Пермский филиал федерального государственного автономного   
образовательного учреждения высшего образования   
«Национальный исследовательский университет   
«Высшая школа экономики»

*Факультет экономики, менеджмента и бизнес-информатики*

Щелкунов Андрей Александрович

**РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ "WORMS"**

*Курсовая работа*

по направлению подготовки *09.03.04 Программная инженерия*

образовательная программа «Программная инженерия»

Руководитель:

Старший преподаватель кафедры информационных технологий в бизнесе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

В.В. Лебедев

Пермь, 2017 год

Аннотация

В работе описываются этапы разработки игровых механик на примере игры Worms с применением платформы Unity и языка программирования C#. На каждом этапе демонстрируются диаграммы языка моделирования UML. Приводятся описание сценария игры, математическая модель, программная модель. Показываются процессы перевода сценария в формальную модель и в модель реализации.

Работа состоит из 5 глав и 1 приложения, занимает 52 страницы и содержит 15 рисунков и 7 таблиц.

Оглавление

[Введение 5](#_Toc478398022)

[Глава 1. Анализ задачи и разработка требований 7](#_Toc478398023)

[1.1. Постановка задачи, формулировка требований 7](#_Toc478398024)

[1.1.1. Диаграмма прецедентов 7](#_Toc478398025)

[1.1.2. Диаграмма активностей 9](#_Toc478398026)

[1.2. Основные элементы игры 10](#_Toc478398027)

[1.2.1. Главный цикл игры 10](#_Toc478398028)

[1.2.2. FPS 11](#_Toc478398029)

[1.2.3. Ресурсы 11](#_Toc478398030)

[1.2.4. Сцена и игровой мир 11](#_Toc478398031)

[1.3. Анализ аналогов 12](#_Toc478398032)

[1.3.1. Обзор и анализ игры Worms Armageddon 12](#_Toc478398033)

[1.3.2. Достоинства игры 14](#_Toc478398034)

[1.3.3. Недостатки игры 15](#_Toc478398035)

[1.4. Диаграмма понятий 15](#_Toc478398036)

[1.5. Диаграмма последовательностей 16](#_Toc478398037)

[1.6. Анализ инструментария Unity3D 17](#_Toc478398038)

[1.7. Сценарий игры и выявление механик 18](#_Toc478398039)

[Глава 2. Формализация описания решений задачи 21](#_Toc478398040)

[2.1. Анимация 21](#_Toc478398041)

[2.2. Физика 23](#_Toc478398042)

[2.3. Обработка столкновений 24](#_Toc478398043)

[Глава 3. Проектирование системы 26](#_Toc478398044)

[3.1. Проектирование архитектуры системы 26](#_Toc478398045)

[3.1.1. Компонентная архитектура 26](#_Toc478398046)

[3.1.2. Диаграмма классов 28](#_Toc478398047)

[3.2. Проектирование пользовательского интерфейса 29](#_Toc478398048)

[3.2.1. Стек состояний 29](#_Toc478398049)

[3.2.2. Начальный экран 32](#_Toc478398050)

[3.2.3. Главное меню 32](#_Toc478398051)

[3.2.4. Игровой процесс 32](#_Toc478398052)

[Глава 4. Реализация программной системы 33](#_Toc478398053)

[4.1. Выбор инструментального средства 33](#_Toc478398054)

[4.2. Процесс разработки 33](#_Toc478398055)

[4.3. Диаграмма компонентов 35](#_Toc478398056)

[Глава 5. Тестирование приложения. Подготовка документации 37](#_Toc478398057)

[5.1. Документация 37](#_Toc478398058)

[5.2. Тестирование 37](#_Toc478398059)

[Заключение 41](#_Toc478398060)

[Библиографический список 42](#_Toc478398061)

[Приложение A. Техническое задание 43](#_Toc478398062)

Введение

Качественный процесс разработки компьютерных игр требует использования современных инструментальных технологий. Они позволяют не только в разы ускорить процесс разработки, но и значительно упростить его. Одним из таких инструментов является игровой движок **Unity3D**. Это современное средство разработки игр, позволяющее создавать 2D и 3D игры и импортировать их на различные платформы.

В данной работе будут описаны этапы разработки компьютерной игры **Worms**. Это пошаговая (turn-based) игра, которая сочетает в себе элементы жанра стратегии и артиллерии (ballistics calculator). Жанр является довольно востребованным, однако связанные с ним игровые механики не реализованы кроссплатформенными средствами.

**Проблема** работы заключается в реализации игровой модели с помощью современных технологий разработки компьютерных игр.

**Объектом** исследования является разработка игры Worms для компьютерной платформы.

**Предмет** исследования – использование игрового движка Unity3D для разработки игры Worms.

**Цель** работы – разработка игровых механик средствами Unity3D на примере игры Worms. Для достижения этой цели необходимо решить следующие **задачи**:

1. Анализ задачи и разработка требований к системе.
2. Формализация описания решения задачи. Выбор и/или разработка алгоритмов для реализации системы.
3. Проектирование приложения.
4. Реализация программной системы.
5. Тестирование и отладка приложения. Подготовка документации.

Последовательное описание выполнения этих задач задает структуру работы, которая разделяется на соответствующие главы.

На этапе анализа будет изучена информация о предметной области, сформулирован набор требований, описаны бизнес‑процессы, проанализированы современные технологии разработки компьютерных игр, рассмотрены аналоги с выявлением достоинств и недостатков, описан сценарий игры и выявлен набор механик, которые составляют геймплей игры.

На втором этапе работы на основе сценария будет построена математическая модель с выделением геометрической модели мира, сцен, игровых объектов и способами их взаимодействия. Также будет осуществлен выбор и приведено описание алгоритмов реализации, в данном случае игровых механик, которые управляют поведением игры.

На стадии проектирования будет описана архитектура системы с рассмотрением отдельных компонентов, классов и их поведения, построена диаграмма классов UML. Будут спроектированы рассмотренные на этапе анализа механики. Затем будет выполнено проектирование графического интерфейса с учетом решаемых системой задач.

На этапе реализации будет описан процесс перевода формальной модели в программную с точки зрения игровых объектов. С учетом описанных требований в систему будут интегрированы существующие алгоритмы реализации необходимых механик. Будут разработаны сцены и создан набор ресурсов (изображения, анимации, звуки, карты, шрифты, скрипты и т.д.), используемых при реализации прототипа.

Этап тестирования и отладки предназначен для проверки надежности разрабатываемой системы. Будет составлен набор тестов, проверяющих качество программы и разработана программная документация.

# Анализ задачи и разработка требований

На этапе анализа изучается информация о предметной области, описывается постановка задачи и формулируются требования. Выделяются и описываются понятия предметной области, их структуры и связи. Также выполняется обзор существующих аналогов и игровых движков, упрощающих процесс разработки, рассматриваются основные сценарии игры.

* 1. Постановка задачи, формулировка требований

Наиболее важной частью этапа анализа является постановка задачи и формулировка требований, цель которой – выявить и описать ситуации, когда пользователь взаимодействует с системой и описать бизнес-процесс для демонстрации контекста возникновения прецедентов. Для описания и анализа функциональных требований используется диаграмма прецедентов. Описание бизнес-процесса системы изображается с помощью диаграммы активностей.

* + 1. Диаграмма прецедентов

**Высокоуровневый прецедент.**

Название: Игра в Worms.

Акторы: Игрок.

Краткое описание: Игрок запускает игру. Выбирает уровень, в который хочет сыграть. Играет, управляя червяком.

**Развернутый прецедент.**

Название: Игра в Worms.

Акторы: Игрок.

Краткое описание: Игрок запускает игру. Выбирает уровень, в который хочет сыграть. Игра дает игроку право управлять одним из червяков. Игрок перемещает червяка по карте. Червяк ползает, прыгает. Игрок стреляет по другим червякам. Червяк достает оружие, наводит прицел, запускает снаряд.

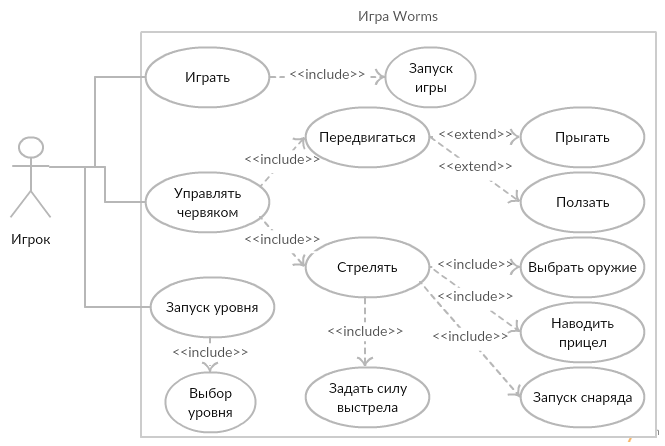
Триггер: Игрок запускает игру.

Основной поток (см. табл. таблица 1.1):

***Таблица 1.1. Диаграмма прецедентов***

| Действия акторов | Отклик системы |
| --- | --- |
| 1. Игрок запускает игру. | 1. Система запускает игру, инициализирует настройки и выводит основное меню игры. |
| 1. Игрок выбирает уровень. | 1. Игра запускает выбранную сцену, расставляет команды червяков на ландшафте. |
| 1. Игрок заставляет червяка ползать и прыгать по карте. | 1. Червяк перемещается по ландшафту. |
| 1. Игрок выбирает оружие, наводит прицел, выбирает силу выстрела, запускает снаряд. | 1. Червяк делает выстрел в указанном направлении в указанной силой выстрела. |

С помощью диаграммы прецедентов игровой процесс описывается на рис. рисунок 1.1.

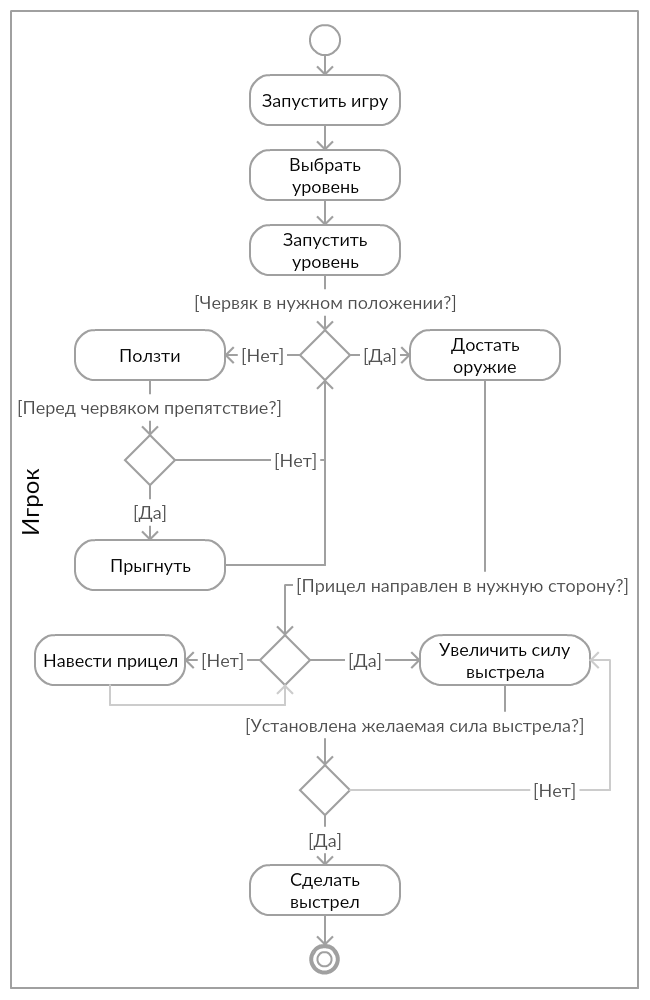


***Рисунок 1.1. Диаграмма прецедентов***

Здесь показаны основные действия, которые необходимо выполнить игроку в процессе игры. Наиболее важными возможностями червяков, которые могут быть инициированы по запросу игрока являются передвижение по карте и стрельба. Как видно, передвигаться по карте червяк может двумя способами: ползти и прыгать. Чтобы червяк сделал выстрел необходимо сначала выбрать оружие, навести прицел, задать силу выстрела и запустить снаряд.

* + 1. Диаграмма активностей

На рис. рисунок 1.2 показан процесс одного хода игрока в нотации диаграммы активностей.



***Рисунок 1.2. Диаграмма активностей***

Здесь показаны действия, которые необходимо выполнить игроку, чтобы сделать один ход. Первым делом необходимо включить игру, выбрать и запустить уровень, после чего система предоставит право управлять одним из червяков пользователю. Цель пользователя – атаковать вражеского червяка. Для этого нужно сначала приползти в нужное место, при необходимости перепрыгивая препятствия ландшафта. Затем игрок выбирает оружие, наводит прицел, устанавливает силу выстрела и запускает снаряд.

* 1. Основные элементы игры

Перед началом разработки имеет смысл определиться с тем, что из себя представляет компьютерная игра. Игра – это основанная на правилах система, внутри которой игрок прилагает усилия в виде некоторой последовательности действия, чтобы повлиять на исход игры с целью получения эмоционально-значимого результата [1]. Любая игра имеет свои правила (или механики), которые определяют процесс протекания игрового процесса (gameplay), взаимоотношения между объектами игры и необходимые для победы или поражения условия. Из этого определения также следует, что игра представляет собой машину состояний (finite state machine), которая начинается с состояния инициализации, а действия игрока меняют состояние в соответствии с игровыми механиками.

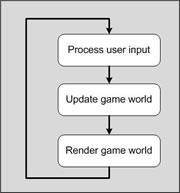
В зависимости от геймплея игры разделяются на жанры. Как уже было сказано ранее, Worms сочетает в себе элементы жанров стратегии и артиллерии (ballistics calculator). Как правило, основными в таких играх механиками являются физика, социальное взаимодействие и тактическое маневрирование [1].

* + 1. Главный цикл игры

Переход игры из одного состояния в другое должен быть представлен так, чтобы игрок осознал результат своих действий. Игровые механики также могут переводить игру из одного состояния в другое, а игра должна каким-то способом уведомить игрока об изменениях. Для этой цели служит главный цикл игры. Его задачами являются:

1. Обработка событий.
2. Обновление игры.
3. Вывод результатов на экран (rendering).

Схематично этот процесс представлен на рис. Рисунок 1.3:



***Рисунок 1.3. Основной цикл игры***

Обработка событий необходима для фиксации действий пользователей, которые переводят игру из одного состояния в другое. Т.к. изменить состояние игры может не только игрок, но и заложенные в нее механики, то во время обновления игрового мира учитывается влияние и того, и другого. И, наконец, третьим действием цикла является рендеринг игрового мира с новым состоянием на экран [2].

* + 1. FPS

Каждая итерация цикла является кадром. Кадрами в секунду (FPS, frames per second) называется количество обновлений экрана за одну секунду, которое способна выполнить игра [3].

Человеческий глаз воспринимает 24 кадра в секунду. Для того чтобы анимация во время игры выглядела плавной, необходимо обеспечить FPS не ниже этого значения. Как правило, многие игры имеют FPS равный около 60. На это значение могут влиять несколько факторов, включая среднюю скорость перерисовки изображения монитором, производительность компьютера и эффективность заложенных в игру алгоритмов [4].

* + 1. Ресурсы

Любая игра работает с некоторым набором ресурсов, которые создают общую атмосферу игры. Ресурсами в игре называются внешние компоненты, которые подгружаются в игру во время выполнения [2]. К таким ресурсам могут относиться изображения, звуки, музыка, анимация, 3D-модели, скрипты и т.д.

* + 1. Сцена и игровой мир

Одним из важных понятий игры является сцена. Сцена – это совокупность визуальных сведений о сущностях объективной или виртуальной реальности в сознании наблюдателя. В данном случае применяется виртуальная реальность, т.е. визуальные сведения создаются с помощью компьютерной графики. Воспринимаемое зрителем изображение, как правило в компьютерных играх, относится к классу абстрактных, т.е. тех, цель которых – донести эмоцию. При этом воспринимается только та часть визуальной информации, которая является для наблюдателя наиболее желаемой.

Сцена должна состоять из объектов. В зависимости от игры они могут быть совершенно различными и иметь разные модели поведения, заложенные в игровые механики. При этом игрок имеет возможность взаимодействовать и управлять расположенными на сцене объектами также согласно правилам игровых механик.

Запуская игру, игрок попадает в игровой мир. Игровой мир – набор сцен, которые сменяют друг друга согласно игровому сценарию. В каждый момент времени игрок может взаимодействовать только с одной сценой игрового мира, и, чтобы наблюдать другую сцену, необходимо выполнить некоторое условие, предусмотренное сценарием игры. Как правило, сцены представляют из себя набор уровней, каждый из них имеет какую‑либо цель, которую необходимо достичь игроку, для перехода на следующий уровень. Такой последовательный переход (продвижение по игре) контролируется механикой прогресса и описывается в линейном сценарии [1].

* 1. Анализ аналогов

Следующим важным этапом на фазе анализа является обзор аналогов. Здесь также выполняется анализ существующих решений с точки зрений предъявляемых к системе требований и выявление преимуществ и недостатков применяемых подходов.

* + 1. Обзор и анализ игры Worms Armageddon

Классической игрой рассматриваемого жанра является Worms Armageddon. Это двумерная пошаговая игра с элементами жанров артиллерии и стратегии. Цель игры – уничтожить команду противника.

В распоряжение игрока поступает команда червяков, каждый из которых имеет некоторое количество единиц жизни, умеет перемещаться по сцене (ползать и прыгать), имеет набор вооружения и может использовать его против червяков (включая своих и червяков вражеской команды), нанося им урон в зависимости от силы оружия. Червяки погибают, когда их очки здоровья опускаются до нуля.

Игрок в один момент времени может управлять только одним червяком и может сделать только один ход, после чего право хода передается червяку вражеской команды. Ход считается совершенным после единоразового использования какого-либо оружия или по истечению времени на ход.

Червяки располагаются на ландшафте, который может быть разрушаемым (ландшафт разрушается при взаимодействии со снарядами вооружения). Также на сцене присутствует вода, при взаимодействии с которой червяк погибает. Помимо червяков на сцене могут быть и другие объекты, такие как ящики, из которых можно взять новое оружие, бочки, которые взрываются при нанесении им урона и мины, которые взрываются через случайный промежуток времени при активации. Пример игровой сцены Worms Armageddon изображен на рис. рисунок 1.4.



***Рисунок 1.4. Сцена сражения в игре Worms Armageddon***

Все объекты сцены подвергаются физическим законам. Наиболее значимыми для игрового процесса законами являются сила притяжения и ветер. Ветер действует только на запущенные снаряды вооружения и изменяет траекторию полета, что в зависимости от ситуации может мешать или помогать игроку выполнить маневр.

В игре присутствует широкий выбор разнообразного вооружения. Некоторые из них доступны игроку сразу в начале игры, другие необходимо открыть, выполняя определенные условия. Вооружение делится на типы, которые включают:

* Оружие ближнего боя, которое наносит урон только близко стоящему червяку (топор, вспышка, «палец» и т.д.).
* Оружие, стреляющее снарядами. Снаряды подвергаются действию физических законов и летят по баллистической траектории (базука, граната, огнемет, лук и т.д.).
* Оружие, стреляющее пулями. На эти виды вооружения физика не действует, поэтому пули летят ровно по прямой и, как правило, достигают цели мгновенно (пистолет, пулемет, автомат-узи и т.д.).
* Управляемые, которыми может манипулировать игрок сразу после выстрела («овца», «старушка» и т.д.).
* Авиаудары позволяют сбросить снаряд с воздуха, указав при этом в какую точку ему следует приземлиться (истребитель, минный удар, «ослик» и т.д.).
* Взрывающиеся. К этим типам вооружения относятся большинство оружия, стреляющего снарядами (которые взрываются при контакте с ландшафтом или по истечению заданного времени) и те, которые червяк выставляет прямо перед собой (мина, динамит).
* Массовые. Это оружие в абсолютном большинстве случаев наносит урон всем червякам на сцене и, как правило, может переломить ход игры («весы правосудия», землетрясение, армагеддон и т.д.).
* Приспособления, которые по сути не являются оружием, но могут быть полезны при выполнении той или иной тактики (веревка, балка, телепорт и т.д.).

Как и любая другая игра, Worms Armageddon позволяет игроку изменять параметры игры и настраивать параметры сцены (количество команд, количество червяков, начальный боезапас, карта ландшафта и т.д.).

В игре имеются несколько режимов игры:

* Одиночная игра с компьютером. Здесь одной из команд управляет игрок, а другой – компьютер.
* Совместная локальная игра. Игра на одном компьютере, при которой несколько человек управляют каждый своей командой.
* Игра по сети. Также совместная игра, но каждый игрок управляет командой с отдельного компьютера.
  + 1. Достоинства игры

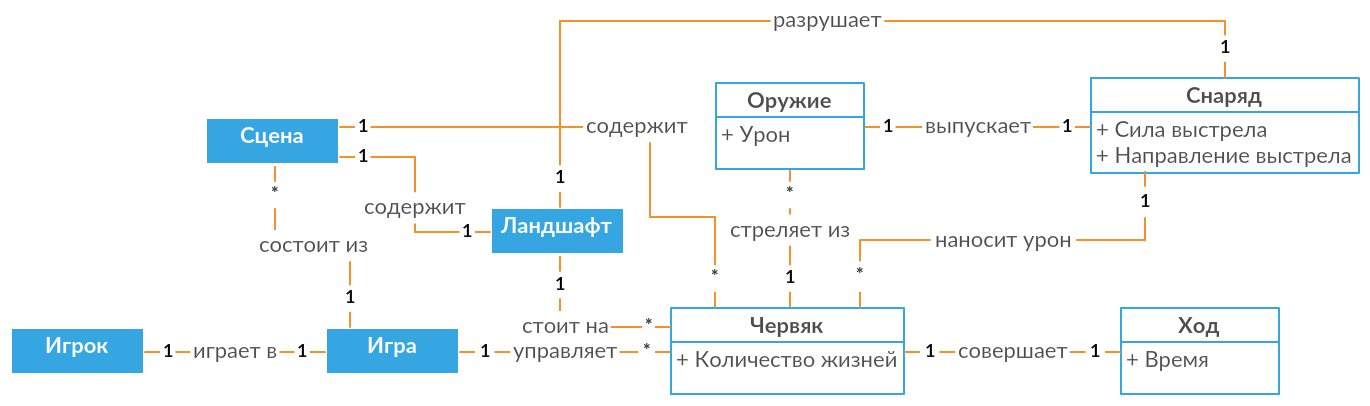
К достоинствам Worms Armageddon относятся:

1. Широкий выбор вооружения. Это позволяет игрокам использовать самые различные тактики сражения, что делает каждую игру неповторимой.
2. Хорошо продуманные механики. В игре подобран наиболее оптимальные набор механик, каждая является значимой для общего геймплея, нет лишних механик, не влияющих на процесс игры.
   * 1. Недостатки игры

К недостаткам игры можно отнести следующие:

1. Плохая настройка урона вооружения. Некоторые виды вооружения имеют низкий урон по сравнению с их аналогами. В этом случае целесообразнее выбрать более мощное оружие, при этом более слабое вооружение используется редко, либо вообще не используется.
2. Игра поддерживает только платформу Windows. Только одна игра из серии игр Worms реализована кроссплатформенными средствами, что позволяет запускать ее на различных платформах. Остальные игры (включая Worms Armageddon) работают только на Windows.
   1. Диаграмма понятий

После анализа аналогов появляется возможность выделить основные понятия, характерные для рассматриваемой предметной области, их структуру и взаимосвязи и построить концептуальную модель. Модель изображена на рис. рисунок 1.5 в нотации диаграмм понятий.

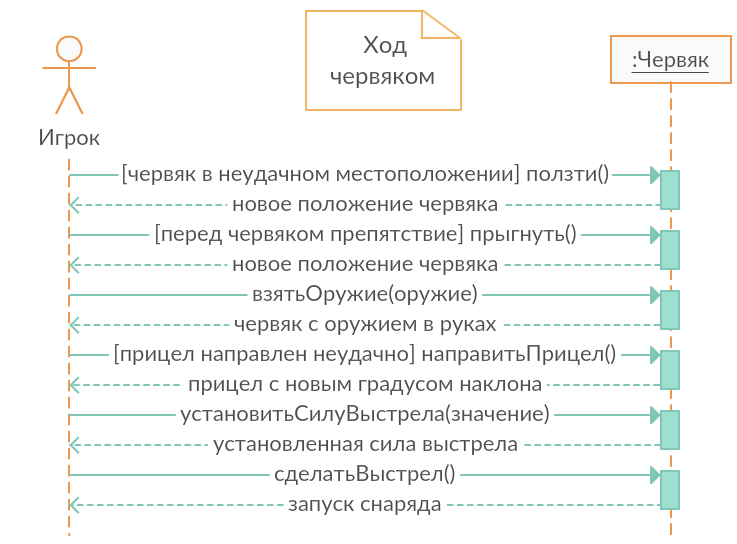


***Рисунок 1.5. Диаграмма понятий***

Здесь показаны основные понятия, характерные для игры, а также их взаимосвязи. Первым понятием является игрок, который запускает и играет в игру. Игра состоит из набора сцен, которые содержат игровые объекты, в данном случае ландшафт и несколько червяков. Червяки располагаются на ландшафте. Игрок передает игре команды, а она в свою очередь интерпретирует их и применяет к червякам. Червяк может совершить ход, если он сделает выстрел или закончится время хода. Стреляет червяк из оружия, которое имеет определенный урон. Во время выстрела оружие выпускает снаряд с заданными силой и направлением. Коснувшись ландшафта снаряд взрывается, разрушая его и нанося близко стоящим червякам урон, который отнимает определенное значение из их очков жизней.

* 1. Диаграмма последовательностей

Следующим этапом анализа является построение модели поведения системы для отображения состава, порядка и параметров сообщений. Модель поведения в нотации диаграммы последовательностей изображена на рис. рисунок 1.6.



***Рисунок 1.6. Диаграмма последовательностей***

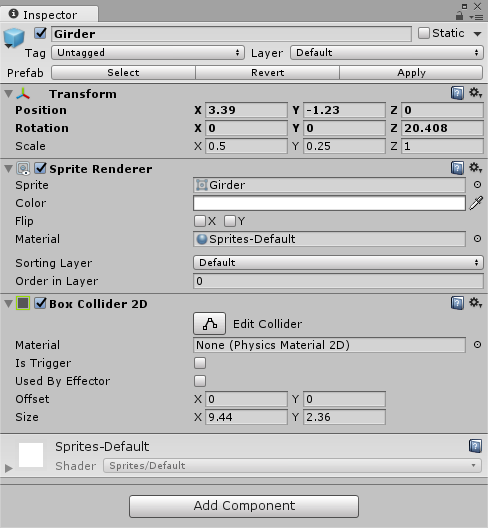
На диаграмме показана последовательность действий, которые должен сделать игрок, чтобы совершить ход червяком. В целом она соответствует диаграмме активностей, которая была приведена ранее. Если в начале хода червяк находится в неудачном месте, игрок применяет к нему функцию «ползти()», которая изменяет месторасположение червяка. В случае если перед ним возникают препятствия, игрок вызывает функцию «прыгнуть()», чтобы преодолеть его. Когда червяк находится в правильном месте игрок выбирает оружие, которым он хочет совершить выстрел, а червяк достает его. Затем игрок изменяет угол наклона прицела, пока ему не покажется, что снаряд полетит в нужном направлении. После этого игрок устанавливает силу выстрела и запускает снаряд.

* 1. Анализ инструментария Unity3D

Unity3D – это современное кроссплатформенное средство, предоставляющее богатый инструментарий для разработки компьютерных игр. Основными преимуществами Unity являются:

* Качество, скорость и простота разработки.
* Возможность кроссплатформенной разработки.
* Множество редакторов, позволяющих делать тонкую настройку любого компонента игры.

Наиболее часто встречающимся понятием в Unity является игровой объект, который по сути является любым объектом в игре. С каждым объектом может быть связан некоторый набор ресурсов (assets), которые представляют собой изображения, звуки, анимации, модели, скрипты и т.д. Объекты также могут иметь несколько свойств, характеризующих их поведение в игре [5]. Таким образом в Unity реализуется компонентная архитектура: игровому объекту присваивается новый компонент, и объект приобретает свойства этого компонента и реализует заложенную в него логику. Пример объекта с некоторым набором компонентов приведен на рис. рисунок 1.7.



***Рисунок 1.7. Пример игрового объекта с тремя компонентами***

Здесь представлен игровой объект girder (балка) с тремя компонентами: transform, sprite renderer и box collider 2D. Компонент transform есть абсолютно у всех игровых объектов. Он отвечает за три основных элемента, связанными с трансформацией: координаты, поворот, масштаб. Компонент sprite renderer отвечает за отрисовку объекта в игре. Он должен иметь ссылку на изображение из набора ресурсов, которое будет выводиться на экран. И, наконец, box collider 2D отвечает за проверку объекта на столкновения (collision detection) с другими коллайдерами.

Unity предоставляет множество других стандартных компонентов, которые можно назначать любому объекту. Также есть возможность создания новых компонентов, реализованных в виде скриптов с использованием объектно-ориентированного языка C#.

Другим фундаментальным понятием в Unity является сцена, о которой уже шла речь ранее в главе. Здесь сцена также состоит из игровых объектов и доносит до наблюдателя содержащуюся в ней визуальную информацию. Сцену также можно рассматривать как игровой ресурс, который демонстрирует текущее состояние игры.

Сцены (состояния) сменяют друг друга при определенных условиях. Последовательность сцен составляет сценарий игры.

* 1. Сценарий игры и выявление механик

Сценарий – последовательность игровых сцен, каждая со своим определенным сюжетом. Наиболее значимым элементом сценария игры Worms являются червяки, которые демонстрируют свои способности, в которых нуждается игрок в определенных условиях игры.

Первое, что видит игрок при запуске игры – начальный экран и главное меню. Меню является важной составляющей для навигации по игре. Оно позволяет начать игру, изменить параметры и выйти из игры. Наиболее интересный сценарий представляет сам процесс игры.

На игровой сцене находится ландшафт, на котором располагаются команды червяков. Ландшафт является статичным объектом, а червяки динамичными. Тот факт, что червяки могут стоять на ландшафте, означает, что в игре должны быть реализованы взаимодействие объектов (в данном случае «ландшафт‑червяк»), обработка столкновений и физический движок, отвечающий за притяжение червяков к земле.

Главной возможностью червяков является передвижение по ландшафту, которое реализуется с помощью команд пользователя. В передвижение включаются ползание и прыжки. При передвижении червяк меняет свое местоположение на ландшафте. Возможность передвижения зависит от угла находящейся перед червяком части ландшафта – на слишком большой угол червяк заползти не сможет. Для того чтобы игрок смог воспринять информацию о том, что червяк передвигается, одного изменения положения недостаточно. Поэтому еще одной механикой является здесь анимация движения.

Вторая важная возможность червяков – стрельба. Во время стрельбы червяку необходимо выбрать оружие и сделать выстрел. Некоторое оружие имеет силу выстрела, от которого будет зависеть траектория полета снаряда. Летящий снаряд взрывается при контакте с ландшафтом. Следовательно, здесь также необходима проверка на столкновения (взаимодействие «ландшафт-снаряд»), и физика, чтобы заставить снаряд лететь по баллистической траектории.

Снаряд, попавший в ландшафт взрывается, тем самым нанося урон близко стоящим червякам. Взрыв, как и передвижение, требует анимации, однако главными его эффектами являются нанесение урона и взрывная волна. Значит, здесь необходима механика, отвечающая за нанесение урона в зависимости от используемого оружия и расстояния между червяком и точкой контакта снаряда с ландшафтом.

Основой геймплея игры Worms является сражение. Во время битвы активным является только один червяк одной из команд, следовательно, только он может передвигаться и стрелять под управлением игрока. После совершенного игроком хода (например, после выстрел из оружия), активным становится червяк команды соперника, т.е. право хода передается ему. Такое поведение требуется реализации механики смены ходов.

Основной жанр игры Worms – стратегия, следовательно, от игрока требуется наиболее удачно расположить червяков своей команды как для атаки, так и для обороны. Такое поведение реализуется с помощью механики «тактика» [1].

Необходима также механика, следящая за здоровьем и количеством червяков в команде. Главной ее задачей является оповещение игрока о победе или поражении. Условием победы является уничтожение всех вражеских команд, а условием поражения – потеря всех червяков своей команды.

Таким образом выявлен следующий набор механик, реализующих правила поведения игры Worms:

1. Обработка столкновений.
2. Взаимодействие объектов.
3. Физика.
4. Анимация.
5. Передвижение червяков.
6. Стрельба.
7. Нанесение урона.
8. Контроль смены очередности ходов.
9. Тактика.
10. Условия победы/поражения.

Наиболее значимыми являются тактика и механика физики, т.к. они определяют жанр игры Worms (стратегия и артиллерия соответственно). Более того физика выполняет роль core-механики (основной), т.к. она влияет на поведение других механик, в частности она контролирует движение червяков и влияет на траекторию полета снарядов после выстрела [1].

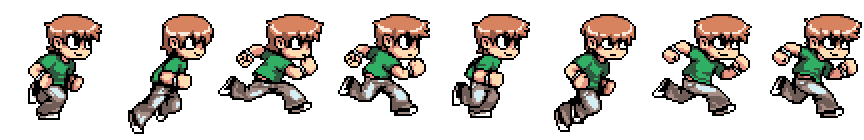
# Формализация описания решений задачи

На этапе формализации строится математическая модель, описывающая решающую проблему, осуществляется выбор (разработка) алгоритмов для реализации компонентов системы. Выполняется анализ математического аппарата и реализуемых алгоритмов, дается их краткая характеристика, оценка сложности и ограничений.

Построение математической модели выполняется с целью замены объекта исследования моделью и изучения свойств оригинала, законов, которым он подчиняется, и связей, которые он имеет с другими объектами.

* 1. Анимация

Анимация – это последовательная смена изображений, в результате которой создается ощущение, что изображение движется. Источником анимации может быть спрайт-лист. Спрайт-лист – это серия анимационных кадров, которые собраны вместе в одном изображении [6]. Пример спрайт-листа представлен на рис. рисунок 2.1:



***Рисунок 2.1. Пример спрайт-листа с анимацией бега***

Отдельные кадры могут находится в разных файлах, однако рекомендуется собирать их в один файл, т.к. такой подход имеет ряд преимуществ. Во-первых, загрузка одного изображения в программу вместо нескольких происходит гораздо быстрее. Во-вторых, файл изображения, в котором собраны все кадры, занимает меньше памяти, чем отдельные файлы (даже, если размеры кадров в обоих случаях одинаковы). В-третьих, один файл может содержать не одну анимацию, а несколько. Это позволяет хранить все связанные анимации в одном месте (например, анимации бега, ходьбы, прыжков одного персонажа) [7].

Для того чтобы реализовать анимацию в программе, необходимо загрузить файл со спрайт-листом в программу, задать размеры и координатные положения кадров, и запустить алгоритм, который с определенной частотой будет «вырезать» нужный кадр из изображения и выводить его на экран. Задержка между кадрами задает скорость анимации: чем ниже задержка, тем быстрее скорость смены кадров.

Алгоритм реализации анимации довольно прост. После загрузки изображения со спрайт-листом необходимо установить координаты и размеры первого кадра. Вместо того, чтобы рисовать все изображение, программе необходимо «вырезать» только выделенный кадр и вывести его на экран. Затем определяются координаты и размеры второго кадра. Как правило, размеры всех кадров одинаковы, поэтому нужно лишь к координатам предыдущего кадра прибавить ширину следующего. Обычно, если размеры и координаты кадров в спрайт-листе различаются, можно сохранить их в отдельном файле, а при необходимости изменения кадра загрузить нужные данные оттуда.

Кадры сменяются по истечению определенного времени, которое называется задержкой. Сразу после прорисовки кадра засекается время. Когда оно достигнет заданной задержки, рисуется следующий кадр, и таймер обнуляется. Обычно задержка между кадрами не отличается, но если это не так, то эти значения также можно хранить в отдельном файле и загружать по мере необходимости.

Анимация имеет свойство цикла. Если есть необходимость зациклить анимацию, то после прорисовки последнего кадра, алгоритм переходит снова к первому. Таким образом анимация выполняется бесконечно. Если это свойство не задано, анимация выполнится только однажды, т.е. каждый кадр из спрайт-листа будет выведен на экран только один раз.

Часто возникает необходимость обратить анимацию. Например, на рис. рисунок 2.1 изображены только кадры бегущего направо персонажа, а персонажа, бегущего налево, анимировать нельзя. Как правило, нет необходимости создания зеркально отраженных кадров, т.к. это требует загрузить в программу еще один спрайт-лист. Альтернативным решением является обращение кадров программно. В этом случае точкой начала координат задается правый верхний угол (вместо левого верхнего), а размеры анимации (в данном случае ширина) задаются с отрицательным значением. Таким образом кадры отразятся по оси X, и персонаж приобретет возможность бежать налево.

* 1. Физика

Наиболее важной механикой в рассматриваемой предметной области является физика. Поведение объектов (в частности, их трансформаций) зависит от того, какие силы на них действуют. Физика не только отвечает за имитацию силы, но и является core‑механикой для обработки столкновений, т.е. поведение объектов при столкновении друг с другом зависит от работы первой механики [1].

За моделирование физических процессов отвечает физический движок. Это программный модуль, в котором заложены алгоритмы, отвечающие за имитацию физических сил, таких как гравитация, импульс, сила трения, сила упругости и т.д. Эти силы влияют на поведение игровых объектов на сцене. На рис. рисунок 2.2 показаны силы, воздействующие на игровой объект.



***Рисунок 2.2. Демонстрация физических сил, действующих на объект***

Как видно из рисунка, каждая сила имеет вектор направления. Сила притяжения действует на объект и притягивает его вниз по оси Y с постоянным ускорением равным 9.8 м/с2. Это значит, что скорость объекта будет увеличиваться на 9.8 м/с каждую секунду. На объект также может действовать импульс, который придает ему некоторую начальную скорость. Если объект соприкасается с другими объектами, например, с землей, то на него также будет действовать сила трения, обратная направлению скорости, которая будет тормозить объект. Скорость затухания скорости зависит от шероховатости поверхностей взаимодействующих объектов. Когда объект лежит на земле, на него также действует сила упругости, которая не позволяет ему провалиться сквозь землю. Ее вектор является противоположным по отношению к вектору силы притяжения [8].

На физический объект постоянно действуют сила притяжения, сила трения и сила упругости. Эти силы контролируются физическим движком, и ничто не может повлиять на их работу. Однако импульс – это сила, которую может контролировать игрок. Например, при ударе по объекту игрок задает ему ускорение, а остальные силы влияют на поведение объекта после удара.

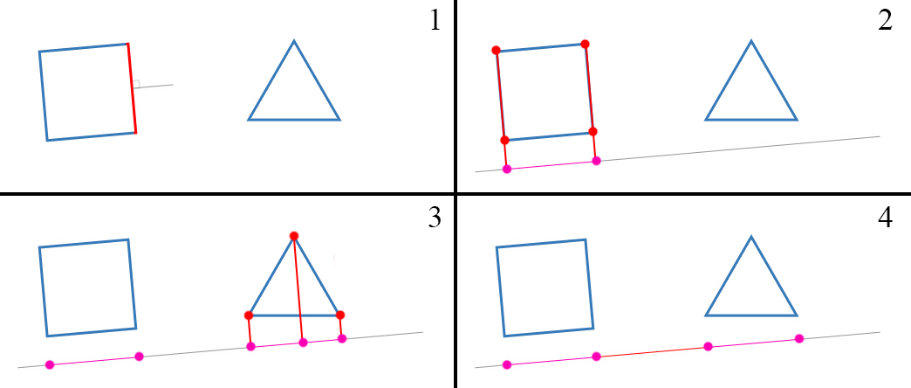
* 1. Обработка столкновений

Как правило, физические движки позволяют обнаруживать столкновения двух объектов. Обработка столкновений необходима в тех случаях, когда необходимо проверить, имеют ли два объекта общие точки. Если общие точки найдены, то два объекта считаются пересекающимися, что в зависимости от ситуации можно интерпретировать разными способами, например, факт того, что персонаж стоит на земле, или снаряд попал по врагу и т.д.

Одним из алгоритмов, решающих эту задачу, является теорема разделения осей (SAT, Separation Axis Theorem) [9]. Принцип действия алгоритма основан на векторной алгебре. Реализация алгоритма требует выполнения следующих шагов:

1. У одного из проверяемых на столкновение полигонов выбирается произвольная сторона, к которой проводится нормаль. Эта нормаль становится осью, на которую будет накладываться проекция на следующих шагах.
2. От каждой вершины полигона откладывается проекция на указанную ось.
3. На оси фиксируются максимальная и минимальная точки пересечения.
4. Каждая вершина второго полигона также проецируется на ось, максимальная и минимальная точки фиксируются.
5. Если максимум первого полигона оказался меньше минимума второго полигона, то столкновения между полигонами не обнаружено, иначе столкновение обнаружено, и алгоритм заканчивается.
6. Если тестируемые полигоны еще имеют стороны, к которой не была проведена нормаль, то шаги 1-5 повторяются для новой стороны.
7. Алгоритм повторяется до тех пор, пока не будут протестированы все стороны обоих полигонов.

Схематично данный алгоритм показан на рис. рисунок 2.3.



***Рисунок 2.3. Алгоритм SAT***

Такая последовательность действий подходит для любых выпуклых полигонов, которые не являются окружностями, т.к. они не имеют сторон. В этом случае, однако алгоритм гораздо проще. Необходимо построить отрезок между центром окружности и ближайшей вершиной второго полигона, и если он окажется короче радиуса окружности, то столкновение обнаружено.

Алгоритм SAT имеет следующие преимущества:

1. Скорость. Построение проекции выполняется довольно быстро. Сразу после обнаружения столкновения алгоритм заканчивает работу. В худшем случае необходимо проверить все стороны обоих полигонов, а в лучшем – только одну.
2. Точность. Алгоритм всегда обнаруживает столкновение правильно при любых заданных выпуклых полигонах.

Также имеются и недостатки, которые включают следующие:

1. Алгоритм работает только с выпуклыми полигонами. Для обхода этого недостатка сложный полигон разбивается на несколько простых. Этот процесс называется тесселяцией [4].
2. Алгоритм не дает ответа на то, какие именно стороны полигонов соприкоснулись. Дается только расстояние между ними.

# Проектирование системы

На этапе проектирования определяются логические программные объекты системы, взаимосвязи и взаимодействия между ними. Описывается архитектура системы, выполняется проектирования отдельных компонентов системы. Также выполняется проектирования графического пользовательского интерфейса с учетом решаемых системой задач.

* 1. Проектирование архитектуры системы

Главной задачей, которую необходимо решить на этапе проектирования архитектуры является способ организации программной системы, позволяющий реализовать необходимые решения и удовлетворить поставленным требованиям.

* + 1. Компонентная архитектура

За основу архитектурного стиля игры Worms берется компонентная архитектура. Среди наиболее значимых преимуществ этого стиля выделяются:

* Простота развертывания. Компоненты, подключенные к системе, являются независимыми, что позволяет с легкостью заменять компоненты старых версий на более новые.
* Внешние компоненты. Используя готовые внешние компоненты, можно в разы ускорить и повысить качество процесс разработки.
* Повторное использование. Одни и те же компоненты могут быть использованы для реализации разных частей программы.
* Интерфейс и реализация. Компоненты имеют открытый интерфейс, в то время как детали реализации скрыты от разработчика.

Как уже было сказано на этапе анализа, Unity3D использует компонентную архитектуру, что позволяет гибко настраивать поведение игровых объектов под нужды разработчика. Каждый компонент, подключаемый к игровому объекту имеет все описанные выше преимущества, однако их недостаточно для описания поведения всех объектов в определенных ситуациях. В этом случае создаются скрипты (классы), которые выполняют роль компонентов, и при подключении к объектам придают им описанные свойства и поведение.

Опишем компоненты объекта «Червяк». Стандартными компонентами, которые предоставляет Unity, необходимые для данного игрового объекта, являются transform, sprite renderer, rigid body 2d, box collider 2d и animator. Некоторые из них были описаны на этапе анализа.

Компонент transform придает объекту свойства трансформации, которые включают координаты, поворот и масштабирование по всем трем осям. Т.к. для двумерной игры необходимы только две оси, ось Z не используется. Объект, который имеет данный компонент приобретает возможность перемещения, поворота и масштабирования в игровом мире.

Задача sprite renderer заключается в представлении игрового объекта на сцене. Он связывается с изображением, которое характеризует игровой объект и выводит его с учетом заданных в компоненте transform свойств.

Компонент rigid body 2d придает червяку свойства физического объекта, т.е. он приобретает возможности, характерные для любого объекта в материальном мире. Наиболее важной силой здесь является сила тяжести, которая постоянно притягивает объект вниз. Таким образом в игре реализуется механика физики.

Box collider 2d отвечает за обработку столкновений. Объекты, к которым подключен данный компонент становятся коллайдерами, что позволяет им взаимодействовать с другими коллайдерами. Так, например, действующая на червяка физика постоянно притягивает его к земле, а box collider 2d не позволяет ему провалиться сквозь нее. Понятно, что ландшафт в данном случае тоже должен быть коллайдером. Box collider 2d реализует механику обработки столкновений.

Компонент animator отвечает за анимацию объекта. Известно, что червяк умеет ползать, и, чтобы показать игроку этот процесс, необходима соответствующая анимация, которая проигрывается при определенных условиях игры. Как уже было показано, анимация представляет собой набор кадров, которые сменяют друг друга. В данному случае animator взаимодействует с компонентом sprite renderer, который подставляет нужный кадр. Помимо этого, червяк может иметь несколько других связанных с ним анимаций, таких как прыжок, падение, выстрел и т.д. Все они реализуются подобным образом. Так реализуется механика, связанная с анимацией.

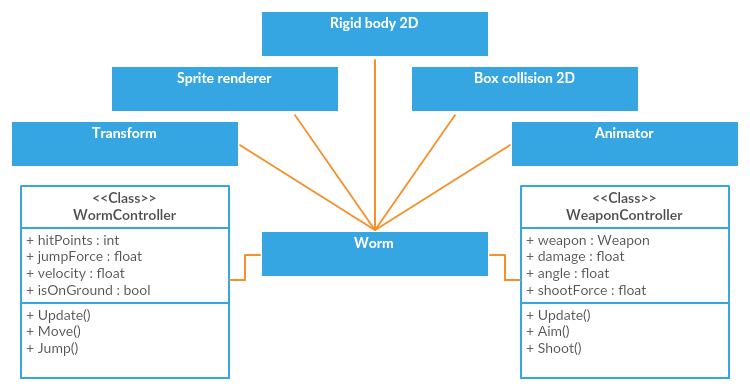
Как уже было сказано, этих компонентов недостаточно для реализации всех механик. До сих пор не решена задача взаимодействия пользователя с червяком. На этапе анализа было выяснено, что главными возможностями червяков являются передвижение и стрельба. Для этих целей создаются два класса WormController и WeaponController, которые становятся компонентами и подключаются к объекту для придания реализованных в скриптах свойств и поведения.

* + 1. Диаграмма классов

Компонент WormController будет отвечать за передвижение игрока, а также содержать основные свойства объекта «Червяк». Как было выяснено на этапе анализа к таким свойства относятся очки здоровья, скорость передвижения, высота прыжка, и переменная, проверяющая стоит ли персонаж на земле или находится в воздухе. API Unity предлагает методы, которые обновляют игровой объект с течением времени. В нем должны быть реализованы функции обработки команд игрока, которые придают червяку соответствующее поведение. К таким функциям относятся передвижение и прыжок.

WeaponController должен отвечать за стрельбу. Как было установлено на этапе анализа, выстрел зависит от его силы и угла, под которым запускает снаряд. Также важен тип оружия и урон, который оно наносит. Здесь опять необходим метод обновления объекта, в котором реализуются функции наведения (изменения угла выстрела), установка силы выстрела и, соответственно, сам запуск снаряда.

Схематично игровой объект «Червяк» и связанные с ним компоненты представлены на рис. рисунок 3.1.



***Рисунок 3.1. Схема компонентов игрового объекта «Червяк»***

Таким образом, данная архитектура описывает все выделенные ранее свойства и поведение игрового объекта «Червяк». Остальные объекты имеют подобную архитектуру, она отличается только компонентами, которые придают объектам необходимые свойства и поведение.

В игре также должны быть механики, которые влияют не на поведение игровых объектов, а на игровой процесс. Как было выяснено, необходимы механики, которые отвечают за контроль условий победы/поражения и контроль смены очередности ходов. За каждый из них должен отвечать отдельный класс. Так, класс GameOverController будет отвечать за проверку количества червяков после каждого хода. Если в одной из команд их количество стало равно нулю, то победа присваивается другой команде, и игра заканчивается. MoveController передает право хода червяку команды соперника после совершенного хода.

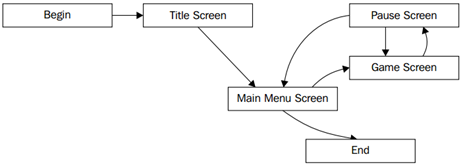
* 1. Проектирование пользовательского интерфейса

Наиболее важной составляющей программы, решающей задачу взаимодействия с пользователем, является графический интерфейс. Он позволяет пользователю взаимодействовать с такими элементами игры, как главное меню, настройки и сам игровой процесс, которые называются состояниями. Контроль перехода игры из одного состояния в другое реализуется с помощью стека состояний.

* + 1. Стек состояний

Как уже было выяснено на этапе анализа, игра представляет собой машину состояний. Процесс перехода из одного состояния в другое инициируется либо пользователем, либо правилами, заложенными в игру. Механики вступают в действие непосредственно во время игры, в то время как пользователи контролируют процесс и во время игры, и перед игрой, когда взаимодействуют с игровым меню, окном настроек, выбором игры и т.д.

Одним из способов показать пользователю определенное состояние игры является стек состояний. Главным преимуществом машины состояний является то, что в любой момент времени только одно состояние является активным, а стек позволяет хранить все состояния игры. Переход между состояниями осуществляется с помощью функций добавления и удаления состояния из вершины стека. Независимо от количества состояний в стеке, активное всегда находится на его вершине. Демонстрация работы стека состояний показана на рис. рисунок 3.2.



***Рисунок 3.2. Стек состояний***

Как видно из рисунка примерами состояний игры могут быть:

1. Начальный экран, демонстрирующий логотип игры или другую подобную информацию. Является первым состоянием, которое видит пользователь, сразу после включения игры.
2. Главное меню игры, с помощью которого игрок может начать игру, изменить параметры, выйти из игры и т.д. Отсюда пользователь имеет возможность перейти в наиболее интересное состояние игры, игровой процесс.
3. Экран игры, демонстрирующий игровой процесс со сценой и игровыми объектами, которыми пользователь может управлять и взаимодействовать.

Помимо этих состояний могут быть также состояния паузы, настройки параметров, выбора уровня, сохранения прогресса и т.д.

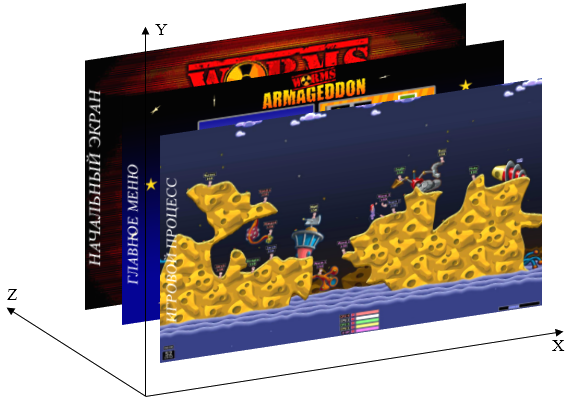
Пример содержимого стека состояний и действия пользователя, приводящие к изменению, показаны в табл. таблица 3.1.

***Таблица 3.1. Демонстрация изменения стека состояний***

| Действие пользователя | Содержимое стека состояний после действия игрока |
| --- | --- |
| Включение игры | 1. **Начальный экран** |
| Нажатие любой кнопки | 1. **Главное меню** 2. Начальный экран |
| Нажатие кнопки «Начать игру» | 1. **Игровой процесс** 2. Главное меню 3. Начальный экран |
| Нажатие кнопки «Пауза» | 1. **Пауза** 2. Игровой процесс 3. Главное меню 4. Начальный экран |
| Нажатие кнопки «Продолжить игру» | 1. **Игровой процесс** 2. Главное меню 3. Начальный экран |
| Нажатие кнопки «Пауза» | 1. **Пауза** 2. Игровой процесс 3. Главное меню 4. Начальный экран |
| Нажатие кнопки «Выход в главное меню» | 1. **Главное меню** 2. Начальный экран |
| Нажатие кнопки «Выход из игры» | Стек пуст, т.к. пользователь закрыл игру |

Как видно из таблицы при определенных действиях пользователя стек состояний изменяется, добавляя или удаляя из себя состояния игры. При этом активным состоянием, т.е. тем, которое видит пользователь в данный момент, является состояние на вершине стека. До начала игры и после завершения стек остается пустым. Несмотря на то, что активным является только одно состояние, остальные точно так же существуют в мире игры до тех пор, пока игрок или игровая механика не выполнят условие, которое задействует остальные состояния.

В мире двумерной игры можно представить, что неактивные состояния находятся «позади» активных, т.е. они дальше от наблюдателя, а активное ближе, поэтому оно «перекрывает» остальные, и игрок видит только ближайшее. В двумерной игре трансформации всех объектов происходят в пределах осей X и Y, однако состояния располагаются на оси Z, которая возрастает вглубь экрана. Следовательно, активное состояние имеет наименьшее значение по оси Z. Реализация сохранения состояний двумерной игры в трехмерном пространстве продемонстрирована на рис. рисунок 3.3.



***Рисунок 3.3. Расположение состояний двумерной игры в трехмерном пространстве***

Как видно из рисунка первым делом при включении игры наблюдатель видит начальный экран, который располагается в произвольном месте на оси Z, после чего открывается главное меню, которое по оси Z располагается ближе к наблюдателю, и затем игровой процесс – это то, что находится на вершине стека и по оси Z принимает наименьшее значение.

* + 1. Начальный экран

Начальный экран является приветственным состоянием для пользователя. Как правило, на нем демонстрируются инструментальные средства, использованные при разработке, создатели игры, логотип игры и т.д. Функциональность этого состояния минимальная. Он всего лишь выводит приглашение нажать любую клавишу, чтобы перейти в главное меню игры.

* + 1. Главное меню

Главное меню представляет собой основное место для перехода к игровому процессу. Здесь располагаются кнопки, загружающие на выбор ту или иную сцену с разным ландшафтом и расположение червяков, а также кнопка выхода, позволяющая закрыть игру.

* + 1. Игровой процесс

В зависимости от выбора пользователя загружается определенная сцена, демонстрирующая игровой процесс. В ней располагается ландшафт и червяки в случайных местах карты. Поверх сцены отображается интерфейс, так называемый HUD (Header-Up Display), на котором отображена дополнительная информация, помогающая игроку: количество очков здоровья червяка, сила выстрела и т.д.

# Реализация программной системы

На этапе реализации выполняется выбор инструментальных средств, описывается процесс разработки механик, выявленных на этапе анализа, а также классов, выделенных на этапе проектирования. Применяются алгоритмы, описанные на этапе формализации.

* 1. Выбор инструментального средства

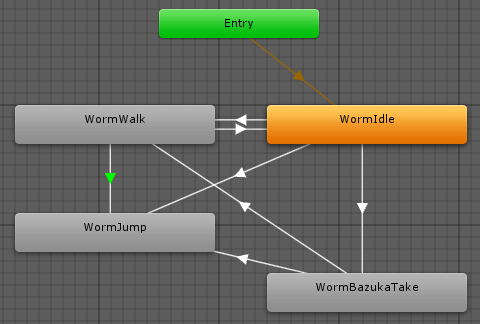
Как было показано на этапе анализа, Unity3D является мощным инструментом для создания компьютерных игр. Он позволяет не только ускорить процесс разработки, но и делает его более качественным. Поэтому именно Unity был выбран в качестве основного инструмента для разработки. Unity поддерживает множество редакторов, необходимых на разных стадиях разработки игры. В данном случае будут использованы лишь некоторые из них, а именно редактор настройки сцены, аниматор, инспектор (для создания шаблонов игровых объектов) и редактор сценариев, позволяющий создавать дополнительные компоненты в среде разработки Visual Studio с использованием объектно‑ориентированного языка C#.

* 1. Процесс разработки

Наиболее важным редактором Unity является редактор сцены. Он позволяет изменять внешний вид сцены, настраивать камеру, расставлять объекты и т.д. Благодаря этому редактору у разработчика появляется возможность сразу увидеть, как будет выглядеть сцена в итоговом варианте, и гибко настраивать параметры игровых объектов для более удобной доводки прямо во время отладки.

Перед тем как расставлять объекты на сцене, сначала их необходимо настроить, присоединить к ним нужные компоненты и подобрать соответствующие значения для каждого свойства. Так, например, как было показано ранее, игровому объекту «Червяк» присваиваются несколько встроенных компонентов и создаются несколько скриптов, добавляющих ему необходимый функционал. После чего из данного объекта создается шаблон, для того чтобы все остальные червяки на сцене имели одинаковые свойства.

Для создания анимаций используется отдельный инструмент, который называется аниматор. Он позволяет добавлять игровому объекту анимации, которые он способен воспроизвести, и настроить условия перехода объекта из одной анимации в другую. На рис. рисунок 4.1 показано, какие анимации способен воспроизвести червяк, и условия перехода одной анимации в другую.



***Рисунок 4.1. Анимации объекта «Червяк»***

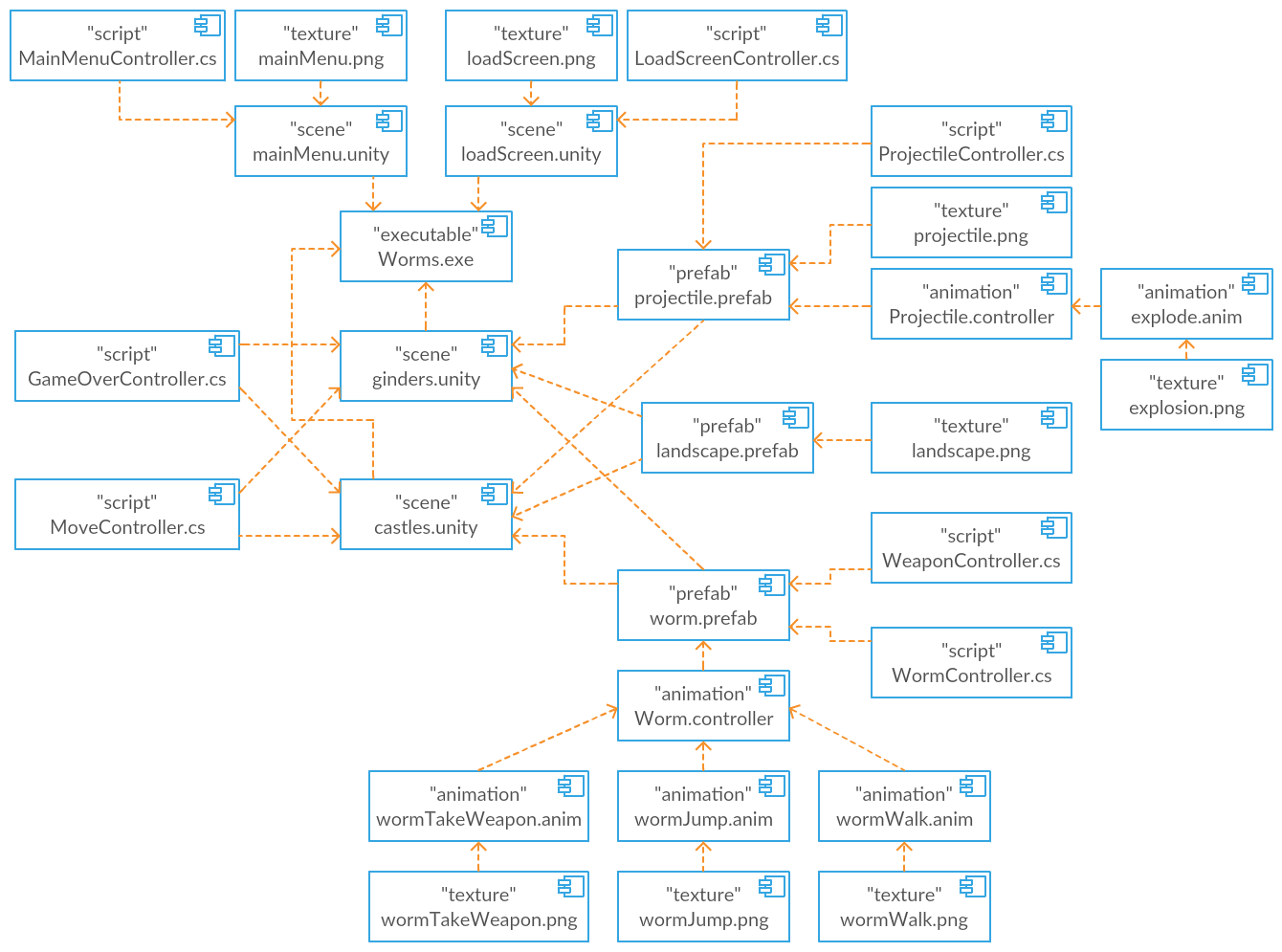
Здесь видно, что основной анимацией червяка является WormIdle, которая по сути состоит из одного кадра и включается, когда он просто стоит на земле. WormWalk активируется когда червяк начинает ползти. Сделать это он может только если до этого стоял или держал оружие. Анимация вытаскивания оружия включается, только если червяк до этого стоял. Прыгнуть персонаж может из любого состояния, однако во время прыжка ни в какое другое перейти не может.

Наиболее интересным является процесс создания новых компонентов, в ходе которого используется интегрированная среда разработки Visual Studio и язык программирования C#. Платформа Unity сводит процесс программирования к минимуму, однако существуют ситуации, в которых игровым объектам необходимо добавить поведение, непредусмотренное в Unity. В этом случае разрабатываются классы, инкапсулирующие поведение объектов.

На этапе проектирования были перечислены классы, которые необходимо добавить в игру для реализации требований. Рассмотрим реализацию класса WormController. Было выяснено, что для этого класса необходимо добавить четыре свойства (hitPoints, jumpForce, velocity, isOnGround) и три метода (Update(), Jump(), Move()) Главным методом является Update(), в котором фиксируются все изменения объекта на каждом кадре, а результаты изменений отображаются на экране при прорисовке. Одним из состояний объекта, которое может измениться во время игры, является скорость, за нее отвечает переменная velocity. Это свойство изменяется посредством статического метода Input.GetAxisRaw("Horizontal"), который вызывается при нажатии клавиш влево и вправо. Затем в методе Move() velocity складывается с текущим положением объекта, и после обновления кадра объект выводится на экран уже с другими координатами. Метод Jump() вызывается при нажатии клавиши пробел, и если свойство isOnGround имеет значение true, то персонажу придается импульс с помощью метода GetComponent<Rigidbody2D>().AddForce(new Vector2(0.0f, jumpForce), что заставляет его подпрыгнуть с указанной в свойстве jumpForce силой. Переменная isOnGround принимает значение true, если было обнаружено столкновение между червяком и ландшафтом. Также во время вызовов функций Jump() и Move() начинают проигрываться соответствующие анимации с помощью метода GetComponent<Animator>().Play("WormWalk").

* 1. Диаграмма компонентов

Во время разработки игры используется множество ресурсов, которые являются компонентами, принадлежащих к тому или иному стереотипу. Этими компонентами является исполняемый файл, сцены, шаблоны, скрипты, аниматоры, анимации и текстуры. Диаграмма компонентов показана на рис. рисунок 4.2.



***Рисунок 4.2. Диаграмма компонентов***

По диаграмме видно, что программа представляет собой исполняемый файл (.exe). Игра в Unity состоит из набора сцен (.unity), которые во время компиляции собираются в исполняемых файл, следовательно, он зависит от сцен. Сцены начального экрана и меню состоят из статичных изображений (.png), и скриптов (.cs), которые предлагают пользователю перейти к главному меню в первом случае и выбрать уровень во втором. Пользователь может выбрать один из уровней, которые контролируются скриптом проверки условия победы/поражения. Игровые сцены состоят из набора шаблонов (.prefab), которые в свою очередь состоят из компонентов Unity и специально созданных скриптов для добавления свойств и изменения поведения игровых объектов. Компонентами Unity в данном случае являются спрайты (зависящие от текстур) и аниматоры (.controller), которые управляют анимациями (.anim). Анимации также зависят от текстур, которые являются спрайт-листом и состоят из набора кадров.

# Тестирование приложения. Подготовка документации

На этапе тестирования выполняется модульное, интеграционное и системное тестирование. Также проводится отладка приложения в случае обнаружения ошибок. Выполняется подготовка документации, куда входит техническое задание.

* 1. Документация

Любой программный продукт должен содержать некоторый набор документации, описывающий спецификацию продукта. Наиболее важным документом является техническое задание, которое приведено в приложении Приложение A и оформлено в соответствии с ГОСТ 19.201-78.

* 1. Тестирование

Как было выяснено, главным объектом игры является червяк, следовательно, в первую очередь необходимо проверить работоспособность базовых возможностей червяка, в которые входят передвижение, прыжок, наведение прицела, увеличение силы выстрела и запуск снаряда. Тесты, проверяющие данные условия, указаны в табл. таблица 5.1.

***Таблица 5.1. Тестирование базовых возможностей червяка***

| Тест | Вход | Ожидаемый результат | Реальный результат | +/- |
| --- | --- | --- | --- | --- |
|  | Пользователь нажимает стрелки вправо, влево. | Червяк ползет влево при нажатии левой кнопки и вправо при нажатии правой кнопки, включается анимация передвижения. | Червяк ползет влево при нажатии левой кнопки и вправо при нажатии правой кнопки, включается анимация передвижения. | + |
|  | Пользователь нажимает клавишу пробел. | Червяк прыгает вверх, если стоит и вперед, если ползет, включается анимация прыжка. | Червяк прыгает вверх, если стоит и вперед, если ползет, включается анимация прыжка. | + |
|  | Пользователь нажимает стрелки вверх, вниз. | Прицел червяка увеличивает угол при нажатии клавиши вверх и уменьшает при нажатии клавиши вниз. | Прицел червяка увеличивает угол при нажатии клавиши вверх и уменьшает при нажатии клавиши вниз. | + |
|  | Пользователь зажимает клавишу Ctrl. | Сила выстрела увеличивается в зависимости от времени нажатия. | Сила выстрела увеличивается в зависимости от времени нажатия. | + |
|  | Пользователь отпускает клавишу Ctrl | Запускается снаряд в указанном направлении и с указанной силой выстрела. | Запускается снаряд в указанном направлении и с указанной силой выстрела. | + |
|  | Сила выстрела достигла максимального значения. | Запускается снаряд в указанном направлении и с максимальной силой выстрела. | Запускается снаряд в указанном направлении и с максимальной силой выстрела. | + |

После запуска снаряда игровой механикой предусмотрено, что червяк сделал ход, тогда право хода должно передаться следующему червяку. Также есть ситуация, когда ход передается раньше, например, если червяк упал в воду. Тестирование смены хода приведено в табл. таблица 5.2.

***Таблица 5.2. Тестирование смены хода***

| Тест | Вход | Ожидаемый результат | Реальный результат | +/- |
| --- | --- | --- | --- | --- |
|  | Червяк сделал выстрел. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | + |
|  | Червяк упал в воду. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | + |
|  | Червяк вышел за пределы сцены. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | + |
|  | У червяка кончились очки здоровья. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | Активным становится другой червяк, теперь пользователь управляет другим червяком. | + |

В игре часто возникают ситуации, когда необходимо удалить из сцены объект, который больше не управляется игрой, или он должен быть уничтожен по правилам игровой механики. К этим объектам относятся червяки и снаряды. Тесты уничтожения объектов показаны в табл. таблица 5.3.

***Таблица 5.3. Тестирование уничтожения игровых объектов***

| Тест | Вход | Ожидаемый результат | Реальный результат | +/- |
| --- | --- | --- | --- | --- |
|  | Снаряд столкнулся с ландшафтом. | Игровой объект «Снаряд» удаляется со сцены, проигрывается анимация взрыва. | Игровой объект «Снаряд» удаляется со сцены, проигрывается анимация взрыва. | + |
|  | Снаряд столкнулся с червяком. | Игровой объект «Снаряд» удаляется со сцены, проигрывается анимация взрыва. Червяк теряет очки жизни. | Игровой объект «Снаряд» удаляется со сцены, проигрывается анимация взрыва. Червяк теряет очки жизни. | + |
|  | Снаряд упал в воду. | Игровой объект «Снаряд» удаляется со сцены. | Игровой объект «Снаряд» удаляется со сцены. | + |
|  | Снаряд вылетел за пределы сцены. | Игровой объект «Снаряд» удаляется со сцены. | Игровой объект «Снаряд» удаляется со сцены. | + |
|  | Червяк упал в воду. | Игровой объект «Червяк» удаляется со сцены. | Игровой объект «Червяк» удаляется со сцены. | + |
|  | Червяк вышел за пределы сцены. | Игровой объект «Червяк» удаляется со сцены. | Игровой объект «Червяк» удаляется со сцены. | + |
|  | У червяка кончились очки здоровья. | Игровой объект «Червяк» удаляется со сцены. | Игровой объект «Червяк» удаляется со сцены. | + |

Следует также провести тестирование игровых механик, которыми управляет Unity. К ним относятся механика физики, проверка на столкновения и анимация. Тесты перечисленных механик приводятся в табл. таблица 5.4.

***Таблица 5.4. Тестирование механик Unity***

| Тест | Вход | Ожидаемый результат | Реальный результат | +/- |
| --- | --- | --- | --- | --- |
|  | Динамичный игровой объект находится в воздухе. | Объект падает вниз и увеличивает скорость падения. | Объект падает вниз и увеличивает скорость падения. | + |
|  | Статичный объект находится в воздухе. | Скорость объекта по всем направлениям равна нулю. | Скорость объекта по всем направлениям равна нулю. | + |
|  | Два объекта-коллайдера столкнулись друг с другом. | Объекты действуют друг на друга с имеющейся силой, объекты приобретают импульс. | Объекты действуют друг на друга с имеющейся силой, объекты приобретают импульс. | + |
|  | Два объекта, хотя бы один из которых не является коллайдером, столкнулись друг с другом. | Объекты никак не взаимодействуют друг с другом. | Объекты никак не взаимодействуют друг с другом. | + |
|  | Проигрывание анимации. | Соответствующая анимация проигрывается при соблюдении необходимых условий игровой механики. | Соответствующая анимация проигрывается при соблюдении необходимых условий игровой механики. | + |

Наконец, необходимо проверить механику смены сцен друг другом. Сюда входит переход из состояния главного экрана в состояние меню, переход из меню к первой или второй сцене и переход из игрового процесса в меню (механика проверки условия победы/поражения). Необходимые тесты показаны в табл. таблица 5.5.

***Таблица 5.5. Тестирование смены состояний игры***

| Тест | Вход | Ожидаемый результат | Реальный результат | +/- |
| --- | --- | --- | --- | --- |
|  | Пользователь нажал клавишу мыши на главном экране. | Игра переходит в состояние меню. | Игра переходит в состояние меню. | + |
|  | Пользователь выбрал первую сцену в меню. | Игра переходит в состояние игрового процесса и открывает первую сцену. | Игра переходит в состояние игрового процесса и открывает первую сцену. | + |
|  | Пользователь выбрал вторую сцену в меню. | Игра переходит в состояние игрового процесса и открывает вторую сцену. | Игра переходит в состояние игрового процесса и открывает вторую сцену. | + |
|  | Достигнуто условие победы/поражения (на сцене осталось не более одного червяка). | Появляется сообщение «Game over», игра переходит в состояние меню. | Появляется сообщение «Game over», игра переходит в состояние меню. | + |

Таким образом, проведено тестирование всех игровых механик, выявленных на этапе анализа. Были проверены функциональные требования, определенные в диаграмме прецедентов и диаграмме активностей, протестированы последовательности действий, обозначенных в диаграмме последовательностей.

Заключение

В работе описаны этапы разработки игровых механик на примере игры Worms средствами платформы создания игр Unity.

На первом этапе проведен анализ предметной области и разработаны требования к системе. Проведен обзор и анализ аналога с выявлением достоинств и недостатков. На его основе написан сценарий игры и выявлены основные игровые механики. Также проведен анализ инструментария Unity.

На этапе формализации построена математическая модель игры, описаны алгоритмы работы основных механик, применяемых к системе, с оценкой сложности и ограничениями, указаны, достоинства и недостатки.

На этапе проектирования была описана архитектура системы, спроектирован пользовательский интерфейс, описано поведение игры при переходе из одного состояния в другое.

На этапе реализации программной системы был выбран инструмент разработки, описан процесс разработки с указанием принципа действия реализованных функций, построена диаграмма компонентов.

На последнем этапе разработан набор тестов, проведено тестирование приложения, подготовлена программная документация.

Таким образом были выполнены все задачи, обозначенные в начале работы. Выполнение всех задач подразумевает достижение цели, а именно разработка игровых механик на примере игры Worms средствами Unity. Предложен вариант решения проблемы, связанной с реализацией игровой модели с помощью современных инструментальных средств.

Библиографический список

x

x

|  |
| --- |
| 1. Adams E., Dormans J. Game Mechanics, Advanced Game Design. Berkeley: New Riders Games, 2012. |
| 1. Haller J., Hansson H.V., Moreira A. SFML Game Development. Birmingham: Packt Publishing Ltd., 2013. |
| 1. Barbier M. SFML Blueprints. Birmingham: Packt Publishing Ltd., 2015. |
| 1. Ву М., Девис Т., Нейдер Д., Шрайнер Д. OpenGL. Руководство по программированию. Библиотека программиста. 4-е издание. СПб: Питер, 2006. |
| 1. Unity - Руководство: игровые объекты [Электронный ресурс] // Руководство Unity: [сайт]. [2015]. URL: https:/​/​docs.unity3d.com/​ru/​500/​Manual/​GameObjects.html (дата обращения: 18.02.2017). |
| 1. Mitchell S.R. SDL Game Development. Birmingham: Packt Publishing Ltd, 2013. |
| 1. TexturePacker Tutorials [Электронный ресурс] // Code and Web. Tools for game developers: [сайт]. [2009]. URL: https:/​/​www.codeandweb.com/​texturepacker/​tutorials (дата обращения: 09.03.2017). |
| 1. Физический движок: взгляд изнутри. Часть 1 [Электронный ресурс] // Tproger: [сайт]. [2015]. URL: https:/​/​tproger.ru/​translations/​whats-in-a-projectile-physics-engine-part-1/ (дата обращения: 12.03.2017). |
| 1. Separating Axis Theorem (SAT) Explanation [Электронный ресурс] // sevenson.com.au: [сайт]. [2009]. URL: http:/​/​www.sevenson.com.au/​actionscript/​sat/ (дата обращения: 14.03.2017). |

1. Техническое задание

**УТВЕРЖДЕНО**

**разработка компьютерной игры «worms»**

**Техническое задание**

**Инв. № подл.**

**Подпись и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подпись и дата**

2017

Содержание

1. Введение 46

1.1. Наименование программы 46

1.2. Краткая характеристика области применения программы 46

2. Основание для разработки 46

2.1. Документы, на основании которых ведется разработка 46

2.2. Организация, утвердившая эти документы и дата их утверждения 46

2.3. Наименование и (или) условное обозначение темы разработки 46

3. Назначение разработки 46

3.1. Функциональное назначение программы 47

3.2. Эксплуатационное назначение разработки 47

4. Требования к программе 47

4.1. Требования к функциональным характеристикам программы 47

4.1.1. Требования к составу выполняемых функций 47

4.1.2. Требования к организации входных и выходных данных 48

4.1.3. Требования к временным характеристикам 48

4.2. Требования к надежности 48

4.2.1. Требования к обеспечению надежного функционирования программы 48

4.2.2. Требования к обеспечению устойчивого функционирования 48

4.3. Условия эксплуатации программы 48

4.3.1. Виды обслуживания программы 48

4.3.2. Необходимое количество и квалификация персонала 49

4.4. Требования к составу и параметрам технических средств 49

4.5. Требования к информационной и программной совместимости 49

4.5.1. Требования к информационным структурам на входе и выходе 49

4.5.2. Требования к методам решения 49

4.5.3. Требования к исходным кодам 49

4.5.4. Требования к языкам программирования 49

4.5.5. Требования к программным средствам, используемым программой 49

4.6. Требования к маркировке и упаковке 50

4.7. Специальные требования 50

5. Требования к программной документации 50

5.1. Предварительный состав программной документации 50

6. Технико-экономические показатели 50

6.1. Ориентировочная экономическая эффективность программы 50

6.2. Предполагаемая годовая потребность в программе 50

7. Стадии и этапы разработки 50

7.1. Стадии разработки 50

7.2. Содержание работ по этапам 51

7.3. Сроки и исполнители разработки 51

8. Порядок контроля и приемки 52

8.1. Виды испытаний программы 52

8.2. Общие требования к приемке работы 52

1. Введение
   1. Наименование программы

Наименование программы – компьютерная игра Worms.

* 1. Краткая характеристика области применения программы

Программа предназначена к применению в профильных подразделениях НИУ ВШЭ в Перми.

1. Основание для разработки
   1. Документы, на основании которых ведется разработка

Основанием для проведения разработки является Договор №1 от 01.10.2016.

Договор согласован со старшим преподавателем кафедры информационных технологий в бизнесе НИУ ВШЭ Лебедевым В.В. и студентом 3-го курса факультета экономики, менеджмента и бизнес-информатики Щелкуновым А.А., 25.12.2016.

Договор утвержден менеджером образовательной программы «Программная инженерия» НИУ ВШЭ Гордеевой О.И., 01.03.2017.

* 1. Организация, утвердившая эти документы и дата их утверждения

НИУ ВШЭ – Пермь, 01.03.2017.

* 1. Наименование и (или) условное обозначение темы разработки

Наименование темы разработки – «Разработка игры Worms».

1. Назначение разработки

Worms – классическая игра жанра стратегия и артиллерия. В распоряжение игрока поступает команда червяков, имеющих определенный набор вооружения. Червяки располагаются на разрушаемом ландшафте. Стреляя из выбранного оружия, они должны уничтожить команду противника и самим остаться в живых, используя определенную тактику.

* 1. Функциональное назначение программы

Функциональным назначением программы является демонстрация игровых механик, которые можно разработать современными инструментальными средствами.

* 1. Эксплуатационное назначение разработки

Программа должна эксплуатироваться в профильных подразделениях НИУ ВШЭ. Пользователи программы должны стать сотрудники профильных подразделений НИУ ВШЭ.

1. Требования к программе
   1. Требования к функциональным характеристикам программы

Система должна обладать следующими характеристиками:

1. Надежность.
2. Сопровождаемость.
3. Удобство применения.
4. Эффективность.
5. Универсальность.
6. Корректность.

Базовые показатели данных критериев качества описаны в «Методике оценки качества».

* + 1. Требования к составу выполняемых функций

Программа должна обладать функциональными характеристиками, которые предполагают реализацию игровых механик Worms:

1. Обработка столкновений.
2. Взаимодействие объектов.
3. Физика.
4. Анимация.
5. Передвижение червяков.
6. Стрельба.
7. Нанесение урона.
8. Контроль смены очередности ходов.
9. Тактика.
10. Условия победы/поражения.
    * 1. Требования к организации входных и выходных данных

Источниками входных данных являются пользовательские команды, передаваемые посредством устройств ввода (клавиатуры, мыши).

Выходными данными являются состояния игры, изображенные в виде сцены с игровыми объектами.

* + 1. Требования к временным характеристикам

Ответ программы на любое действие пользователя должен происходить не дольше, чем за одну секунду.

* 1. Требования к надежности

Требования к надежности описаны в «Методике оценки качества».

* + 1. Требования к обеспечению надежного функционирования программы

Рекомендуемый базовый показатель надежности равен 0,9 (по шкале оценки критерия надежности из «ГОСТа 28195-89 – Оценка качества программных средств»).

* + 1. Требования к обеспечению устойчивого функционирования

Рекомендуемый базовый показатель надежности равен 0,9 (по шкале оценки критерия надежности из «ГОСТа 28195-89 – Оценка качества программных средств»).

* 1. Условия эксплуатации программы

Программа должна эксплуатироваться на персональном компьютере под управлением операционной системы Windows 7.

* + 1. Виды обслуживания программы

Программа не нуждается в обслуживании.

* + 1. Необходимое количество и квалификация персонала

Количество игроков равно количеству команд, которые позволяет создать Worms во время игры (в данном случае два игрока). Требования к квалификации персонала не предъявляются.

* 1. Требования к составу и параметрам технических средств

В состав технических средств должны входить:

1. IBM-совместимый компьютер.
2. Процессор Pentium 4 или выше с тактовой частотой не ниже 1.5 ГГц.
3. Оперативная память объемом не менее 1 Гб.
4. Свободное место на жестком диске в объеме 10 Мб.
5. Клавиатура, мышь.
   1. Требования к информационной и программной совместимости

Программа должна быть совместима с аппаратурой НИУ ВШЭ Пермь.

* + 1. Требования к информационным структурам на входе и выходе

Требования к информационным структурам не предъявляются.

* + 1. Требования к методам решения

Программа должна реализовывать методы решения, предлагаемые инструментарием Unity.

* + 1. Требования к исходным кодам

Требования к исходным кодам не предъявляются.

* + 1. Требования к языкам программирования

Целевой язык программирования – C#.

* + 1. Требования к программным средствам, используемым программой

Программа должна использовать программные средства, предоставляемые инструментарием Unity.

* 1. Требования к маркировке и упаковке

Программа предоставляется в виде .exe файла.

* 1. Специальные требования

Программа должна обеспечивать взаимодействие с пользователем, посредством интерфейса, разработанного с помощью инструментария Unity.

1. Требования к программной документации
   1. Предварительный состав программной документации

Состав программно документации должен включать:

1. Техническое задание.
2. Руководство пользователя.
3. Руководство программиста.
4. Программа и методика испытаний, тесты.
5. Технико-экономические показатели
   1. Ориентировочная экономическая эффективность программы

Ориентировочная экономическая эффективность не вычисляется.

* 1. Предполагаемая годовая потребность в программе

Предполагаемая годовая потребность в программе – один раз во время защиты курсовой работы.

1. Стадии и этапы разработки
   1. Стадии разработки

Этапы разработки должны включать:

1. Анализ задачи и разработка требований к системе.
2. Формализация описания решения задачи. Выбор и/или разработка алгоритмов для реализации системы.
3. Проектирование приложения.
4. Реализация программной системы.
5. Тестирование и отладка приложения. Подготовка документации.
   1. Содержание работ по этапам

На этапе анализа изучается информация о предметной области, формулируется набор требований, описываются бизнес‑процессы, анализируются современные технологии разработки компьютерных игр, рассматриваются аналоги с выявлением достоинств и недостатков, описывается сценарий игры и выявляется набор механик, которые составляют геймплей игры.

На втором этапе работы на основе сценария строится математическая модель с выделением геометрической модели мира, сцен, игровых объектов и способами их взаимодействия. Также осуществляется выбор и приводится описание алгоритмов реализации, в данном случае игровых механик, которые управляют поведением игры.

На стадии проектирования описывается архитектура системы с рассмотрением отдельных компонентов, классов и их поведения, строится диаграмма классов UML. Проектируются рассмотренные на этапе анализа механики. Затем выполняется проектирование графического интерфейса с учетом решаемых системой задач.

На этапе реализации описывается процесс перевода формальной модели в программную с точки зрения игровых объектов. С учетом описанных требований в систему интегрируются существующие алгоритмы реализации необходимых механик. Разрабатываются сцены и создается набор ресурсов (изображения, анимации, звуки, карты, шрифты, скрипты и т.д.), используемых при реализации прототипа.

Этап тестирования и отладки предназначен для проверки надежности разрабатываемой системы. Составляется набор тестов, проверяющих качество программы и разрабатывается программная документация.

* 1. Сроки и исполнители разработки

Началом срока разработки считается момент согласования темы работы с научным руководителем (01.10.2016). Окончанием срока разработки считается дата сдачи отчета научному руководителю и демонстрация программной системы (29.03.2017). Исполнителем разработки является студент 3-го курса направления «Программная инженерия» НИУ ВШЭ Пермь, Щелкунов А.А.

1. Порядок контроля и приемки
   1. Виды испытаний программы

Испытания проводятся согласно разработанным методике оценки качества и плану испытаний. Тестирование проводится на основе разработанного набора тестов.

* 1. Общие требования к приемке работы

Отчет о работе и программная система передается научному руководителю не позднее семи дней до защиты курсовой работы. Не позднее четырех дней текст работы проходит проверку в системе «Антиплагиат». Не позднее одного дня научный руководитель пишет отзыв о проделанной работе, текст работы с отзывом и результатами проверки системы «Антиплагиат» сдается в учебный офис.