибылью в 9,5 миллиардов долларов в США в 2007 году и 11,7 миллиардов в 2008”.

Для привлечения числа пользователей, заведения стали массово скупать и использоваться игровые автоматы. Это был один из первых и основных шагов для привлечения пользователей к видеоиграм, которые не могли в те времена покупать персональные компьютеры и приставки для личного пользования. Из-за отсутствия мощностей первых игровых автоматов о графика представляла из себя набор из простых спрайтов\*, а игровой процесс состоял в том, чтобы победить соперника загнав мяч в свободное поле. Одной из первых игр был Ping-Pong. До 1980 года игры были исключительно на аркадных автоматах. В 1980 году компания Nintendo\* выпускает карманное портативное устройство, которое содержало в себе игру по ловле яиц в корзину, в том же году эта компания начала выпускать игры на персональные компьютеры, первой их игрой была “Don King Kong”.

Дальнейшем развитием был выпуск персональных приставок, основной серий игр были “Mario”, от компании “Nintendo”. Далее новые машины не появлялись, единственными изменениями являлись наращивания мощностей уже имеющихся ЭВМ.

Бюджеты первых игр не превышали миллиона долларов и разрабатывались небольшими командами, штат которых не превышал 100 человек. Продажи первых копий игр не превышали десятков миллионов долларов. Основываясь на данных популярной серии игр GTA, можно сделать выводы. GTA III выпущенная в 2001 году продалась тиражом 17.5 миллионов копий. В 2022 году игры крупными компаниями разрабатываются штатом в количестве 2000 человек и более. Бюджеты на разработку игры могут достигать десятки, а то и сотни миллионов долларов. GTA V выпущенная в 2013 имеет бюджет 265 миллионов долларов, а продажи по последним оценкам составили более 160 миллионов копий. Исходя из открытых источников в сети Интернет за 2021 год прибыль компании составила 6.4 миллиарда долларов.

Сегодня игры - выходят за рамки своего привычного понимания. Меняется концепция игр, создаются отдельные вселенные, в которых игрок может принять облик кого угодно, стать кем угодно. Компании разработчики экспериментируют, переносят свои игры с экрана компьютера в экраны шлемов виртуальной реальности\*. Такой подход полностью меняет поведение игрока, погружает его в созданный большой командой новый мир\*. Компания Meta\*(запрещена на территории Российской Федерации), сегодня направила все свои силы на создание метавселенной\*. В метавселенной игроку дозволено делать все, что он захочет, что не нарушает действующие законодательства страны, в которой она работает. Она позволяет общаться с друг другом используя аватары\* имитируя реальное общение. Также в метавселенной можно учиться и работать, не покидая порога своей квартиры. Всецело ведётся разработка костюмов, покрывающих все тело, для того чтобы, пользователь смог ощутить тактильные прикосновения сталкиваясь с виртуальными предметами и собеседниками, как реальными игроками, так и искусственно созданными.

Студии делятся на несколько типов - инди разработчики, средние разработчики и крупные ААА разработчики. Разработчики из средних студий и крупных в основном имеют издателя и располагают большим штатом сотрудников, начиная от гейм дизайнеров\*, разработчиков и дизайнеров уровней, заканчивая специалистами по оптимизации. Издатели занимаются всем, что не касается части разработки. Издатель руководит процессом, устанавливает сроки разработки, продвигает продукт в СМИ и другие площадки. Также издатель выделяет инвестиции на весь этап разработки игры. Издатели принимают решение войдет та или иная идея в финальную версию игры, что ставит разработчиков в определенные рамки и не дает полной свободы мысли, какой обладают инди разработчики. Инди разработчики могут иметь издателя и не иметь его вовсе, разрабатывая игру на энтузиазме. Отличия издателя инди от крупных и средних разработчиков заключается в том, что издатели менее требовательны и дают студиям свободу мысли, что впоследствии может привести игру к большему успеху в сравнении с крупными студиями, поэтому в нашем сценарии выбор очевиден - инди разработчики без дополнительного инвестирования. Разработчик Super Meat Boy сказал - “Так странно выпускать игру одновременно с Fallout и получать отзывы чуть ли не лучше, чем у неё. А я ведь и сам собирался купить Fallout. Это странно. Это чертовски круто!”

В настоящее время огромную популярность имеют игры в жанре шутер\*. С каждым годом на рынке появляются все больше однотипных и похожих друг на друга шутеров, которые не привносят ничего нового в жанр, а лишь копируют друг друга. Каждая новая часть популярной франшизы\* похожа на предыдущую с небольшими визуальными изменениями. Нашим решением было объединить два совершенно разных жанра - шутер и платформер. Вдохновение на создание игры повлияли, такие игры как “Katana Zero” и “My Friend Pedro”.

**Объектом исследования** выпускной квалификационной работы, является создание демо-версии двумерной динамической экшн игры в жанре платформера

**Предметом исследования** выпускной квалификационной работы, является разработка компьютерной видеоигры на операционной системе Windows, в которой люди могут провести свое время с целью развлечения.

**Задачи:**

* сформировать видение проекта;
* сформировать актуальность проекта;
* проанализировать конкурентов;
* проанализировать возможности рынка;
* проанализировать обороты рынка;
* разработать бизнес-модель;
* разработать монетизацию;
* разработать продвижение продукта;
* разработать компьютерную видеоигру на операционной системе Windows;
* проверить гипотезы;
* сформировать метрики.

**Цель работы:** заполнить пустой сегмент российской индустрии игр, путем разработки видеоигры на операционной системе Windows.

Для разработки компьютерной видеоигры на операционной системе Windows необходимо завершить следующие этапы разработки:

* разработка концепции компьютерной видеоигры;
* проектирование видеоигры;
* построение уровней;
* разработка скриптов;
* установка системы управления;
* разработка алгоритмов;
* разработка сюжета;
* наполнение уровня.

1. ОПИСАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметная область – игровая индустрия, одиночные видеоигры.

Игровая индустрия – индустрия компьютерных видеоигр (также индустрия интерактивных развлечений) — сектор экономики, связанный с разработкой, продвижением и продажей компьютерных видеоигр. В нее входит большое количество специальностей, по которым работают десятки тысяч человек по всему миру [1].

Разработчик занимается разработкой видеоигр. Обычно разработкой занимаются команда разработчиков, а их спонсором является крупная компания-издатель, которая в свою очередь берёт на себя все затраты связанные с изданием видеоигры и ее распространением. Также команда разработчиков может выпускать свои видеоигры, без помощи компании-издателя, используя свои средства для выпуска и продвижения видеоигры.

Steam – онлайн-сервис цифрового распространения компьютерных видеоигр и программ, разработанный и поддерживаемый компанией Valve. Steam выполняет роль средства технической защиты авторских прав, платформы для многопользовательских видеоигр и потокового вещания, а также социальной сети для игроков. Программный клиент Steam также обеспечивает установку и регулярное обновление видеоигр, облачные сохранения видеоигр, текстовую и голосовую связь между игроками [2].

Компания Valve была основана 24 августа 1996 года двумя бывшими сотрудниками Microsoft - Гейбом Ньюэллом и Майком Харрингтоном, в Киркланде. Издателем Valve стала Sierra, пережившая в 1996 году тяжёлые времена. Для упрощения работы Valve приобретает лицензию Quake Engine, на котором базировалась видеоигра Quake. С этого момента начинается разработка Half-Life.

Рабочим названием видеоигры было "Quiver". Для работы над сценарием был приглашен писатель-фантаст Марк Лэйдлоу. Видеоигра стала научно-фантастическим трехмерным шутером от первого лица, так как Ньюэлл и Харрингтон считали, что экшн является единственным жанром, где можно придумать что-нибудь новое. По задумке атмосфера Half-Life должна была походить на мрачную атмосферу Doom и в некотором роде должна была стать ее переосмыслением. Концепция мира видеоигры должна была походить на повесть Стивена Кинга "Туман".

Впервые видеоигру показали на выставке E3 в 1997 году, где она стала настоящим хитом и получила награду лучшая видеоигра шоу. Релиз планировался на конец года, однако, по техническим причинам был перенесён. Ближе к концу 1997 года авторы видеоигры решили, что большую часть сделанного необходимо переделать. Параллельно с Half-Life шла разработка другого не анонсированного проекта, которым, вероятно, должен был стать Prospero, но была отменена для сосредоточения сил на разработке Half-Life.

Сначала Valve собиралась выпустить видеоигру весной, затем в июне, потом просто летом, затем в сентябре и после этого, наконец, на Рождество. Постоянные переносы даты релиза служили причиной беспокойства в Sierra. На выставке E3 1998 года Half-Life снова получил награду лучшая видеоигра шоу. До выпуска Half-Life компания выпустила демоверсию под названием Day One. Вскоре её стали бесплатно распространять вместе с видеокартами. Уже вскоре демоверсия обрела огромную популярность.

Когда разработка была близка к завершению оставалась лишь одна маленькая программная ошибка. Большая часть сотрудников уже ничего не делала. Когда ошибка была, наконец, найдена, была записана копия видеоигры, которая была отправлена на тестирование. Релиз видеоигры был отмечен в Valve уничтожением хедкраба-пиньяты. Наконец, после окончания разработки, по словам Марка Лэйдлоу, люди вернулись к уже забытой ими обычной жизни. После годового отпуска эти люди снова собрались вместе для работы над Team Fortress, а затем и над Half-Life 2.

Valve сосредоточилась на разработке SDK, который вышел в апреле 1999 года. Мин Ли, совместно с разработчиком, скрывающимся под ником «Cliffea», использует его для создания модификации для Half-Life под названием Counter-Strike (CS), многопользовательского шутера, основанного на идее противостояния группировки террористов и специального подразделения полиции. Видеоигра стала одной из ведущих киберспортивных дисциплин.

В конце 1999 года Valve начала разработку Half-Life 2. Для новой видеоигры компания разработала собственный движок Source Engine. Разработка держалась в строжайшем секрете, и впервые видеоигру показали лишь в 2003 году на E3. Видеоигра произвела не меньшее, чем первая часть, впечатление на посетителей выставки.

Новый движок стал отличным подспорьем для мододелов, и Valve предоставила сообществу SDK, документацию и набор утилит. Релиз Half-Life 2 был намечен на сентябрь 2003 года, однако, неожиданно произошла организованная хакером утечка файлов, и видеоигра была отложена на неопределённый срок.

Видео игру пришлось делать снова. Весь год команда трудилась над видеоигрой, существенно улучшив физику и графику. Half-Life 2 вышел 16 ноября 2004 года, спустя год после кражи. Видеоигра собрала более 35 наград "Видеоигра года". Официальная версия видеоигры требовала регистрации Steam, только что вышедшей системе цифровой дистрибуции от Valve [3].

В России в целом хорошее образование, большое количество талантливых и хорошо образованных людей. Конкретно в видеоиграх есть одна большая проблема – этому не учат в вузах. Так что игровая индустрия буквально сама зародилась много лет назад и сегодня саморазвивается.

Являясь лидером на рынке в России, мы делаем все, чтобы молодые команды приходили к нам и рассказывали про свои идеи, а мы им помогали. Общий цикл выглядит примерно так: найти очень хороший и интересный молодой коллектив с хорошей идеей, помочь им деньгами и опытом, помочь им с оперированием и маркетингом и в итоге добавить к себе новую студию, которая будет в чем-то уникальна [4].

В наше время сегмент российской индустрии видеоигр в жанре шутер хорошо развит, но имеет множество одинаковых игр в плане механик, меняя только визуальную составляющую. Это свидетельствует о том, что разработка, связанная с нововведениями в данном жанре высоко востребована. Исходя из отзывов платформы Steam выяснилось, что эта проблема является существенной и актуальной по сей день.

Обоснование необходимости разработки информационной системы:

1. Рынок ожидает нововведений;
2. Заинтересованность российских игроков в видеоиграх жанра шутер;
3. Разработка подобных видеоигр не занимает много времени и ресурсов;
4. Завышение цен на видеоигры для российских пользователей:
5. Крупные корпорации уходят с рынка видеоигр России.

Вывод: в данной главе была разобрана предметная область. Был проведён опрос, в котором выяснилось, что разработка видеоигры на российском рынке в жанре шутер очень перспективна.

1.1 Организационная структура объекта автоматизации

Объект исследования: демоверсия двумерной динамической экшн игры в жанре платформера “Equilibrium”.

Структура видеоигры включает следующие разделы:

* главное меню:

Дает пользователю возможность взаимодействия с началом игры. В главном меню размещены:

* новая игра;
* загрузить игру;
* продолжить игру;
* настройки;
* выход из игры.
* Настройки:

Дает пользователю возможность настроить игру для своего удобства, по собственному желанию. В режиме настроек размещены:

* настройки графики;
* настройки аудио;
* управление.
* пауза:

Дает пользователю возможность приостановить игру. В режиме паузы размещены:

* Продолжить игру;
* Настройки;
* Выход в главное меню.

1.2 Движение потоков данных

Модель считывания потоков данных представлена на рисунке 1.1.

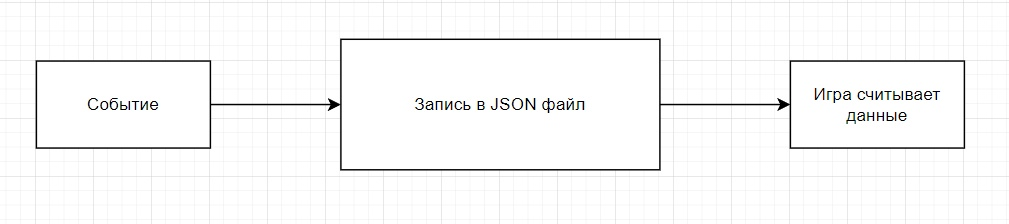


Рисунок 1.1. Модель движения потока данных

Сохранения реализованы в виде записи в JSON файл. Сохранение происходит автоматически при выходе из игры или по нажатию на определенную кнопку. Формируется или перезаписывается JSON файл с информацией о данных игры. В JSON файл записывается одна строка с номером уровня, на котором игрок сделал сохранение. При следующем заходе в игру игра считывает JSON файл и загружает необходимый уровень.

1.3. Нормативно-справочная информация

Видеоигра была разработана, с использованием следующих, правовых механизмов регулирования оборота компьютерных видеоигр в Российской Федерации:

1. с 1 сентября 2012 года, после вступления в силу федерального закона «О защите детей от информации, причиняющей вред их здоровью и развитию» в России введена возрастная классификация информационной продукции.

Информация, запрещенная для распространения среди детей (18+)

* 1. побуждающая детей к совершению действий, представляющих угрозу их жизни и (или) здоровью, в том числе к причинению вреда своему здоровью, самоубийству;
  2. способная вызвать у детей желание употребить наркотические средства, психотропные и (или) одурманивающие вещества, табачные изделия, алкогольную и спиртосодержащую продукцию, пиво и напитки, изготавливаемые на его основе, принять участие в азартных играх, заниматься проституцией, бродяжничеством или попрошайничеством;
  3. отрицающая семейные ценности, пропагандирующая нетрадиционные сексуальные отношения и формирующая неуважение к родителям и (или) другим членам семьи;
  4. содержащая информацию порнографического характера;
  5. содержащая нецензурную брань и другие ругательные слова;
  6. информация, оправдывающая преступления, жестокость и другие противоправные действия, и изображение последствия кровавого насилия, не исключающего сексуального насилия;
  7. о несовершеннолетнем, пострадавшем в результате противоправных действий (бездействия);

1.4 Разработка модели процесса

**Диаграмма классов**

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого илиобратного проектирования. Управление искусственным интеллектом происходит при помощи машины конечных состояний на рисунке 1.2 изображена абстракция модели управления искусственным интеллектом.

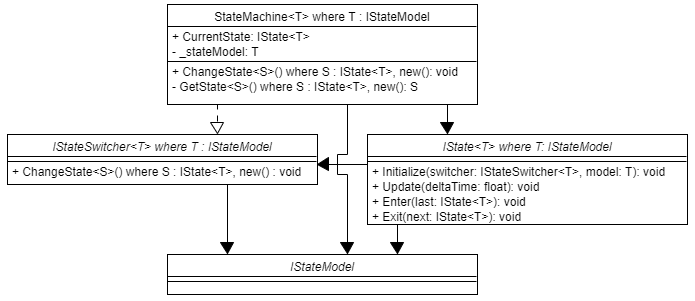


Рисунок 1.2. Абстрактная модель управления искусственным интеллектом (таблица 1-4)

Таблица – 1 класс Машина состояний (StateMachine<T> where T : IStateMode)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + CurrentState: IState<T>  - \_stateModel: T |
| Операции | +ChangeState<S>() where S : IState<T>, new(): void  -GetState<S>() where S : IState<T>, new(): S |

Таблица 2 – контракт Переключатель состояний (IStateSwitcher<T> where T : IStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | +ChangeState<S>() where S : IState<T>, new() : void |

Таблица 3 – контракт Состояния (IState<T> where T: IStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: T): void  + Update(deltaTime: float): void  + Enter(last: IState<T>): void  + Exit(next: IState<T>): void |

Таблица 4 – контракт Модель состояния (IStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции |  |

Управление искусственным интеллектом происходит при помощи машины конечных состояний на рисунке 1.3 изображена имплементация модели управления искусственным интеллектом.

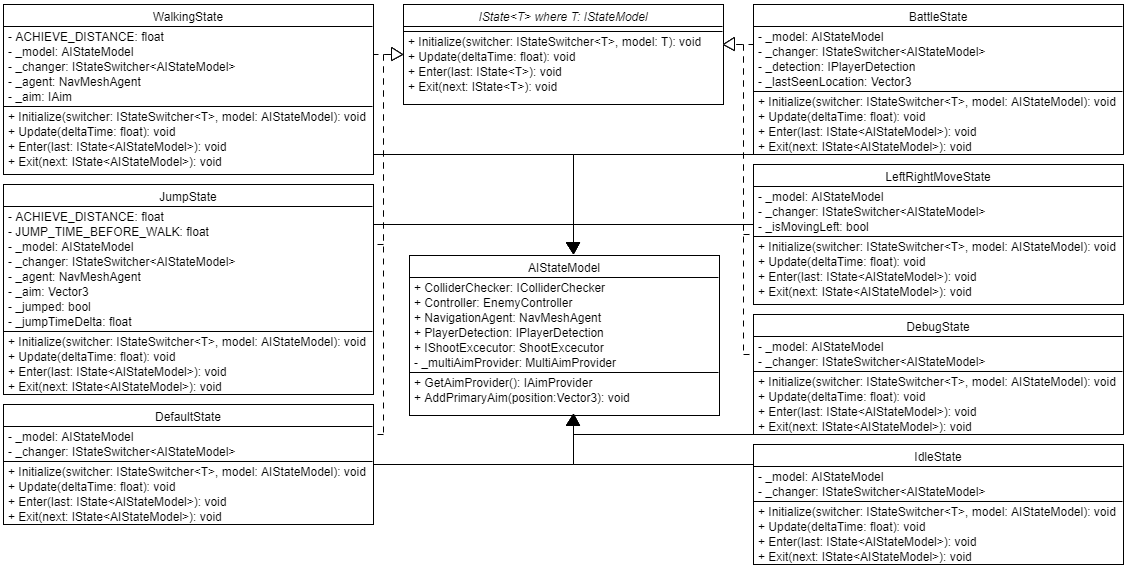


Рисунок 1.3. Имплементационная модель модель управления искусственным интеллектом (таблица 5-13)

Таблица – 5 класс Состояния движения (WalkingState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - ACHIEVE\_DISTANCE: float  - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel>  - \_agent: NavMeshAgent  - \_aim: IAim |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  +Enter(last:IState<AIStateModel>): void  +Exit(next:IState<AIStateModel>): void |

Таблица 6 – класс Состояния прыжков (JumpState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - ACHIEVE\_DISTANCE: float  - JUMP\_TIME\_BEFORE\_WALK: float  - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel>  - \_agent: NavMeshAgent  - \_aim: Vector3  - \_jumped: bool  - \_jumpTimeDelta: float |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  +Enter(last:IState<AIStateModel>): void  +Exit(next:IState<AIStateModel>): void |

Таблица 7 – класс Состояние бездействия (DefaultState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel> |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  + Enter(last: IState<AIStateModel>): void  + Exit(next: IState<AIStateModel>): void |

Таблица 8 – контракт Состояний (IState<T> where T: IStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: T): void  + Update(deltaTime: float): void  + Enter(last: IState<T>): void  + Exit(next: IState<T>): void |

Таблица 9 – класс Состояния AI модели (AIStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + ColliderChecker: IColliderChecker  + Controller: EnemyController  + NavigationAgent: NavMeshAgent  + PlayerDetection: IPlayerDetection  + IShootExcecutor: ShootExcecutor  - \_multiAimProvider: MultiAimProvider |
| Операции | + GetAimProvider(): IAimProvider  +AddPrimaryAim(position:Vector3): void |

Таблица 10 – класс Cостояние отладки (DebugState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel> |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  +Enter(last: IState<AIStateModel>): void  +Exit(next: IState<AIStateModel>): void |

Таблица 11 – класс Состояния бесцельного передвижения (LeftRightMoveState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel>  - \_isMovingLeft: bool |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  +Enter(last: IState<AIStateModel>): void  +Exit(next: IState<AIStateModel>): void |

Таблица 12 – класс Боевое состояние (BattleState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel>  - \_detection: IPlayerDetection  - \_lastSeenLocation: Vector3 |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  +Enter(last: IState<AIStateModel>): void  +Exit(next: IState<AIStateModel>): void |

Таблица 13 – класс Состояние ожидания (IdleState)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_model: AIStateModel  -\_changer: IStateSwitcher<AIStateModel> |
| Операции | +Initialize(switcher: IStateSwitcher<T>, model: AIStateModel): void  + Update(deltaTime: float): void  +Enter(last: IState<AIStateModel>): void  +Exit(next: IState<AIStateModel>): void |

Машина состояний — это математическая модель вычислений. Это — абстрактная концепция, в соответствии с которой машина может иметь различные состояния, но, в некий момент времени, пребывать лишь в одном из них. Реализованная машина состояний и её состояния для ИИ изображена на рисунке 1.4.

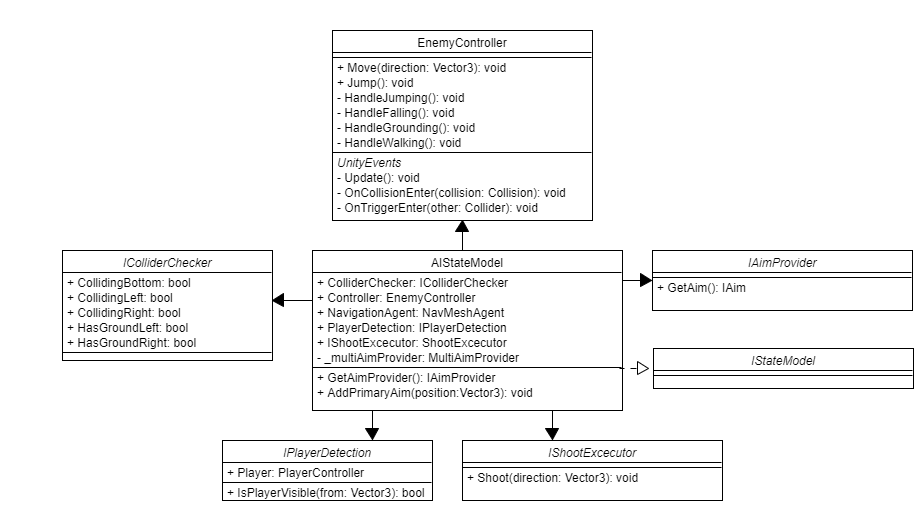


Рисунок 1.4. Модель машины состояний и её состояния для ии (таблица 14-20)

Таблица – 14 класс Вражеский контроллер (EnemyController)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + Move(direction: Vector3): void  + Jump(): void  - HandleJumping(): void  - HandleFalling(): void  - HandleGrounding(): void  - HandleWalking(): void |
| События | - Update(): void  - OnCollisionEnter(collision: Collision): void  - OnTriggerEnter(other: Collider): void |

Таблица 15 – класс Состояния AI модели (AIStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + ColliderChecker: IColliderChecker  + Controller: EnemyController  + NavigationAgent: NavMeshAgent  + PlayerDetection: IPlayerDetection  + IShootExcecutor: ShootExcecutor  - \_multiAimProvider: MultiAimProvider |
| Операции | + GetAimProvider(): IAimProvider  +AddPrimaryAim(position:Vector3): void |

Таблица 16 – контракт Cредство проверки коллайдера (IColliderChecker)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + CollidingBottom: bool  + CollidingLeft: bool  + CollidingRight: bool  + HasGroundLeft: bool  + HasGroundRight: bool |
| Операции |  |

Таблица 17 – контракт Обнаружение игрока (IPlayerDetection)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + Player: PlayerController |
| Операции | + IsPlayerVisible(from: Vector3): bool |

Таблица 18 – контракт Стрельба в исполнителя (IShootExcecutor)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + Shoot(direction: Vector3): void |

Таблица 19 – контракт Модель состояния (IStateModel)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции |  |

Таблица 20 – контракт Поставщик целей (IAimProvider)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + GetAim(): IAim |

Для проверки соприкосновений с какими-либо объектами, создана абстракция, и имплементация зависимости модели управления ИИ для движка Unity изображённая на рисунке 1.5.

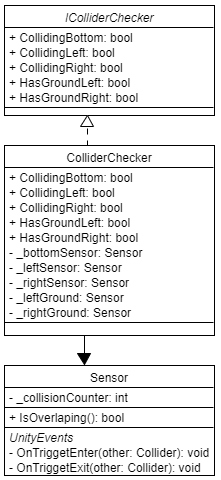


Рисунок 1.5. Модель для управления ии (таблица 21-23)

Таблица – 21 контракт Cредство проверки коллайдера (IColliderChecker)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + CollidingBottom: bool  + CollidingLeft: bool  + CollidingRight: bool  + HasGroundLeft: bool  + HasGroundRight: bool |
| Операции |  |

Таблица 22 – класс Cредство проверки коллайдера (ColliderChecker)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + CollidingBottom: bool  + CollidingLeft: bool  + CollidingRight: bool  + HasGroundLeft: bool  + HasGroundRight: bool  - \_bottomSensor: Sensor  - \_leftSensor: Sensor  - \_rightSensor: Sensor  - \_leftGround: Sensor  - \_rightGround: Sensor |
| Операции |  |

Таблица 23 – класс Датчик (Sensor)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_collisionCounter: int |
| Операции | + IsOverlaping(): bool |
| Событие | - OnTriggetEnter(other: Collider): void  - OnTriggetExit(other: Collider): void |

Для определения зоны видимости и достижимости главного игрока создан контракт, реализованные для движка Unity изображённый на рисунке 1.6.

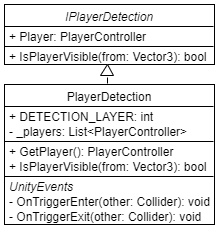


Рисунок 1.6. Модель видимости и достижимости главного игрока (таблица 24-25)

Таблица – 24 контракт контракт Обнаружение игрока (IPlayerDetection)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + Player: PlayerController |
| Операции | + IsPlayerVisible(from: Vector3): bool |

Таблица 25 – класс Обнаружение игрока (PlayerDetection)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + DETECTION\_LAYER: int  - \_players: List<PlayerController> |
| Операции | + GetPlayer(): PlayerController  + IsPlayerVisible(from: Vector3): bool |
| Событие | - OnTriggerEnter(other: Collider): void  - OnTriggerExit(other: Collider): void |

Для проведения боевых действий создан класс для работы с оружием, его может использовать как игрок, так и ИИ изображённый на рисунке 1.7.

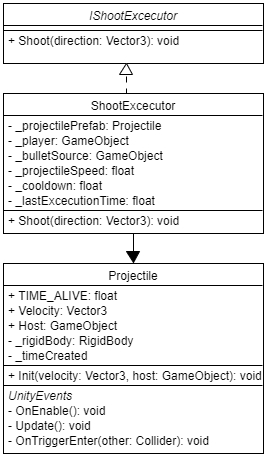


Рисунок 1.7. Модель проведения боевых действий (таблица 26-28)

Таблица – 26 контракт Стрельба в исполнителя (IShootExcecutor)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + Shoot(direction: Vector3): void |

Таблица 27 – класс Стрельба в исполнителя (ShootExcecutor)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_projectilePrefab: Projectile  - \_player: GameObject  - \_bulletSource: GameObject  - \_projectileSpeed: float  - \_cooldown: float  - \_lastExcecutionTime: float |
| Операции | + Shoot(direction: Vector3): void |

Таблица 28 – класс Снаряд (Projectile)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + TIME\_ALIVE: float  + Velocity: Vector3  + Host: GameObject  - \_rigidBody: RigidBody  - \_timeCreated |
| Операции | + Init(velocity: Vector3, host: GameObject): void |
| Событие | - OnEnable(): void  - Update(): void  - OnTriggerEnter(other: Collider): void |

Для определения текущих задач для ИИ создан контракт цели, и несколько имплементаций, для разного поведения изображённый на рисунке 1.8.

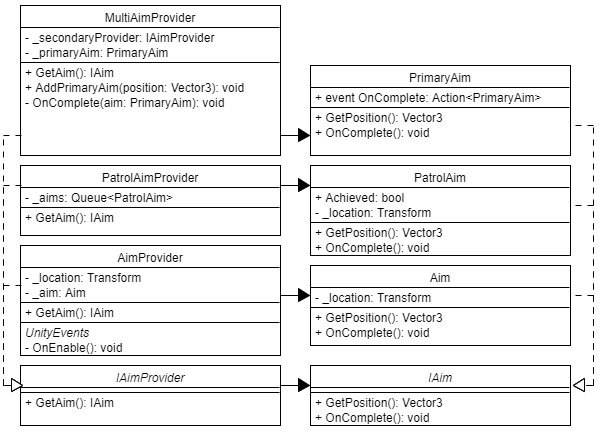


Рисунок 1.8. Модель текущих задач для ИИ (таблица 29-36)

Таблица – 29 класс Многоцелевой провайдер (MultiAimProvider)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_secondaryProvider: IAimProvider  - \_primaryAim: PrimaryAim |
| Операции | + GetAim(): IAim  + AddPrimaryAim(position: Vector3): void  - OnComplete(aim: PrimaryAim): void |

Таблица 30 – класс Провайдер цели патруля (PatrolAimProvider)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_aims: Queue<PatrolAim> |
| Операции | + GetAim(): IAim |

Таблица 31 – класс Поставщик целей (AimProvider)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_location: Transform  - \_aim: Aim |
| Операции | + GetAim(): IAim |
| Событие | - OnEnable(): void |

Таблица 32 – контракт Поставщик целей (IAimProvider)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + GetAim(): IAim |

Таблица 33 – класс Основная цель (PrimaryAim)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | +event OnComplete: Action<PrimaryAim> |
| Операции | + GetPosition(): Vector3  + OnComplete(): void |

Таблица 34 – класс Патрульная цель (PatrolAim)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + Achieved: bool  - \_location: Transform |
| Операции | + GetPosition(): Vector3  + OnComplete(): void |

Таблица 35 – класс Цель (Aim)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_location: Transform |
| Операции | + GetPosition(): Vector3  + OnComplete(): void |

Таблица 36 – контракт Цель (IAim)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + GetPosition(): Vector3  + OnComplete(): void |

Для управления персонажем созданы классы для чтения входных данных с мышки и клавиатуры, изображённые на рисунке 1.9.

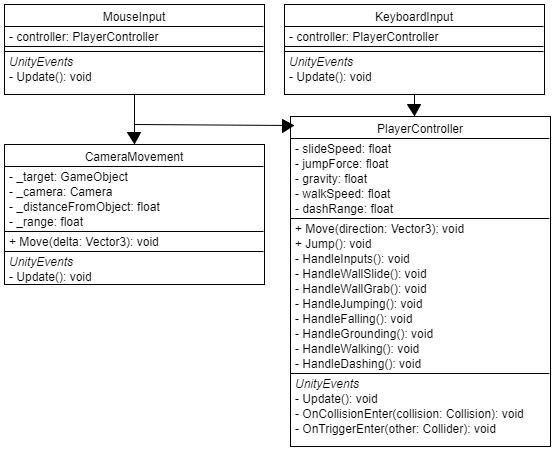


Рисунок 1.9. Модель управления персонажем (таблица 37-40)

Таблица – 37 класс Ввод с помощью мыши (MouseInput)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - controller: PlayerController |
| Операции |  |
| События | - Update(): void |

Таблица 38 – класс Ввод с клавиатуры (KeyboardInput)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - controller: PlayerController |
| Операции |  |
| События | - Update(): void |

Таблица 39 – класс Движение камеры (CameraMovement)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - \_target: GameObject  - \_camera: Camera  - \_distanceFromObject: float  - \_range: float |
| Операции | + Move(delta: Vector3): void |
| События | - Update(): void |

Таблица 40 – контракт Контроллер игрока (PlayerController)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - slideSpeed: float  - jumpForce: float  - gravity: float  - walkSpeed: float  - dashRange: float |
| Операции | + Move(direction: Vector3): void  + Jump(): void  - HandleInputs(): void  - HandleWallSlide(): void  - HandleWallGrab(): void  - HandleJumping(): void  - HandleFalling(): void  - HandleGrounding(): void  - HandleWalking(): void  - HandleDashing(): void |
| События | - Update(): void  - OnCollisionEnter(collision: Collision): void  - OnTriggerEnter(other: Collider): void |

На уровне находятся объекты, наносящие урон, а также урон наносится оружием противника, и главного игрока. Выделены компоненты для здоровья, и опасных объектов изображённые на рисунке 1.10.

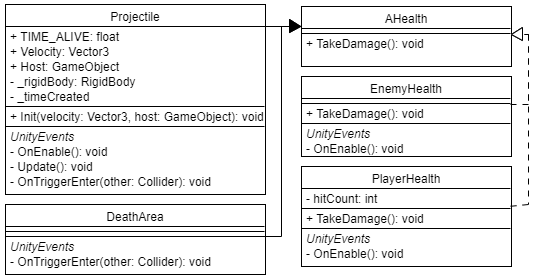


Рисунок 1.10. Модель объектов наносящих урон (таблица 41-45)

Таблица – 41 класс Снаряд (Projectile)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | + TIME\_ALIVE: float  + Velocity: Vector3  + Host: GameObject  - \_rigidBody: RigidBody  - \_timeCreated |
| Операции | +Init(velocity: Vector3, host: GameObject): void |
| События | - OnEnable(): void  - Update(): void  - OnTriggerEnter(other: Collider): void |

Таблица 42 – класс Зона смерти (DeathArea)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции |  |
| События | - OnTriggerEnter(other: Collider): void |

Таблица 43 –класс Здоровье (AHealth)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + TakeDamage(): void |

Таблица 44 – класс Вражеское здоровье (EnemyHealth)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты |  |
| Операции | + TakeDamage(): void |
| События | - OnEnable(): void |

Таблица 45 – класс Здоровье игрока (PlayerHealth)

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Комментарий |  |
| Атрибуты | - hitCount: int |
| Операции | + TakeDamage(): void |
| События | - OnEnable(): void |

**Диаграмма состояний**

Диаграмма состояний представляет собой состояния, соединенные переходами. Переходы могут сработать в результате заданного в качестве условия перехода события – это может быть истечение заданного таймаута, получение диаграммой состояний сообщения, выполнение заданного логического условия и т.д. Срабатывание перехода приводит к переходу управления диаграммы состояний в то состояние, в которое ведет этот переход. Состояния могут быть иерархическими, т.е. содержать другие состояния и переходы, на рисунке 1.11. изображена диаграмма состояний.

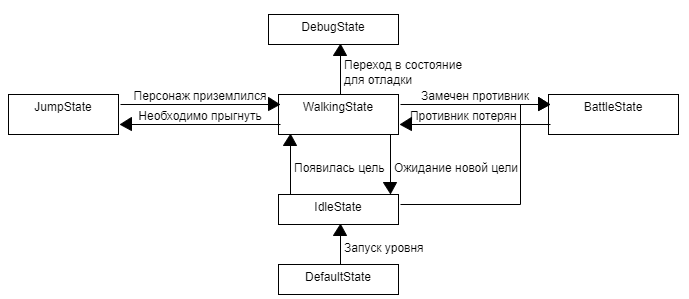


Рисунок 1.11. Модель перехода состояний

Таблица 46 – Описание диаграммы состояний.

|  |  |
| --- | --- |
| **Состояние** | **Описание состояния** |
| Состояние прыжка(JumpState) |  |
| Состояние ходьбы(WalkingState) |  |
| Cостояние отладки(DebugState) |  |
| Боевое состояние(BattleState) |  |
| Состояние ожидания(IdleState) |  |
| Состояние бездействия(DefaultState) |  |

**Диаграмма последовательности**

Диаграмма последовательности — [UML-диаграмма](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_(UML)), на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках [прецедента](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82_(UML)).

Основными элементами диаграммы последовательности являются обозначения [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (прямоугольники с названиями объектов), вертикальные «линии жизни» отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами изображена на рисунке 1.12.

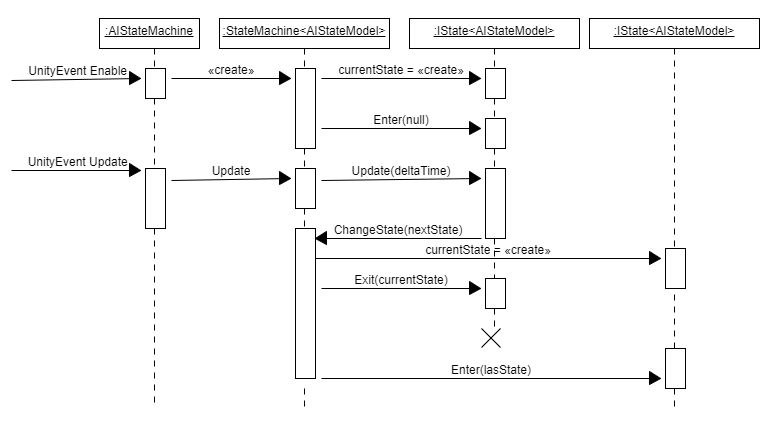


Рисунок 1.12. Модель последовательности машины состояний

1.5 Обзор существующих аналогов

Проект реализуется, довольно быстро и проще по сравнению с аналогами. В совокупности наши механики уникальны. Реализована псевдотрехмерность. Быстрый экшн, также является ключевым отличием. Более структурированное сравнение с аналогами, представлено в таблице 47.

Таблица 47 – Сравнение с конкурентами

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Bleed** | **Intrusion 2** | **Наш продукт** |
| Быстрый экшн | + | - | + |
| Уникальность механик | + | + | + |
| Быстрая реализация продукта | - | - | + |
| Псевдотрехмерность | - | - | + |

1.5.1 Размер и количество фирм, работающих в отрасли

В видеоигровой индустрии существует огромное множество компаний, которые занимаются разработкой и внедрением видеоигр. Так, например, на территории стран СНГ насчитывается около 250 официально зарегистрированных разработчиков игр, если говорить об отдельных незарегистрированных командах и людях, занимающихся разработкой игр в одиночку, то это количество будет намного больше. Из-за огромного количества разработчиков видеоигр, компаниям сложно влиять на цену продукта, поэтому компании в данной отрасли должны превзойти соперника по качеству выпускаемого продукта.

1.5.2 Анализ покупателей

Подавляющую часть покупателей составляют физические лица, которые приобретают видеоигры или товары-комплементы для личного потребления. Физические лица, при скоординированности мнения, имеют возможность влиять на разработчиков и издателей игр. Во-первых, влияние на рыночного партнёра в своей основе имеет принципиальную возможность отказа иметь с ним дело. Во-вторых, насколько стороны могут укрепить своё положение за счёт опыта и искусности в ведении игры. Домашние хозяйства (геймеры) имеют 2 основные характеристики:

* Приверженность к качеству товара. Покупатель выбирает, при прочих благоприятных обстоятельствах, наиболее качественный продукт.
* Организованность мнения. Компании производители вынуждены учитывать мнение геймерского сообщества, так как, игнорирование мнения или заявлений приводит к организованной ответной реакции.

1.5.3 Подведение итогов.

Современная игровая индустрия – это сложный симбиоз высокотехнологичных продуктов, неценовой конкуренции компаний и множества рычагов давления. При успешной маркетинговой стратегии, разрабатывая качественный продукт и учитывая мнения геймерского сообщества, шансы на успех игры будут высоки. Хотя количество конкурентов достигает больших отметок, но стоит учитывать, что и количество покупателей огромное множество, о чем свидетельствуют данные оборота рынка видеоигр.

1.6. Область применения

Видеоигра будет применяться русскоязычными игроками на персональных компьютерах. Видеоигра, позволит игрокам провести время с целью развлечения. Распространяется данная видеоигра будет на таких крупных игровых торговых площадках, как:

* Steam;
* Epic Games.

1.7. Классификация компьютерных игр

Компьютерные игры можно классифицировать по различным критериям:

* жанр: геймплей задает жанровую направленность, возможно несколько разновидностей;
* режим игры: одиночный или многопользовательский;
* визуальное представление: игра может, как использовать графические средства оформления, так и напротив, быть текстовой. Игра также может быть двухмерной или трехмерной. Есть и звуковые игры — в них вместо визуального представления используются звуки;
* платформозависимость: на каких платформах возможет запуск игры, так же игры бывают платформонезависимыми (кроссплатформенными).

Классификация по жанрам:

* приключенческая игра или квест (Adventure) — игра с продуманным и обычно линейным сюжетом. В этом направлении широко используют различные головоломки;
* боевик (Action) — игра, требующая от игрока постоянных действий, насыщенная боевыми сценами, драками и перестрелками;
* ролевая игра (RPG — англ. Role Playing Game) — игра, отличительной особенностью которой является наличие у персонажей определённых навыков и характеристик, которые можно обрести, а впоследствии развивать, выполняя какие-либо действия. К этому жанру относятся и многопользовательские ролевые игры (ММОРПГ), которые, в отличие от однопользовательских, не имеют ни конечной цели, ни законченного сюжета. Примеры: Gothic (серия);
* стратегическая игра (Strategy) — игра, представляющая собой управление глобальными процессами, как например, развитие экономики, создание армии, строительство парков и т. д. В данном жанре обычно используется два режима игры: в реальное время или пошаговый режим;
* компьютерный симулятор (Simulator) — игра, созданная с целью симуляции реальных действий в жизни, например, управление автомобилем или симулятор танкиста.

1.8. Игровая терминология

Вся игровая терминология в русском языке была заимствована из английского. Очень сложно найти хорошую аналогию английским терминам. Но иногда возможно применять русскоязычные названия.

Texture – текстура. Это обычно двухмерное изображение, которое можно использовать как для наложения её на трёхмерные модели, так и отрисовки на экране как картинку.

2D-game – двумерная игра – игра, в которой используется только двухмерное пространство.

3D-game – трёхмерная игра – использующая трёхмерные модели и трехмерный игровой мир.

Tile – тайл – небольшое изображение, которое используется для конструирования уровней в играх так же существует такое понятие как AutoTile – это механизм создания бесшовных карт.

Polygone (полигон, многоугольник) – пространственный многоугольник, который используется для создания трехмерных объектов. Как правило, в компьютерной графике используются треугольники.

Pixel (пиксель) – наименьший элемент растрового изображения, точка, отображаемая на экране. Обычно в пикселях измеряют разрешение текстур (например – 640х640), экранное разрешение монитора, размеры игровых окон. Слово Pixel – это аббревиатура от Picture's Element.

Texture Filtering (фильтрация текстур) – уменьшение искажений при наложении текстур на трехмерный объект.

Camera (камера) – Это проекция на игровой мир, которая имеет ширину, высоту и угол поворота, также она имеет координаты в пространстве, будь то трёхмерное, будь то двухмерное. Камера играть важную роль в представлении игры.

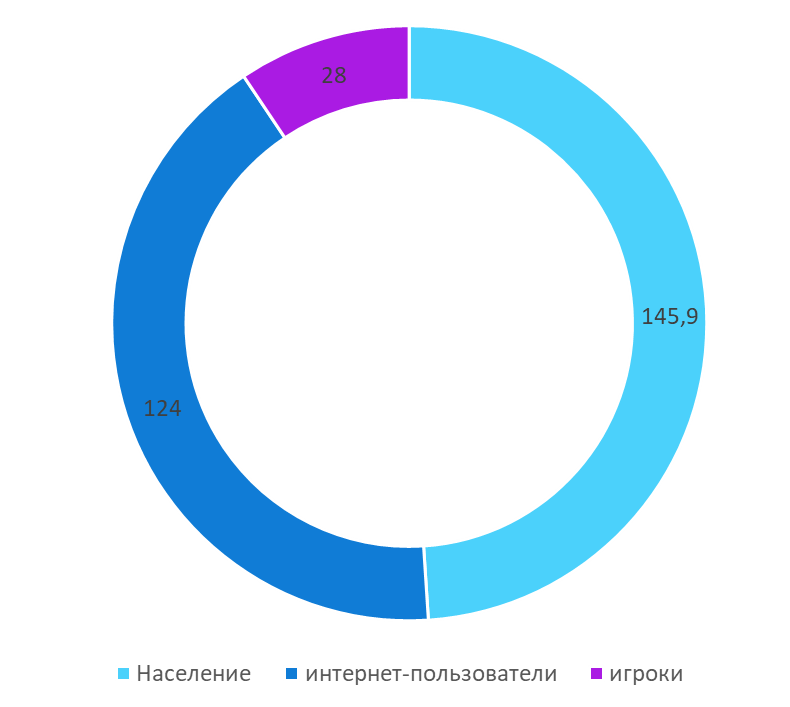
Light Model (модель освещения) – способы моделирования освещения объектов.

2. Постановка задачи

2.1 Актуальность, цель и назначение

На данный момент в связи с ситуацией в мире многие разработчики игр перестают выпускать игры для российских пользователей или завышают цены на свои продукты. Мы же разрабатываем игру ориентированную на российский рынок где у пользователей не возникнет вышеперечисленных проблем.

На сегодняшний день игровая индустрия развивается достаточно быстро, сейчас в игры стали играть даже тот сегмент пользователей, который раньше не был в них как-либо заинтересован. Российский интернет-рынок показал себя с лучшей стороны, ведь даже в кризис он не переставал расти. “На 2021 год, количество интернет-пользователей в России выросло на 6 миллионов и достигло отметки в 124 млн интернет-пользователей”. Об этом заявил глава Роскомнадзора Андрей Липов в октябре 2021 года. Это составляет 85% населения Российской Федерации. По данным ВЦИОМа, 19% россиян активно играют в видеоигры, что составляет 28 млн человек. Диаграмма возможностей рынка представлена на рисунке 2.1.

****Рисунок 2.1. Возможности рынка.

Объем российского рынка видеоигр в 2021 году достиг 158 млрд рублей, увеличившись на 7,7% в сравнении с 2020-м. Такие данные были обнародованы аналитиками ПрайсвотерхаусКуперс в середине января 2022 года. (PwC предоставляет аудиторские и консультационные услуги, а также услуги в области налогообложения и юридические услуги публичным и частным компаниям разных отраслей).

По прогнозу, рынок компьютерных игр по-прежнему остается самым крупным сегментом в стране. Несмотря на то, что за год он сократился на 4,6%, его объем в 2021 составляет 78,1 млрд рублей.

На втором месте будет сегмент мобильных игр. В 2021 году он может вырасти на 10% и составить 73,8 млрд рублей. Диаграмма оборота рынка в России представлена на рисунке 2.2.

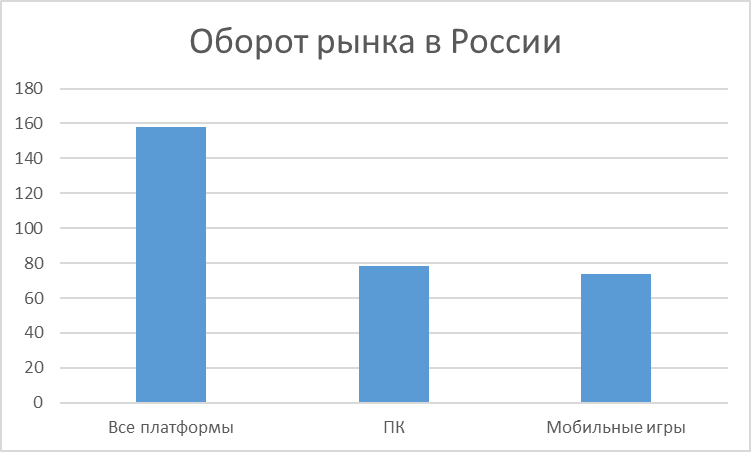


Рисунок 2.2. Оборот рынка в России

С уверенностью можно говорить о том, что игра разработанная нашей командой будет востребована и хорошо продаваться, ведь этому способствует выбранный жанр. Игры в жанре платформер, были популярны с начала времен создание видеоигр, популярность сохранилась и по наши дни. Популярность обуславливается, следующими факторами:

1. интересный игровой процесс;
2. низкий порог вхождения;
3. простые в освоении механики.

На основе вышеописанного выделяются цель и назначение проекта.

Целью проекта является разработка и предоставление видеоигры, для приятного времяпровождения пользователей.

Назначением проекта является развлечение пользователей.

2.2. Функционал проекта

Прежде всего, проект должен отвечать следующим требованиям:

1. игра должна основываться на простоте и интуитивно понятном геймплее;
2. основная механика видеоигры заключается в возможностях игрока манипулировать гравитацией (простое перемещение по полу, перемещение по стенам и потолку);
3. враги патрулируют определенную область, либо обитают в заданных местах. При обнаружении игрока, враги следуют за игроком с целью его нейтрализации;
4. на уровне должны присутствовать ловушки. Эти объекты должны добавить дополнительные сложности прохождения игроку;
5. при нейтрализации игрока, должно произвестись мгновенное перемещение в начало локации с последующим откатом уровня к начальному состоянию;
6. меню паузы не должно быть громоздким, должно предоставлять основную информацию.

Состав документации для всех этапов жизненного цикла проекта представлен в таблице 1.

Таблица 1 – состав документации.

|  |  |
| --- | --- |
| **Этап** | **Документ** |
| Проектирование. Разработка технического задания. Разработка спецификаций. | Ведомость технического проекта |
| Пояснительная записка к техническому проекту |
| Описание автоматизируемых функций |
| Описание комплекса технических средств |
| Разработка. Дизайн. Кодирование. Тестирование. Документирование. | Ведомость рабочей и эксплуатационной документации |
| Формуляр |
| Общее описание системы |
| Руководство пользователя |
| Технологическая инструкция |
| Программа и методика испытаний |
| Ввод в эксплуатацию. Техподдержка. Внедрение. Сопровождение. | Протокол развертывания системы |
| Протокол предварительных испытаний |
| Акт приемки в опытную эксплуатацию |
| Журнал опытной эксплуатации |
| Акт о завершении опытной эксплуатации |
| Протокол приемочных испытаний |
| Акт приемки системы в постоянную эксплуатацию |
| Формуляр |

2.3. Сценарий

В нашей видеоигре события разворачиваются в мире будущего, где технологии продвинулись далеко вперёд. Локации будут представлены в виде двумерного платформера, с переходами по уровням. Обитателями данного мира будут являться обычные люди. По задумке игры, из-за тирании правительства, обычное население в мире будущего не процветает, а бедствует. В связи чем у населения падает прирост настроения, что приводит к низкой мотивации к труду, спаду продолжительности жизни и общего уровню образования, что естественным образом доводит до бунтов и восстаний. Появляются повстанческие организации.

Из-за чего же случился весь конфликт? В данном мире правительство решило отказаться от эмоций и принуждает людей к этому (люди остались без выбора, им остается только подчиниться). Наш главный герой вначале игры работал на правительство тем самым поддерживая порядок среди мирных граждан, а потом осознав весь абсурд происходящего присоединится к повстанцем, дабы разрушить эту систему.

О нашем персонаже. Это мужчина, средних лет, брюнет, рост выше среднего, отличительных особенностей нет. Особыми навыками не наделен. Из истории персонажа: наш главный герой работал на правительство, после того как его жену лишили эмоций, ему стало не по себе. Он понял, насколько бесчеловечны действия правительства и решил присоединиться к повстанцам.

2.4. Калькуляция проекта

Для реализации проекта были использованы множество различных элементов, начиная от статичных картинок и заканчивая несколькими скриптами для реализации механики игры. Во время разработки игры было создано 2 сцены: Главное меню и Игровой уровень. В главном меню присутствуют лишь фоновое изображение и 2 кнопки, отвечающие за запуск Игрового уровня и выход из игры. Общее количество объектов, которые были использованы или созданы во время реализации проекта, представлены в таблице 1.

Таблица 1 — Калькуляция элементов игры.

|  |  |
| --- | --- |
| **Элемент** | **Комментарий** |
| Фоновые изображения | n-ое количество растровых изображений |
| Главный герой | n анимаций, n спрайтов |
| Враг | n анимаций, n спрайтов |
| Ловушки | n анимаций, n спрайтов |
| Скрипты | n скриптов, с общим количеством строчек кода n |

В Игровом уровне присутствуют все основные элементы игры, и в общей сложности содержит более n объектов, как повторяющихся, так и нет. В том числе главный персонаж, n врагов, несколько ловушек.

2.4. Характеристики комплекса задач

Задачей данной выпускной квалификационной работы, является разработка демо-версии двумерной динамической экшн игры в стиле платформера “Equilibrium”, а также приобретение теоретических знаний и практических умений по разработке видеоигры с применением различных технологий и языков программирования C#.

Демоверсия – это предварительная версия, и как правило бесплатная, служащая для рекламы и привлечения внимания пользователей, создавая ажиотаж вокруг будущего продукта. Демоверсия содержит небольшой фрагмент будущей видеоигры - основные механики, 1-2 уровня, которые смогут дать пользователю полноценное представление о грядущей видеоигре. В нашем проекте в качестве демо версии реализовано:

1. один уровень;
2. основные механики;
3. графика;
4. система ловушек;
5. система врагов;
6. система AI.

Объектом исследования является процесс разработки видеоигры. Предметом исследования является технология создания видеоигры.

Для реализации цели выпускной квалификационной работы необходимо решить следующие задачи:

* изучить теоретические аспекты по созданию видеоигры;
* провести анализ существующих технологий, движков и языков программирования, используемых при создании видеоигры.

Видеоигра должна включать в себя несколько основных пунктов:

* главное меню — дает пользователю возможность взаимодействия с началом игры:
* настройки — дает пользователю возможность настроить игру для своего удобства, по собственному желанию;
* пауза — дает пользователю возможность приостановить игру.

3. Выбор средств реализации

3.1. Инструментальные средства

В качестве основной среды разработки для данного проекта используется игровой движок Unity. Unity - “межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идет постоянное развитие”.

Unity — это мультиплатформенный инструмент для разработки двух- и трёхмерных приложений и игр, работающий под операционными системами Windows и OS X. Игры, созданные на данном инструменте можно портировать на: Windows, OS X, Android, Apple iOS, Linux, а также на игровые приставки Wii, PlayStation 3 и XBox 360. Также можно создавать игры, работающие в браузере, для этого надо установить специальный модуль Unity Web Player. Также приложения созданные, с помощью Unity3D поддерживают обе спецификации 3D графики DirectX и OpenGL.

Особенности:

* несколько сценарных языков программирования: C#, JavaScript (модификация) и Boo;
* возможность мгновенного запуска игры;
* простая работа с ресурсами через Drag-and-Drop;
* широкие возможности импорта;
* полностью настраиваемый и доступный большинству людей интерфейс;
* кроссплатформенность;
* мощь, гибкость и бесконечная расширяемость;
* наличие бесплатной версии с некоторыми ограничениями.

Microsoft Visual Studio — линейка продуктов компании Майкрософт, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server). В данной работе Visual Studio 2012 используется как лучшая альтернатива MonoDevelop, которую использует Unity3D “из коробки”.

**Язык программирования C#**

C# (произносится си шарп) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Данный язык был выбран в качестве основного, так как обладает нужными качествами для реализации, имеет встроенную поддержку обобщений, делегатов и событий, что облегчит реализацию.

GitHub — крупнейший [веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для [хостинга](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3) [IT-проектов](https://ru.wikipedia.org/wiki/IT) и их совместной разработки.

Веб-сервис основан на системе контроля версий [Git](https://ru.wikipedia.org/wiki/Git) и разработан на [Ruby on Rails](https://ru.wikipedia.org/wiki/Ruby_on_Rails) и [Erlang](https://ru.wikipedia.org/wiki/Erlang) компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая [SSL](https://ru.wikipedia.org/wiki/SSL)), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Создатели сайта называют GitHub «[социальной сетью](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) для разработчиков».

Кроме размещения кода, участники могут общаться, комментировать правки друг друга, а также следить за новостями знакомых.

С помощью широких возможностей Git программисты могут объединять свои [репозитории](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9) — GitHub предлагает удобный интерфейс для этого и может отражать вклад каждого участника в виде [дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)).

Для проектов есть личные страницы, небольшие [Вики](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8) и [система отслеживания ошибок](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BE%D1%82%D1%81%D0%BB%D0%B5%D0%B6%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA).

Прямо на сайте можно просмотреть файлы проектов с [подсветкой синтаксиса](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%81%D0%B2%D0%B5%D1%82%D0%BA%D0%B0_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81%D0%B0) для большинства [языков программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

* Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям. Раньше такая возможность была платной;
* есть возможность прямого добавления новых файлов в свой репозиторий через веб-интерфейс сервиса;
* код проектов можно не только скопировать через [Git](https://ru.wikipedia.org/wiki/Git), но и скачать в виде обычных архивов с сайта;
* кроме [Git](https://ru.wikipedia.org/wiki/Git), сервис поддерживает получение и редактирование кода через [SVN](https://ru.wikipedia.org/wiki/SVN) и [Mercurial](https://ru.wikipedia.org/wiki/Mercurial);
* на сайте есть [pastebin](https://ru.wikipedia.org/wiki/Pastebin)-сервис [gist.github.com](https://gist.github.com/) для быстрой публикации фрагментов кода.

Trello — облачная программа для управления проектами небольших групп, разработанная Fog Creek Software.

Trello использует парадигму для управления проектами, известную как [канбан](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD_(%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0)), метод, который первоначально был популяризирован [Toyota](https://ru.wikipedia.org/wiki/Toyota) в 1980-х для управления цепочками поставок. Trello использует [freemium](https://ru.wikipedia.org/wiki/Freemium)-бизнес-модель, платные услуги были запущены в 2013 году. В 2017 году куплен [Atlassian](https://ru.wikipedia.org/wiki/Atlassian) за 425 миллионов долларов США[[6]](https://ru.wikipedia.org/wiki/Trello#cite_note-6).

особенности

Trello ограничил поддержку тегов в виде десяти цветных меток, которые можно переименовать. Карточки поддерживают комментарии, вложения, сроки выполнения и контрольные списки. Trello имеет [API](https://ru.wikipedia.org/wiki/API)[[7]](https://ru.wikipedia.org/wiki/Trello#cite_note-7). В настоящее время поддерживаются мобильные платформы приложений [iPhone](https://ru.wikipedia.org/wiki/IPhone) и [Android](https://ru.wikipedia.org/wiki/Android). Также был разработан веб-сайт, чтобы быть доступным в большинстве мобильных веб-браузеров. Приложение для [iPad](https://ru.wikipedia.org/wiki/IPad) было выпущено 12 марта 2013 года[[8]](https://ru.wikipedia.org/wiki/Trello#cite_note-8). У Trello есть собственная [дизайн-система](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), которая называется Nachos.

3.1. Обоснование выбора среды моделирования

Для моделирования системы использовались следующие среды – StarUML и Draw.io.

StarUML — это программный инструмент визуального моделирования с открытым исходным кодом, который поддерживает стандартизованный язык графического описания UML (Unified Modeling Language) для моделирования систем и программного обеспечения.

Программный продукт StarUML от разработчика MKLabs предназначен для создания и применения графических моделей в нотации UML. Система основана на UML 2.0 и предоставляет одиннадцать различных видов диаграмм, активно поддерживая таким образом подход построения архитектур на базе моделей (англ. Model Driven Architecture, MDA). Система может эффективно применяться системными аналитиками, проектировщиками и архитекторами систем, инженерами-программистами.

Программное обеспечение StarUML отличается высокой настраиваемостью в соответствии с пользовательской средой и высокой расширяемостью в своей функциональности. Использование StarUML обеспечивает высокую производительность и качество программных проектов [6].

Draw.io — инструмент для создания диаграмм, блок-схем, интеллект-карт, бизнес-макетов, отношений сущностей, программных блоков и другого. Сервис распространяется на бесплатной основе с открытым исходным кодом. Draw.io обладает богатым набором функций для визуализации большинства задач пользователя.

Инструмент работает с Google Диск, Google Workspace и Dropbox, глубоко интегрирован и удобен для работы с продуктами Confluence и Jira от Atlassian. Пользователи также могут работать с диаграммами в автономном режиме и сохранять их локально, используя настольное приложение для персональных компьютеров.

Инструмент позволяет создавать: графики, диаграммы, таблицы, презентации, блок-схемы, планы помещений, воронки продаж, ментальный карты, карты сайтов [7].

3.2 Функциональное моделирование ИС, стандарт IDEF0

Для описания процессов информационной системы использовалось функциональное моделирование в стандарте IDEF0.

Функциональное моделирование — это процесс моделирования функций, выполняемых рассматриваемой информационной системой/объектом, путем создания описательного структурированного графического изображения, показывающего, что, как и кем делается в рамках функционирования объекта и объектов, связывающих эти функции, с учетом имеющейся информации [8].

3.2.1 Контекстная диаграмма

Контекстная диаграмма — вид IDEF0-диаграммы. Это диаграмма, расположенная на вершине древовидной структуры диаграмм, представляющая собой самое общее описание системы и ее взаимодействие с внешней средой (как правило, здесь описывается основное назначение моделируемого объекта). Контекстная диаграмма состоит из одного блока, описывающего функцию верхнего уровня, ее входы, выходы, управления, и механизмы, вместе с формулировками цели модели и точки зрения, с которой строится модель.

# 

ГЛАВА 4. ПРОЕКТИРОВАНИЕ ПО

Программное обеспечение для торговой площадки “Steam” представляет собой приложение, на котором будут реализованы подсистемы регистрации, подключения на сервер.

При регистрации на сайте пользователь должен иметь возможность создать свой аккаунт и зайти на сервер, и в случае необходимости изменить его данные.

Требования к составу, структуре, и функциональным характеристикам программных средств.

В состав технических средств входит:

* компьютер оператора, включающий в себя:
* процессор x86 с тактовой частотой, не менее 2 ГГц;
* оперативную память объемом, не менее 2 Гб;
* видеокарту, монитор, мышь, клавиатура.

Компьютер пользователя, включающий в себя:

* процессор x86 с тактовой частотой, не менее 2 ГГц;
* оперативную память объемом, не менее 2 Гб;
* видеокарту, монитор, мышь.
* два компьютера для СУБД (основной и резервный), включающий в себя:
* процессор x86 с тактовой частотой, не менее 2 ГГц;
* оперативную память объемом, не менее 2 Гб;
* видеокарту, монитор, мышь.

4.1 Требования к надежности

Вероятность безотказной работы системы должна составлять не менее 99.99% при условии исправности компьютера и его составляющих.

4.1.1 Требования к обеспечению надежного (устойчивого) функционирования программы

В связи с тем, что в базе данных хранятся данные о совершенных клиентами покупках (финансовая информация) — базу данных стоит резервировать (резервирование замещением).

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.

4.1.2 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 10 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 средств.

4.2 Требования к составу и параметрам технических средств

Состав технических средств:

Компьютер оператора, включающий в себя:

* процессор x86 с тактовой частотой, не менее 2 ГГц;
* оперативную память объемом, не менее 2 Гб;
* видеокарту, монитор, мышь, клавиатура.

Компьютер посетителя, включающий в себя:

* процессор x86 с тактовой частотой, не менее 2 ГГц;
* оперативную память объемом, не менее 2 Гб;
* видеокарту, монитор, мышь.

Два компьютера для СУБД (основной и резервный), включающий в себя:

* процессор x86 с тактовой частотой, не менее 2 ГГц;
* оперативную память объемом, не менее 2 Гб;
* видеокарту, монитор, мышь.

5. Бизнес-модель Lean Canvas

Lean Canvas - представляет собой таблицу из 9 блоков, суть которой является сжатие большого плана бизнес-модели, на один лист, сохраняя суть и экономя время для презентаций. Lean Canvas, представлен на рисунке 5.1.

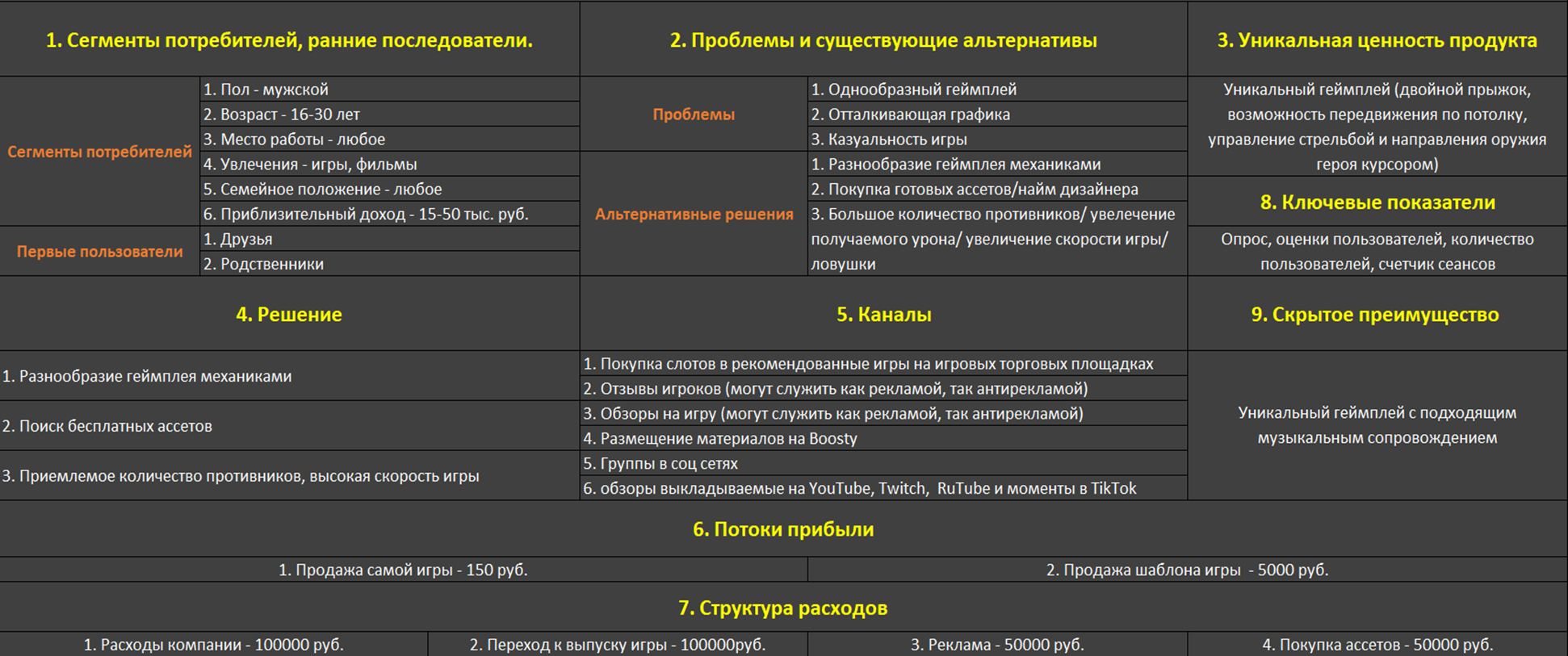


Рисунок 5.1. Бизнес-модель Lean Canvas

5.1. Сегменты потребителей, ранние последователи

Целевая аудитория:

1. Пол - мужской (Большинство людей играющих в ПК игры в России, это мужчины. И жанр нашей игры больше подходит для мужской аудитории, нежели для женской);
2. Возраст - 16–30 лет (Средний возраст ПК-игроков в России по данным из «Яндекс», 30 лет. К нашей игре больше подходит данная возрастная группа. Поскольку игра выйдет довольно динамичная и большинству людей младше и старше данной возрастной группы, будет менее интересно в неё играть);
3. Место работы, как и место учебы – любое (Место работы, как и место учебы не играет особой роли для игр и может быть любым. Единственное, что можно сказать, так это то, что людям с довольно малым количеством свободного времени (работают на нескольких работах, и т. д.), будет труднее играть в нашу игру)**;**
4. Увлечения – игры, просмотр фильмов (Люди любящие играть в различные игры, особенно динамические, больше всего подходят к нашей целевой аудитории. Также поскольку наша игра будет иметь сюжетную составляющую (к тому же основанную на сюжете фильма), то людям, которым интересен помимо геймплея сюжет, также являются нашей целевой аудиторией, люди, увлекающиеся просмотром фильмов как нельзя, кстати подходят под этот критерий);
5. Семейное положение – любое (Семейное положение не играет особой роли и может быть абсолютно любым. Будь то семейный человек, который любит в свободное время поиграть в игры, либо же человек без семейного положения, который также проводит свое свободное время);
6. Приблизительный доход – 15–50 тыс. руб. (Наш продукт будет стоить в районе 150 руб. поэтому заработок не обязательно должен быть высоким).

Первые пользователи:

1. Друзья (которые подходят под определенные сегменты, которые были описаны выше);
2. Родственники (которые подходят под определенные сегменты, которые были описаны выше).

5.2. Проблемы и существующие альтернативы

Проблемы:

1. Однообразный геймплей;
2. Отталкивающая графика;
3. Казуальность игры.

Решения:

1. Разнообразие геймплея различными механиками;
2. Найти готовые бесплатные ассеты, если нет подходящих создать самим;
3. Сделать непростые уровни, чтобы игроку пришлось не легко в прохождении.

5.3. Уникальная ценность продукта

Уникальный геймплей: дабл джамп, возможность передвигаться по потолку, управление курсором.

5.4. Решение проблем

Решения:

1. разнообразие геймплея множеством механик;
2. поиск уже существующих бесплатных ассетов, либо же их покупка;
3. создание множество вражеских AI, увеличение динамики игры.

5.5. Каналы продвижения

Про рекламу информацию найти не удалось, но можно сказать, что в магазинах Steam и Epic Games есть покупка слотов в «рекомендованные» и «популярные» игры, так же как и рекламой и антирекламой могут выступать сами игроки, которые пишут отзывы и делают обзоры на игры.

5.6. Потоки прибыли

Продажа игры в размере 150 руб. Также продажа шаблона игры в размере 5000 руб. разработчикам игр.

5.7. Структура расходов

1. найм специалистов;
2. найм бета тестеров, премии команде за работу с неисправностями;
3. найм SMM менеджера;
4. приобретение готовых текстур, спрайтов, музыки.

5.8. Ключевые показатели

Опрос, оценки пользователей демо уровня на торговой площадке, количество пользователей.

**5.9. Скрытое преимущество**

Уникальный геймплей с подходящим музыкальным сопровождением.

ЗАКЛЮЧЕНИЕ

Во время анализа доступных источников было проведено исследование понятия компьютерная игра, во время которого была проведена классификация компьютерных игр по 4 критериям, но из-за сравнительной молодости игровой индустрии, а также того, что классификация компьютерных игр не была систематизирована, составить подробную классификацию не удалось. Дополнительно был составлен алгоритм разработки видеоигр.

Были проанализированы популярные средства разработки. В ходе анализа, было проведено их сравнение и выбраны наиболее актуальные средства разработки для начинающих разработчиков. Выбор приоритетных средств разработки проходил по двум критериям: доступность и функциональность.

При анализе существующих разработок, был проведено их сравнение и выделены их достоинства и недостатки. В ходе анализа стало ясно, что при разработке компьютерной игры с простой игровой механикой, стоит обратить внимание на дополнительные элементы игры, такие как сюжет и графическое оформление. Это нужно для того, чтобы удержать потенциального игрока и продлить жизненный цикл разработки.

Основываясь на все полученной в ходе исследования информации, было решено разработать прототип двумерного платформера для одного игрока на игровом движке Unity. Такое решение было принято по нескольким причинам:

1. двумерная графика, в отличие от трехмерной легче в создании;
2. по игровой механике, игра жанра платформер проще реализуется;
3. игровой движок Unity распространяется бесплатно и позволяет разрабатывать приложения на языке программирования C#.

После выбора средств разработки было начато изучение Unity, а также разработка самого проекта. В ходе разработки был изучен игровой движок Unity и были приобретены необходимые знания и умения, а именно:

* создание анимаций;
* создание сцен;
* создание и написание скриптов;
* создание UI;
* создание AI;
* настройка объектов;
* компиляция проекта.

Освоение среды разработки Unity несет немаловажный характер, так как в современном мире индустрия разработки игр все сильнее распространяется в нашем обществе. Игры перестали быть лишь предметом для развлечений, и теперь используются и в других областях, например, в науке или в обучении пользователей. Поэтому развитие в данном направлении можно считать одним из самых важных в современном обществе. В ходе реализации проекта были выполнены следующие задачи:

1. изучены особенности и состояние компьютерной индустрии России;
2. выбраны жанр, вид и платформа для компьютерной игры;
3. разработаны сценарий и концепция основных элементов;
4. выбрано и изучено средство реализации;
5. подготовлены необходимые для игры анимации;
6. реализован прототип игры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Компьютерные игры как искусство [Электронный ресурс]. — Режим доступа: http://gamesisart.ru (дата обращения: 05.05.2022).
2. Торн А. Основы анимации в Unity [Текст]: учебное пособие/ Алан Торн– ред.: Д. Мовчан, переводчик: Р. Рагимов – Москва: ДМК, 2016 – 176с.
3. Ильин В. Основы создания 2D персонажа в Unity 3D 4.3 [Электронный ресурс]/ Ильин Вячеслав // Хабрахабр. — Режим доступа: https://habrahabr.ru/post/211472/ (дата обращения: 05.05.2022).
4. Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C# [Текст]: учебное пособие/ Джозеф Хокинг – Санкт-Петербург: Питер, 2016 – 336с.
5. Unity Manual, Unity Documentation [Электронный ресурс]: справочник. — Режим доступа: https://docs.unity3d.com/Manual/ (дата обращения: 05.05.2022).
6. Unity Game Engine [Электронный ресурс]. — Режим па: https://unity3d.com/ru (Дата обращения: 04.05.2017).
7. Майк Гейг. Разработка игр на Unity 2018 за 24 часа [Текст] — Режим доступа: http://gamesisart.ru (дата обращения: 05.05.2022).
8. .
9. .
10. .
11. .
12. .
13. .
14. .
15. .
16. .
17. .
18. .
19. .
20. .
21. .
22. .
23. 12
24. 12
25. 12
26. 12
27. 12
28. 12
29. 12
30. КОНЕЦ!

## 