ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ
КАФЕДРА ОБЩЕЙ МАТЕМАТИКИ

ГЕОМЕТРИЧЕСКОЕ МОДЕЛИРОВАНИЕ ПРИ СОЗДАНИИ ИГРОВОГО ПРИЛОЖЕНИЯ ДЛЯ ОС WINDOWS

Выпускная квалификационная работа
обучающегося по направлению подготовки
01.03.02 Прикладная математика и информатика
очной формы обучения, группы 12001510
Моралес Прохорова Алексея Франсисковича

Научный руководитель
к. ф.-м. н.
Куртова Л.Н.

БЕЛГОРОД 2019

Оглавление

Введение

Компьютерные и видеоигры к нашему времени обрели широкую популярность. Говоря о развлекательных видеоиграх, можно отметить, что оборот индустрии вырос в 2018 году до 135 миллиардов долларов.

Этим достижением игры обязаны нескольким факторам. Во-первых, благодаря быстрому технологическому развитию всего человечества, регулярно появляются новые платформы для игр, такие, как планшеты, смартфоны, и даже умные часы. Благодаря этому, игры как средство самовыражения становятся всё доступней для самых больших слоёв населения.

Во-вторых, сами по себе игры являются крайне гибким инструментом. Так, существуют не только развлекательные, но и научно-исследовательские, образовательные, даже медицинские проекты, например, системы реабилитации для больных, страдающих церебральным параличом, или перенёсших инсульт.

В-третьих, с появлением простых для понимания и, к тому же, бесплатных инструментов создания проектов, например, RPG Maker, Unity или Unreal Engine, разработка игры становится доступной для пользователей даже без технического образования. Разработчики же этого инструментария нередко снабжают свою продукцию сериями обучающих видео или текстовых курсов, позволяющих любому желающему детально разобраться в процессе создания видеоигры.

Соответственно, цель данной работы – разработка игрового приложения с использованием методов геометрического моделирования.

Для разработки был выбран игровой жанр roguelike. Это один из самых распространённых жанров видеоигр и при этом простой для освоения. Благодаря некоторым характерным особенностям, проекты в этом жанре поощряют исследовательский интерес пользователя, а также весьма насыщены действием.

Для достижения поставленной цели в работе были решены следующие задачи:

- выбрана инструментальная среда, в которой будет вестись разработка;
- изучены методы геометрического моделирования;
- с помощью выбранной инструментальной среды разработано приложение для платформы ПК;
- приложение прошло тестирование;

Структура и объем работы: выпускная квалификационная работа выполнена на страницах машинописного текста, состоит из введения, четырех глав, заключения и приложения.

В первой главе дана характеристика жанра roguelike, изучены различные инструментальные среды для реализации проекта, а также рассмотрены методы геометрического моделирования и геометрические модели.

Во второй главе рассматриваются специфика работы в выбранной инструментальной среде, и архитектура созданного приложения.

В третьей главе проводится тестирование созданного программного продукта.

Глава 1. Проектирование программного продукта

1.1 Основные черты жанра Roguelike

Игровой процесс представителей жанра Roguelike может значительно отличаться. Интернет-магазин видеоигр Steam, один из крупнейших существующих, предлагает покупателю в качестве вариантов Roguelike такие игры, как FTL: Faster than light, Dwarf Fortress, The Binding of Isaac и Don’t Starve, радикально отличающиеся количеством контролируемых персонажей, положением камеры, динамикой и течением времени в игровом процессе, боевыми системами, и так далее.

Рассмотрим несколько популярных представителей жанра, чтобы выяснить определяющие жанровые черты.

1.1.1 FTL: Faster than light

В FTL игрок контролирует команду боевого корабля, вынужденного путешествовать из одной процедурно генерируемой солнечной системы в другую ограниченное количество раз. Под процедурной генерацией подразумевается составление уровня по созданному разработчиками алгоритму, из заранее подготовленного списка компонентов.

Контроль команды заключается в распределении обязанностей между членами команды, как то – один из космонавтов должен управлять вооружением корабля, другой – управлять двигатели, третий – сражаться и ремонтировать повреждения обшивки корабля.

При уничтожении корабля, игра заканчивается, причём как-то восстановить жизни членов команды, корабль, либо какое-то ещё накопленное во время прохождения имущество невозможно.

Игру обязательно каждый раз начинать заново. Во время каждого прохождения дополнительные задания, выдаваемые игроку при встрече с не враждебно настроенными персонажами, а также точки встреч с этими персонажами, генерируются случайно.

Тем не менее, игровую сессию (т.е. единожды начатое прохождение, в ходе которого игрок накапливает сохраняемые только в рамках этого прохождения ресурсы и действует в границах существующего только во время этого прохождения мира) возможно продолжить, если просто выйти из игры самостоятельно, отложив её завершение до следующего посещения игры.

Также, в игре есть ограниченная связь между различными игровыми сессиями. Например, благодаря каким-либо совершенным в игре действиям, можно открыть различные корабли с различным экипажем, на которых можно начать игровую сессию. Это добавляет в игру больше вариативности, и позволяет игроку проходить игру в зависимости от его предпочтений.

Рисунок 1 – Экран битвы в FTL.

Условные обозначения:

1. Список членов команды
2. Схема корабля для управления перемещением членов команды
3. Изображение корабля противника
4. Панель вооружения корабля

1.1.2 The Binding of Isaac

В The Binding of Isaac игрок контролирует только одного персонажа, а задача игры заключается в прохождении через n-ное количество процедурно генерируемых боевых арен, и уничтожении на этих аренах всех противников. Дизайн игры соблюдает такой же подход к игровым сессиям – в рамках одной

сессии смерть игрока перманентна, а все предметы и вооружение, появляющиеся на уровнях, определяются случайным образом из набора заранее созданных разработчиком.

Важное отличие The Binding of Isaac в том, что игра сохраняет большую связь между сессиями, чем Faster than light. Во время сессий игрок уничтожает рядовых противников, боссов, открывает разнообразные секретные локации.

Победа над определёнными противниками и нахождение определённых секретов несколько изменяют подход игры к генерации уровней. Разблокируются новые предметы, новые типы уровней, а само максимальное количество уровней увеличивается. Игра генерирует на уровнях также более сложных противников, какие-либо уникальные события, либо вообще открывает доступ к сложнодоступным элементам игры.

Таким образом, существует сохраняемый прогресс между различными игровыми сессиями, и, несмотря на единоразовость игровых сессий, игра создаёт ощущение прогресса в её освоении.

Рисунок 2 – Экран битвы в The Binding of Isaac

Условные обозначения:

1 – Персонаж игрока

2 – Противники

3 – Количество очков здоровья игрока

4 – Счётчик ресурсов

1.1.3 Don’t Starve

В Don’t Starve мы также, как и в The Binding of Isaac, мы контролируем одного игрового персонажа, помещённого в процедурно сгенерированный мир, но в данном случае всё же достаточно много отличий от предыдущих представителей жанра.

Во-первых, мир не разбит на арены, как в FTL и The Binding of Isaac, а един в рамках одной игровой сессии. Во-вторых, персонаж игрока должен не только сражаться с противниками, но и, обеспечивать собственное выживание – собирать ресурсы, строить для себя жилище, добывать продукты и готовить из них пищу, и так далее. Во-третьих, у игры, в зависимости от режима, есть вариант «бесконечной сессии», т.е. такой, которая закончится только тогда, когда игроку надоест, и он сам забросит игру. До тех пор в мире можно действовать столько, сколько угодно игроку. Даже перманентная смерть в Don’t Starve опциональна, хотя и присутствует как вариант.

Рисунок 3 – Don’t Starve

1.1.4 Nuclear Throne

В Nuclear Throne мы также контролируем одного персонажа, который должен пройти через ряд процедурно сгенерированных уровней для прохождения игры. Как и в прочих играх, можно выбирать из ряда персонажей с различными способностями. Точно так же, как и The Binding of Isaac, данная игра является абсолютно классическим образцом roguelike. Помимо процедурно сгенерированных уровней, в игре присутствуют перманентная смерть, а оружие, состав противников на уровне, и появляющееся снаряжение выбираются случайно из списка заранее созданных образцов.

Отличительной чертой данной игры является возможность условно бесконечной сессии. В отличие от Don’t Starve, данная возможность реализована посредством закольцовывания определённого набора уровней. Т.е. на строго определённом уровне игрока всегда будет ждать наиболее сильный противник, после которого прохождение снова начнётся с первого уровня, но персонаж сохранит все собранные за прохождение ресурсы.

Рисунок 4 – Nuclear Throne

Условные обозначения:

1 – Очки здоровья персонажа

2 – Противники

3 – Активированные за сессию модификаторы

Таким образом, на основании рассмотрения нескольких представителей жанра, а также на основании изученных материалов, можно сделать вывод, что чаще всего для Roguelike-игр характерны две черты.

1. Процедурно генерируемые уровни.
2. Перманентная смерть подконтрольного игроку персонажа.

Остановимся на каждом из этих пунктов подробнее.

Вне зависимости от поджанра игры, идёт ли речь о стратегии, как в случае с Dwarf Fortress, или о космическом симуляторе, как в случае с FTL, уровни, встречи со многими персонажами и большая доступного во время сессии контента генерируется процедурно при начале каждой новой сессии. Для реализации данного подхода в играх используется множество различных алгоритмов, например, BSP–деревья, алгоритмы туннелирования, если речь идёт об уровнях. Использование процедурной генерации для моделирования геометрии уровня будет описано ниже, в главе 1.2.

Разные игры используют разные подходы к игровым сессиям, иногда даже сохраняя связь вне сессий. Тем не менее, случайность сгенерированного для каждого прохождения мира – обязательное условие Roguelike.

Перманентная смерть же встречается не в каждом проекте, но, тем не менее, попадается настолько часто, что, согласно некоторым источникам, её всё же считают одной из характерных особенностей жанра.

Сочетание этих черт позволяет делать игровой опыт уникальным в рамках разных игровых сессий, а способность жанра принимать элементы формы других жанров, будь то ролевая игра, аркадный шутер, симулятор, или какая-либо разновидность видеоигр, позволяет представителям Roguelike значительно отличаться друг от друга во множестве аспектов, при этом, сохраняя главную черту – уникальность каждой игровой сессии.

1.2 Выбор среды разработки

Перед созданием проекта разработчик должен определиться с тем, какой инструментарий будет использоваться. В случае данного проекта, выбор осуществлялся из нескольких игровых движков, которые могут взаимодействовать с различными средами разработки. Рассмотрим эти игровые движки.

1.2.1 Unity

Игровой движок Unity получил достаточно широкие освещение в тематических СМИ и популярность ещё в конце 2000х, а к моменту написания данной работы стал одним из самых распространённых на рынке. Тому есть несколько причин:

1. Количество документации

   Создатели движка опубликовали множество бесплатных видеокурсов для начинающих разработчиков. Данные материалы затрагивают большой спектр разнообразных тем, начиная от работы с 3д-моделями в движке как таковыми или присоединением к проекту готовых ресурсов, и заканчивая разработкой простых игр на базе движка. Один из курсов, созданных разработчиками Unity, затрагивает в том числе необходимую для данной работы тему процедурной генерации уровней.

   Следует отметить, что бесплатные курсы актуальны только для устаревших версий движка, а потому обучающимся приходится потратить некоторое время, чтобы приспособить учебные материалы к современной версии ПО.

2. Простота освоения

   Благодаря широкому применению в интерфейсе среды разработки современных решений, таких, как умное подключение одного скрипта к другому посредством перетаскивания иконки одного на иконку другого

мышкой, а также использованию языков C# и JavaScript для написания скриптов, освоение Unity не представляет сложности для знакомых с базовыми понятиями программирования и, в частности, объектно-ориентированного программирования пользователями.

3. Возможности движка

На Unity было разработано множество известных игр, как 2D, так и 3D. Только за 2018-й год на данном ПО были созданы такие получившие широкое признание продукты, как Hollow Knight, Escape from Tarkov, Cuphead, Endless Space 2. Причём данные игры принадлежат самым различным жанрам и платформам. Тем не менее, на Unity не было изготовлено ни одного проекта с по-настоящему крупным бюджетом и большими масштабами разработки, по типу таких известных серий как Mass Effect, Battlefield или Call of Duty. Главная причина тому – недостаточные широкий инструментарий движка для таких задач.

Таким образом, Unity представляет собой современный и востребованный инструмент. Так, на сайте headhunter.ru можно увидеть большое количество вакансий для Unity–разработчиков с зарплатой выше средней по РФ, что только подтверждает актуальность данного ПО на момент написания работы.

Рисунок 5 – Интерфейс среды разработки на Unity

Рисунок 6 – Снимок экрана игры на Unity (Escape From Tarkov, PC)

1.2.2 Unreal Engine

Unreal Engine – один из старейших и наиболее популярных движков, используемых при создании видеоигр. До выхода версии 4 он был полностью платным, но в данный момент, благодаря гибкой системе оплаты, любой начинающий разработчик может получить его бесплатно и использовать для разработки коммерческого приложения.

Рассмотрим преимущества и недостатки движка.

1. Возможности движка.

   На движке было создано большое количество как дорогих и крупномасштабных проектов, так и относительно нишевых. Кроме того, проекты на движке реализовывались как для ОС Android, так и для ОС Windows, Linux и других. Движок считается одним из самых технологичных на данный момент, с поддержкой реализации в играх самых современных технологий, например, Real-Time Ray Tracing.

2. Простота освоения.

   Работа с Unreal Engine требует большей подготовки относительно работы с, например, Unity. Одна из главных причин тому – большие возможности движка относительно остальных представленных в списке, а также в основном использование C++ как языка, взаимодействующего с движком. Порог вхождения в специализированное программирование на C++ в целом выше относительно C# или Java, из–за необходимости регулярной работы с указателями и, как следствия, требовательности к хорошему пониманию программистом принципов работы с памятью компьютера. Соответственно, освоить UE сложнее, чем прочие движки.

3. Количество документации.

   По Unreal Engine 4 существует достаточно большое учебной литературы и материалов на видеохостинге youtube.com, но количество книг на русском языке довольно невелико. Количество книг на английском языке гораздо больше, но многие из них невозможно получить никак иначе, кроме как заказав в специализированных магазинах за немалую цену. Это делает проблематичным обучение выполнению специализированных задач на Unreal Engine.

Рисунок 7 – Интерфейс среды Unreal Engine 4

Рисунок 8 – пример графики, созданной на Unreal Engine (Unreal Tournament, 2016)

1.2.3 GameMaker Studio

GameMaker Studio – мультиплатформенный игровой движок, использующийся в основном для разработки бюджетных камерных проектов, например, платформеров или beat em up–ов. Распространяется частично-бесплатно, с ограниченным функционалом в бесплатной версии. Рассмотрим особенности GameMaker.

1. Простота освоения

   Благодаря собственному языку программирования, GameMaker Language, созданному специально для работы с данным движком, разработку игр может освоить даже пользователь, не разбиравшийся в программировании до знакомства в GameMaker.
2. Возможности движка

   На движке созданы некоторые популярные проекты, такие, как HyperLight Drifter, Hotline Miami. Однако, из–за большой сложности работы с 3D-графикой, на движке не было создано высокобюджетных масштабных проектов, полностью использующих возможности современной 3D-графики.
3. Количество документации

   Создатели движка опубликовали большое количество доступных видеокурсов по изучению движка. Например, на официальном сайте существуют курсы, посвящённые настройке камеры в играх на GameMaker, или созданию системы инвентаря. Как и в случае с Unity, курсы разбиты на несколько уровней сложности, но, в отличие от Unity, уровней сложности только 2: Beginner и Intermediate.

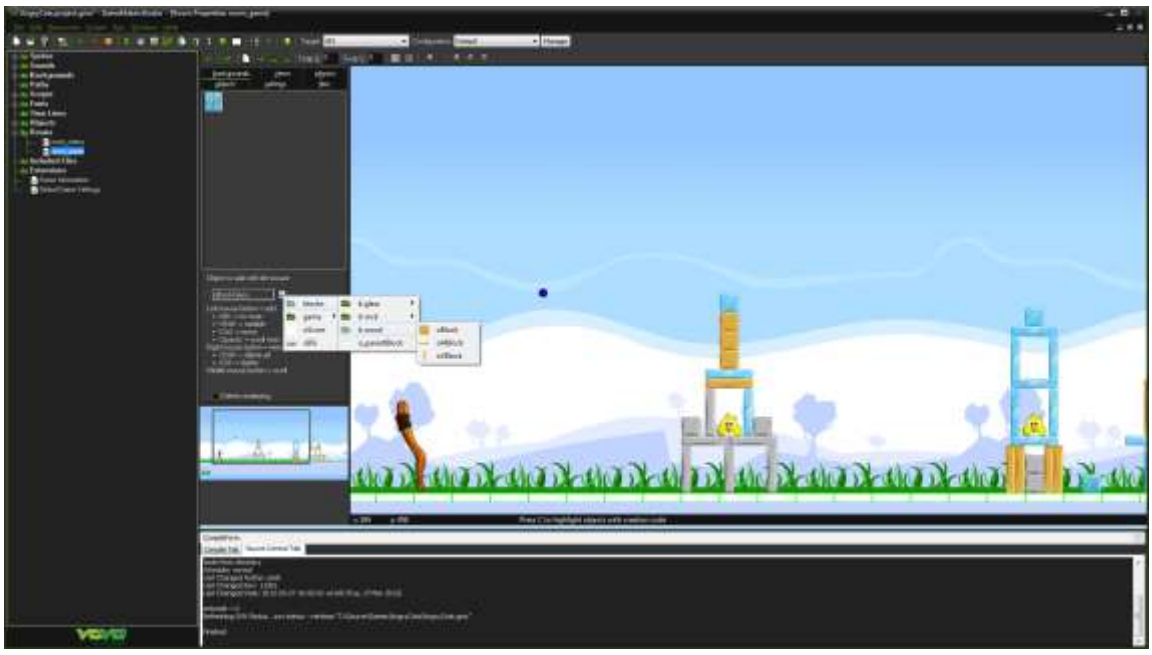

Рисунок 9 – Интерфейс Game Maker Studio

Рисунок 10 – Игра, сделанная с использованием Game Maker Studio (Hotline Miami, 2012)

Таким образом, на основании проведённого анализа был сделан вывод о том, что игровой движок Unity наилучшим образом подходит для создания задуманного программного продукта. Благодаря простоте освоения, базовые навыки работы с движком могут быть получены за сравнительно короткие

сроки, а также, благодаря возможностям движка, реализовать поставленные в рамках квалификационной работы задачи не составило бы труда.

1.3 Объектная модель разрабатываемого приложения.

Объектная модель должна отображать структуру объектов, присутствующих в игре, их свойства, методы их взаимодействия.

В разработанной объектной модели существуют два типа объектов: объекты, относящиеся к уровню (возможные препятствия на игровом поле, ресурсы, прочие), и объекты–персонажи (игрок, противники).

Игрок контролирует персонажа, находящегося на небольшом игровом поле, и способного перемещаться по данному полю в четырёх направлениях, но неспособного покидать его границы. На поле также существуют различные препятствия, реализованные как непроходимые клетки карты.

Поле полностью ограждено непроходимыми клетками, и персонаж игрока не может покинуть уровень, не встав на клетку выхода.

На игровом поле присутствуют подвижные противники, которых игрок должен уничтожить. Когда игрок попадает в поле зрения противника, тот стремится приблизиться к игроку и нанести ему урон.

И игрок, и противники двигаются один раз за ход. Время на принятие игроком решения о направлении передвижения не ограничено, общее количество ходов не ограничено. Ходы противников и игрока никак не пересекаются. С каждым ходом, игрок теряет определённое количество единиц ресурса.

Задача игрока – собрать максимально возможное количество ресурсов на уровне, при этом получить как можно меньше урона, и покинуть уровень, встав на специальную клетку. При этом, ни то, ни другое, не обязательно в рамках одного уровня.

Тем не менее, игра продолжается только до тех пор, пока количество ресурсов игрока больше нуля. Соответственно, хотя в рамках одного уровня

игрок не обязан собирать ресурсы, в рамках игровой сессии это необходимый процесс.

Конечная объектная модель изображена на рисунке .

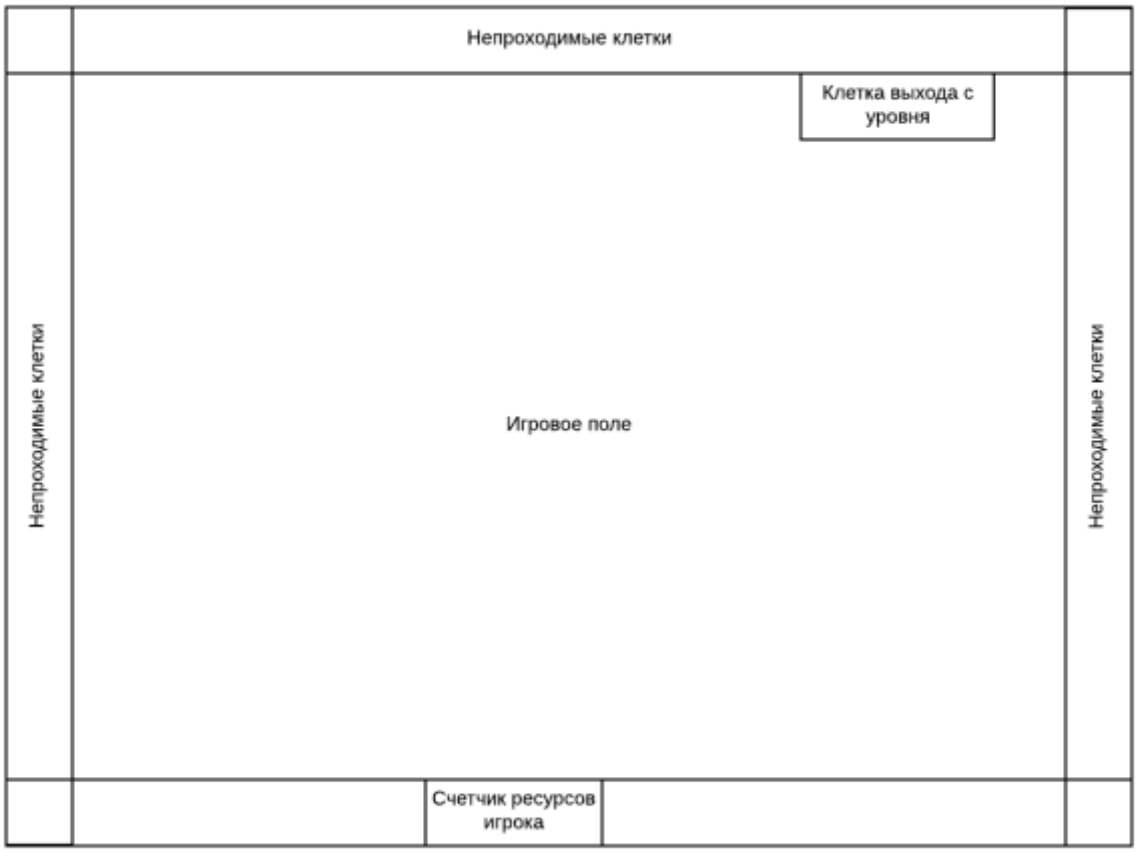

Рисунок 11 – объектная модель игры

1.4 Методы геометрического моделирования.

Для отрисовки графики и позиционирования объектов на уровнях используются различные геометрические модели и методы геометрического моделирования. Рассмотрим различные методы и модели в общем, и, в частности, использованные при разработке приложения.

1.4.1 Геометрические модели

1) Каркасная модель

Модель, представляющая собой скелетное описание объекта. Состоит из вершин и рёбер. Требует немного памяти. Самая простая модель объекта, и, в то же время, наиболее низкоуровневая. Unity, как игровой движок, не поддерживает создание моделей на таком уровне. Каждый из объектов-шаблонов, которые можно использовать при создании проекта (параллелепипеды, шары, цилиндры) уже содержит не только каркасную модель, но и описание информации между гранями. Тем не менее, скелет модели в Unity всегда отображается.

Создание каркасных моделей относится к функционалу редакторов 3D-графики.

Простейшие модели были использованы при подготовке к реализации проекта (т.е. в обучающем курсе Unity).

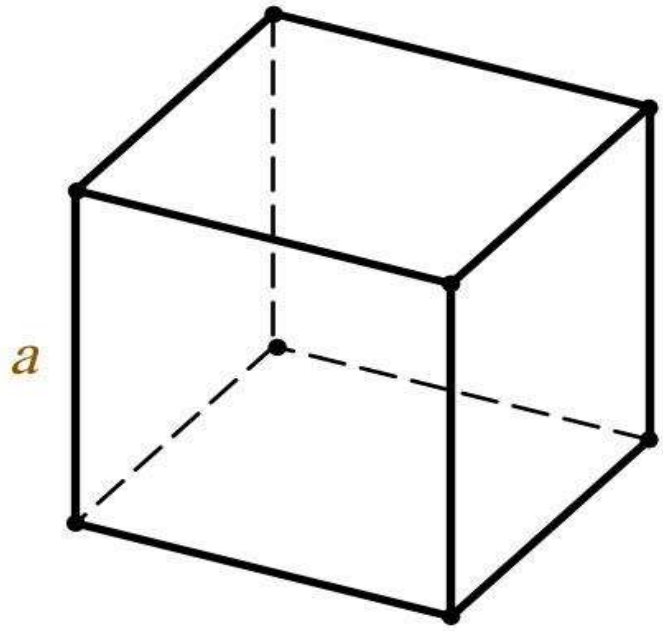

Рисунок 12 – каркасная модель куба

2) Кусочно–аналитическая модель

Математическая модель. Используется при моделировании объектов на низком уровне. Поверхность объектов представляется

отдельными кусками гладких поверхностей. Грани задаются уравнением поверхности и границей грани. Ребра геометрического объекта это линии пересечения поверхностей, ограничивающих ГО.

3) Объемно–параметрическая модель

Объект представляется как совокупность каких–то базовых примитивов (цилиндр, конус, и другие геометрические фигуры), в совокупности представляющих собой сплошное тело.

Примитивы задаются двумя группами параметров.

Размерные параметры определяют геометрические размеры примитива, а параметры положения устанавливают положение и ориентацию относительно системы координат мира.

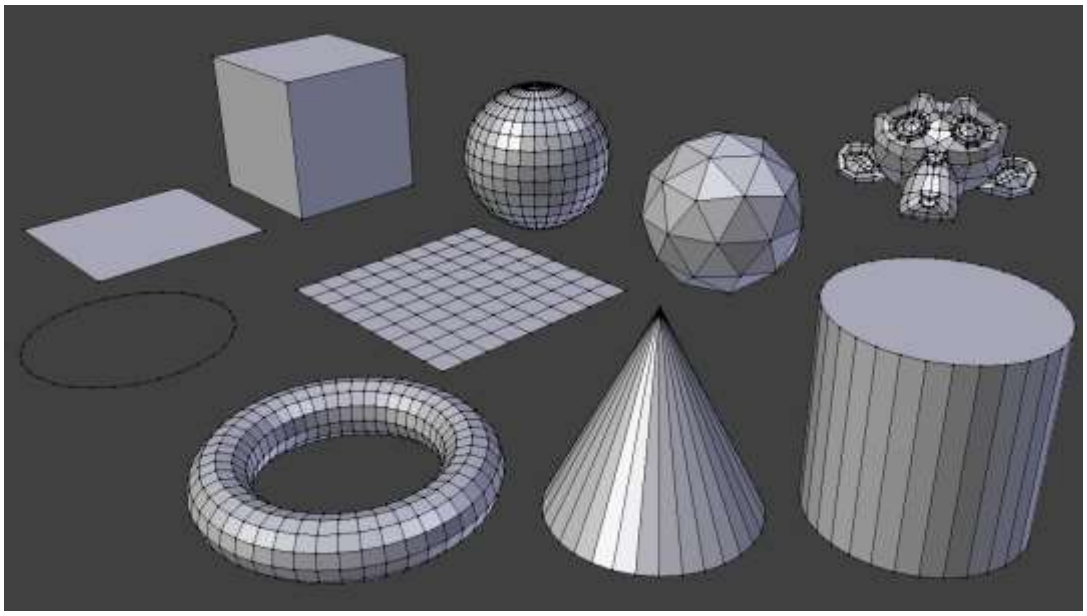

Рисунок 13 – пример объемно–параметрической модели в редакторе Blender 3D.

Unity работает с моделями, экспортированными из различных игровых редакторов, т.е. именно с материалами такого типа.

1.4.2 Методы геометрического моделирования

Рассмотрим также некоторые методы геометрического моделирования, применяемые в 3D-графике.

1) Булевы операции над многоугольниками. Поскольку во многих графических редакторах объекты описываются как набор плоских многоугольников, для операций с объектами на самом низком уровне используется множество различных методов, включая методы, находящиеся на пересечении алгебры логики и геометрии. Примером может служить метод построения пересечений многоугольников.
2) Граничное представление. Представляет собой один из методов описания объемной формы путём описания её границ. Представляет собой, по сути, продолжение идей каркасной модели объекта. Объект представляет собой набор поверхностей, объединённых по замкнутым границам, и образующих объем. Данное представление позволяет определять объем тела, а также использовать модель объекта для задач инженерного плана. Пример использования граничного представления – формула простого многогранника, выражает связь между числом вершин, рёбер и граней: V–T+F=2, удовлетворяет формуле Эйлера.

Следует отметить несколько моментов об использовании указанных 3D–моделей в Unity, и геометрического моделирования в целом.

Как упоминалось выше, в среде разработки отсутствует возможность создания каркасных, либо ещё каких-либо 3D-моделей. Низкополигональное моделирование движок не поддерживает. На выбор пользователю предлагается либо импорт уже заранее созданных моделей, либо использование уже существующих объектов.

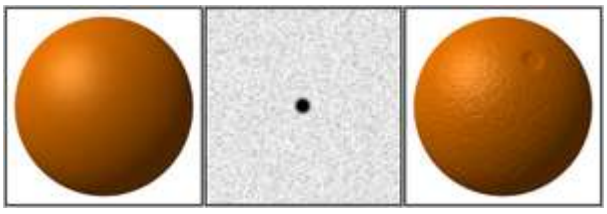

Рисунок 14 – рельефное текстурирование

Во-вторых, в движок уже встроены функции по работе с импортированными или использованными 3D-моделями. Несколько примеров данных функций приведены ниже.

Рельефное текстурирование. Суть данной технологии заключается в освещении поверхностей источником света и одноканальной карты высот, для изменения ориентации нормалей. В результате данной операции, создаётся детализация бугристости поверхностей, за счёт получения множества по–разному освещённых участков.

Рисунок 15 – пример рельефного текстурирования

Помимо этого, движок поддерживает и другие технологии для создания более реалистичной картинки, например, отсечение.

Принцип организации каждой сцены на движке таков.

Существует мировая система координат, с координатами x,y,z. Именно на ней размещаются все объекты в игровой сцене, и именно относительно неё

рассчитывается, например, перемещение различных игровых объектов. Например, заполнение этой системы игровыми клетками в проекте происходит по следующему алгоритму:

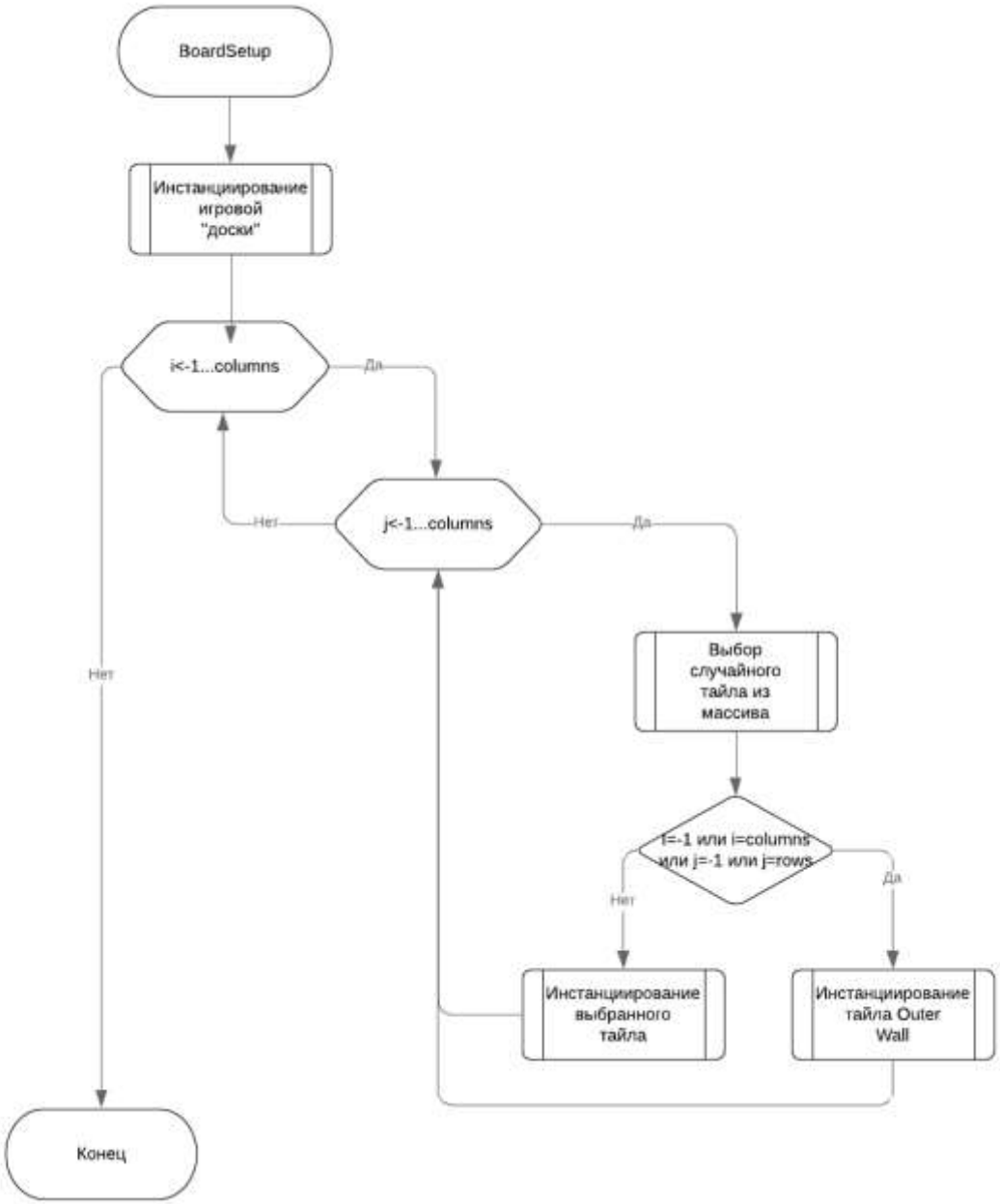

Рисунок 16 – алгоритм заполнения игровой сцены тайлами

Данный алгоритм, помимо генерации определённого количества случайных проходимых и непроходимых клеток, отвечает также за генерацию определённого количества абсолютно непроходимых клеток по периметру арены, чтобы игрок не мог покинуть уровень, кроме как перейдя на клетку выхода.

В алгоритме нет встроенного ограничения на создание какого–то определённого количества непроходимых тайлов, т.е. всё игровое поле можно заполнить непроходимыми тайлами. Тем не менее, алгоритм позволяет исключить такой вариант.

Взаимодействия с сеткой осуществляются с помощью списка элементов класса Vector3. В свою очередь, класс Vector3 нужен для передачи различной информации о позиционировании 3D-объектов в Unity.

В движке также существуют классы Vector2 и Vector4, для работы в системах координат соответствующей размерности.

Наконец, в самом проекте для реализации графики в основном используются спрайты.

Спрайт представляет собой растровое изображение, способное свободно перемещаться по экрану. В трехмерной графике спрайт представляет собой проекцию какого-либо изображения на экран. Отличие спрайта от обыкновенного рисунка в том, что он может включать, например, анимацию, не обязательно должен быть исключительно двумерным, и т.д.

Рисунок 17 – пример атласа спрайтов, используемого для игры (Roguelike tutorial)

В самом проекте используется спрайтовая графика, как созданная самостоятельно, так и под заказ, или найденная в различных источника.

Преимущества спрайтовой графики заключаются в простоте работы с ней, невысоких системных требованиях для игры, полагающейся на спрайтовую графику, поскольку графическому ядру проще обрабатывать такую графику.

Глава 2. Разработка приложения

2.1 Архитектура проекта на движке Unity

При разработке игры на Unity обыкновенно используют следующие понятия.

1) Проект – представляет собой заготовку игры, состоящую из множества различных элементов, список которых будет представлен ниже.
2) Сцена – элемент проекта, представляющий собой отдельный экран игры. Сцены могут содержать главное меню, игровые уровни, и т.д. Проект содержит две сцены.

Рисунок 18

MainMenu содержит главное меню, и элементы, необходимые для его работы.

concept содержит основной игровой экран, логику игрового цикла, ресурсы, т.е. техническую реализацию проекта.

На рисунке 19 изображен жизненный цикл сцены. Рисунок взят из официальной документации Unity, и переведён на русский язык.

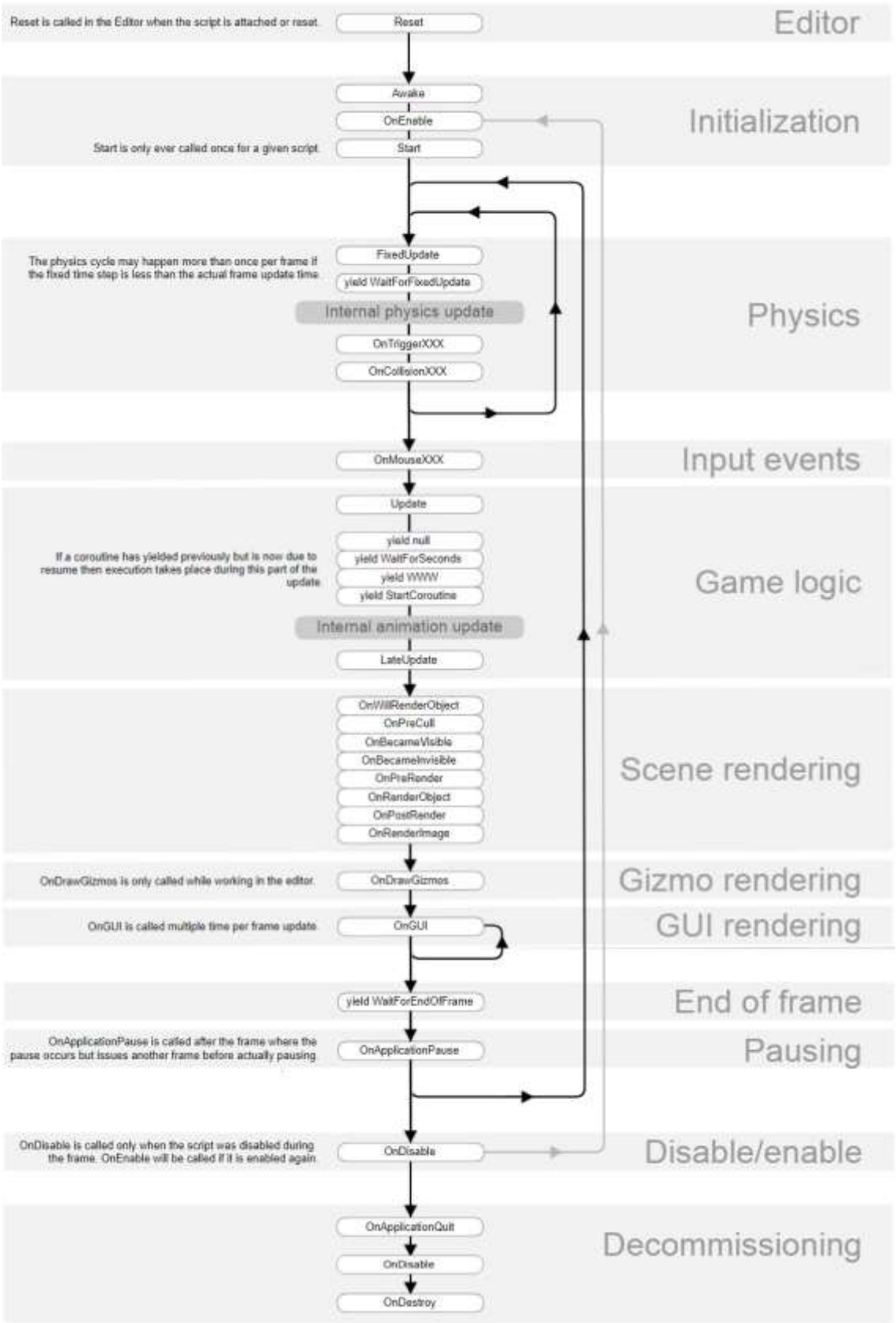

Рисунок 19 – жизненный цикл приложения на движке Unity

3) Ресурсы или ассеты (от англ. asset – ресурс, имущество) – представляют собой данные игры. Программный код, звуковые и графические файлы, текстовые данные, и так далее.
Ресурсы привязаны не к отдельным сценам, а к проекту в целом.

2.2 Элементы сцен

Соответственно, разработка игры на Unity состоит из создания проекта, создания и редактирования сцен внутри проекта и наполнения проекта содержанием в виде различных ресурсов.

Ресурсы делятся на следующие типы:

1) Prefabs (от слова Prefabricated).

Рисунок 20

Это объекты проекта, являющиеся заготовками для различных сущностей, реализуемых в сцене. Как правило, такой проект содержит информацию о положении объекта на игровом экране, графическом контенте, используемом объектом, присоединенных к объекту скриптах etc. В примере префаба Player на рисунке 21 видно, что для данного объекта определены скрипт поведения в игре (Player (Script)), физическая модель (Rigidbody 2D), коллижн-модель в рамках игрового мира (Box Collider 2D), набор анимаций

(Animator 2D), положение в игровом мире (Transform) и графическая оболочка объекта (Sprite Renderer).

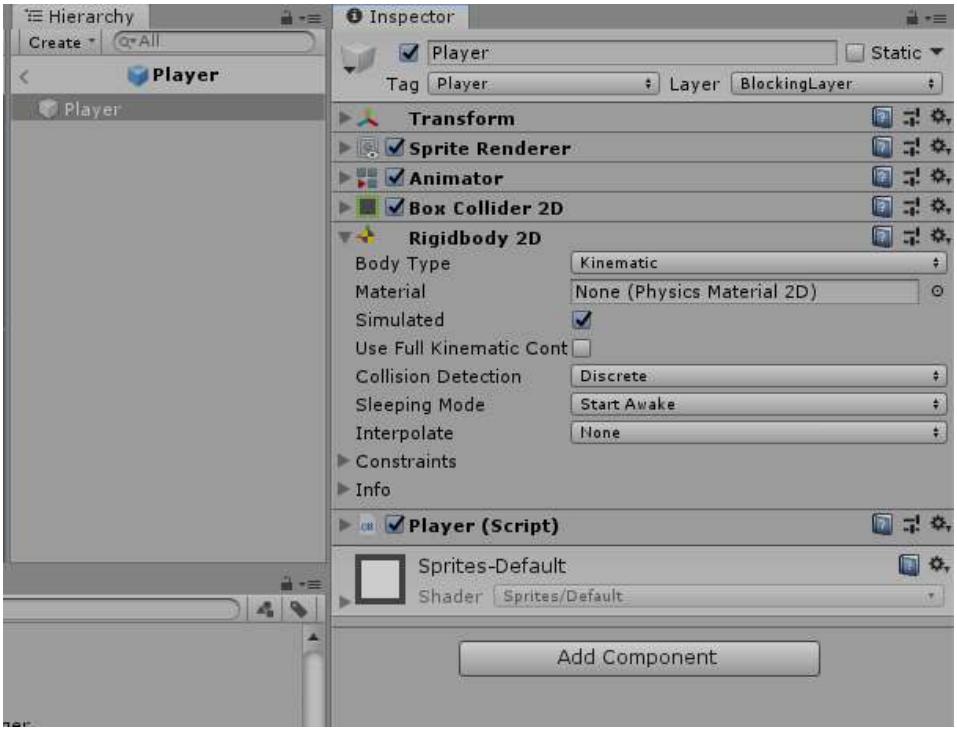

Рисунок 21 – префаб Player

2) Файлы анимаций (Animator)

Файлы анимаций бывают двух типов – собственно файл анимации, и controller-файл. Controller-файл объединяет несколько файлов анимации в единую систему анимации для какого–либо объекта. Например, на рисунке 22 видно систему анимации объектов типа Enemy1.

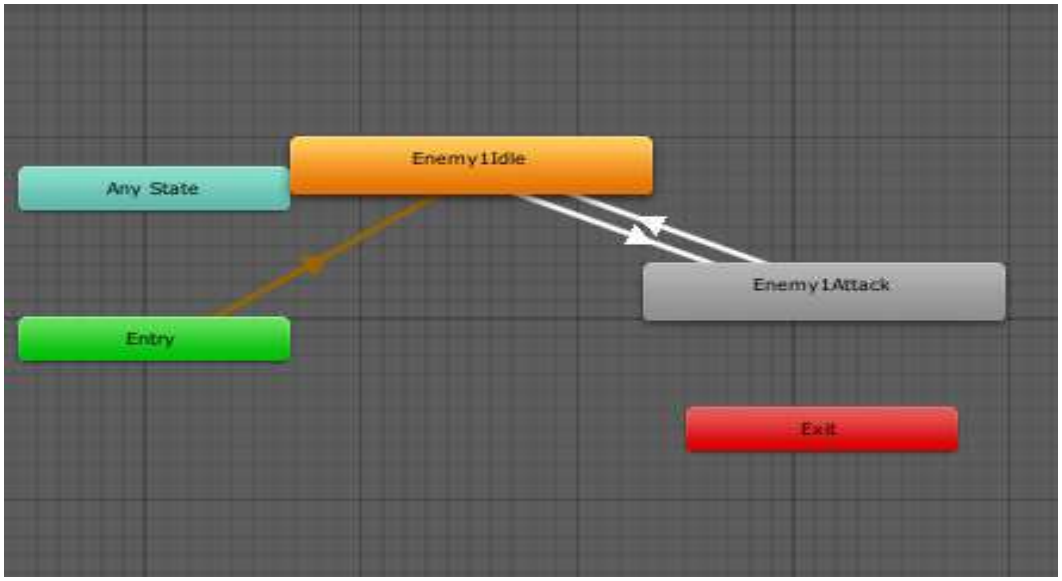

Рисунок 22 – Система анимаций Enemy1

Состояния отмечены прямоугольниками разного цвета. Между состояниями существуют связи-стрелки, содержащие условия перехода от одного состояния к другому. Условия перехода добавляются отдельно в окне.

Пример можно видеть на рисунке 23.

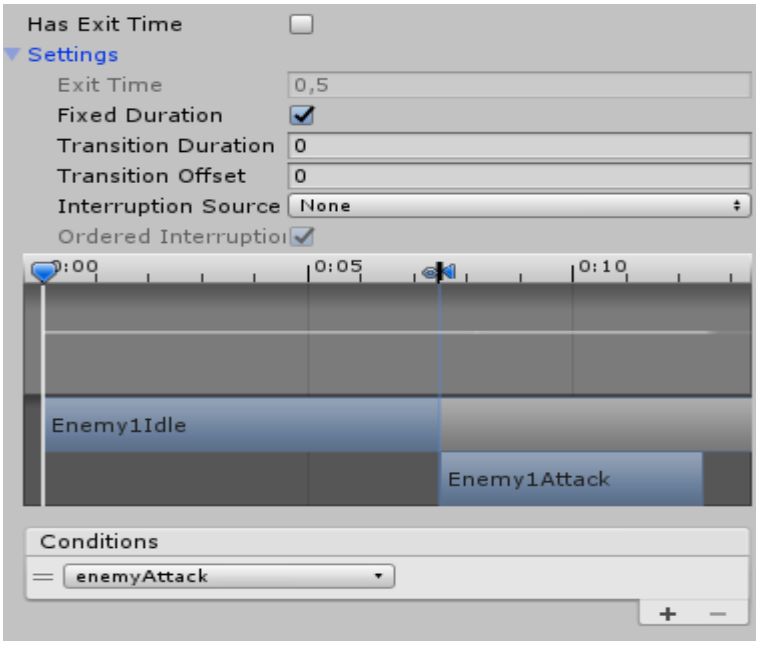

Рисунок 23 – Условия изменения состояния системы анимации

Как видно на рисунке, у анимаций есть множество различных параметров, например, условие выхода из анимации, время перехода от одной анимации к другой, и прочие.

3) Спрайты.

Представляют собой графический контент игры. Описаны в главе 2. Для каждого спрайта можно выбрать набор различных параметров, например, выбрать из алгоритма Митчелла или билинейной интерполяции для масштабирования изображения.

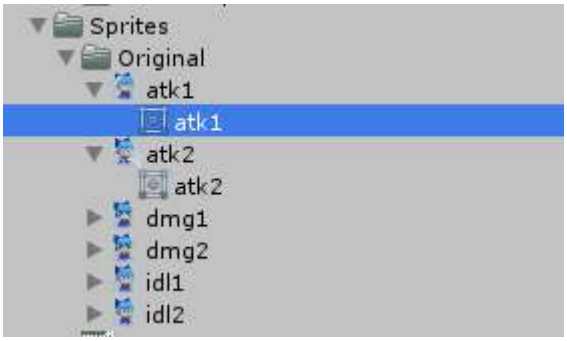

Рисунок 24

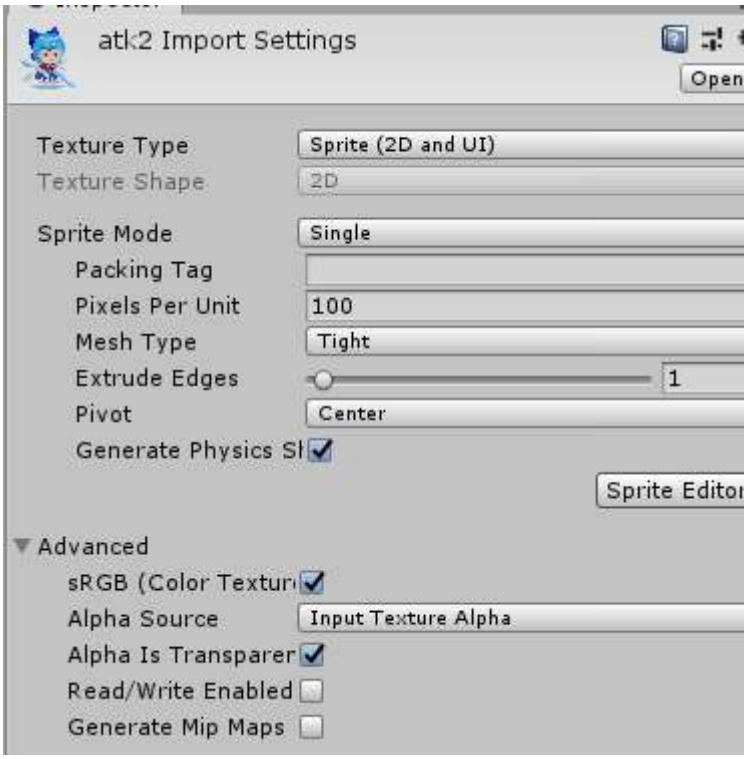

Рисунок 25

Есть 2 разных способа загрузить спрайты в Unity-проект: по отдельности, и в цельном файле, разбитом на xy-сетку с помощью прочего программного обеспечения (например, Adobe Photoshop).

Ресурсы для проекта были частично взяты из имеющихся в свободном доступе спрайтов подходящего размера (32x32 пикселя), частично созданы специально для проекта.

4) Скрипты.

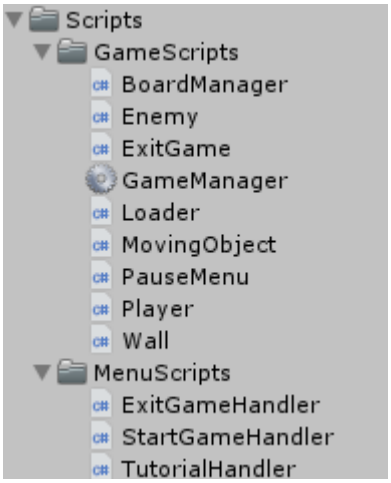

Рисунок 26

Скрипты для Unity должны быть созданы с использованием языка C#. Существует стандартный набор библиотек Unity, который необходимо подключить к скриптам для того, чтобы они взаимодействовали с движком. Скрипты определяют поведение различных объектов в игре, игровой цикл, и прочие функциональные элементы игры.

2.3 Реализация проекта.

Рассмотрим реализацию проекта на примере отдельных элементов, используемых в игровых сценах, а также игровую логику.

Рисунок 27 – Иерархия сцены MainMenu

В главном меню игры есть один элемент типа (ВСПОМНИТЬ), содержащий фон меню, и элементы типа Canvas (холст), содержащие различные страницы главного меню. Например, основной элемент Canvas содержит три объекта типа Button, в то время как TutorialCanvas – текстовый элемент Tutorial.

Объект Directional Light отвечает за присутствующий в сцене источник освещения. Есть выбор из различных параметров, например, типа освещения, типа теней, насыщенности освещения.

Объект EventSystem отвечает обработку событий, связанных с пользовательским интерфейсом, физическими объектами в сцене, рейкастингом (один из методов рендеринга), и прочими составляющими сцены. При создании объекта Canvas, Unity автоматически создаёт объект EventSystem. В указанном на рисунке примере подсветка элемента интерфейса – это реакция на событие IPointerEnterHandler, как раз реализуемого системой событий.

Рисунок 28

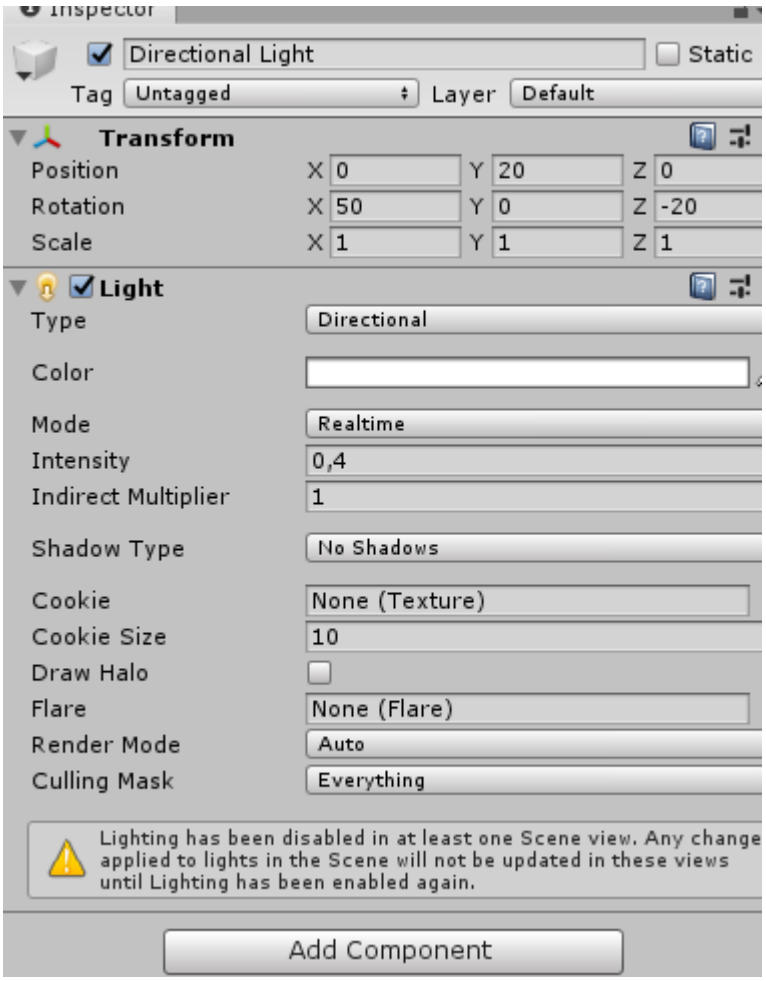

Рисунок 29 – Параметры объекта Directional Light

Все объекты, принадлежащие сцене, могут быть тем или иным образом отпозиционированы на её системе координат. Например, даже у элемента Event System есть координаты X,Y, Z, как и у прочих игровых объектов.

Рассмотрим подробней главную сцену, concept.

В иерархии данной сцены также присутствуют несколько Canvas, Event System и другие объекты, использовавшиеся в сцене Main menu. Тем не менее, набор объектов для этой сцены существенно шире, чем для главного меню.

Во-первых, в данной сцене используется объект Player.

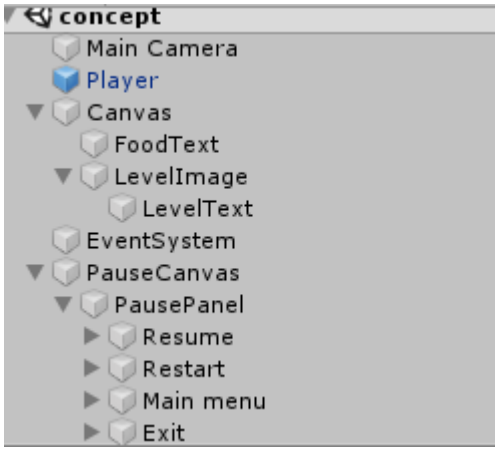

Рисунок 30 – иерархия сцены concept

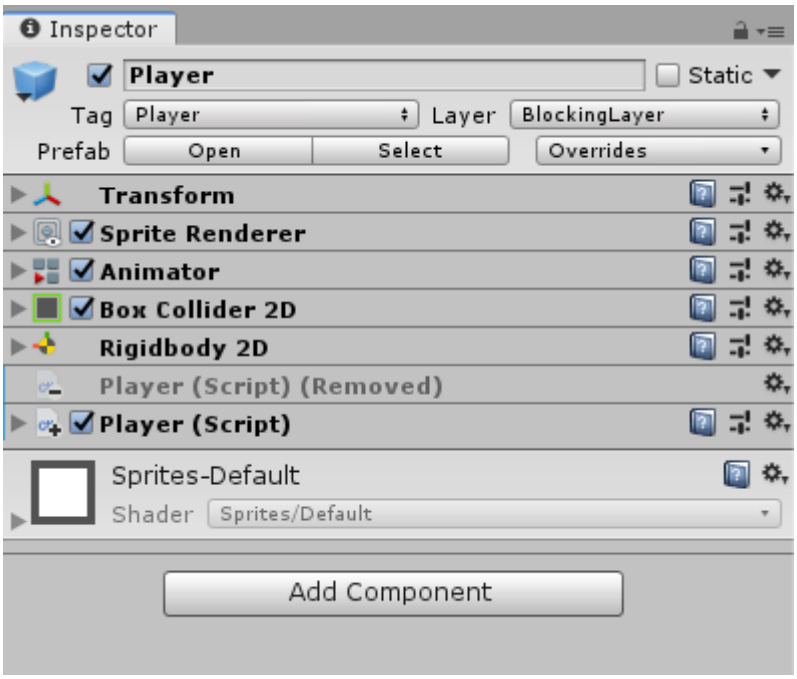

Рисунок 31 – компоненты объекта Player

Объект Player обладает набором анимаций (Animator), физикой и моделью столкновений (Rigidbody 2D, Box Collider 2D), скриптами (Player), и спрайтовым компонентом.

Рассмотрим для примера некоторые параметры компонента Rigidbody 2D, чтобы увидеть, как этот компонент влияет на поведение объекта в игре.

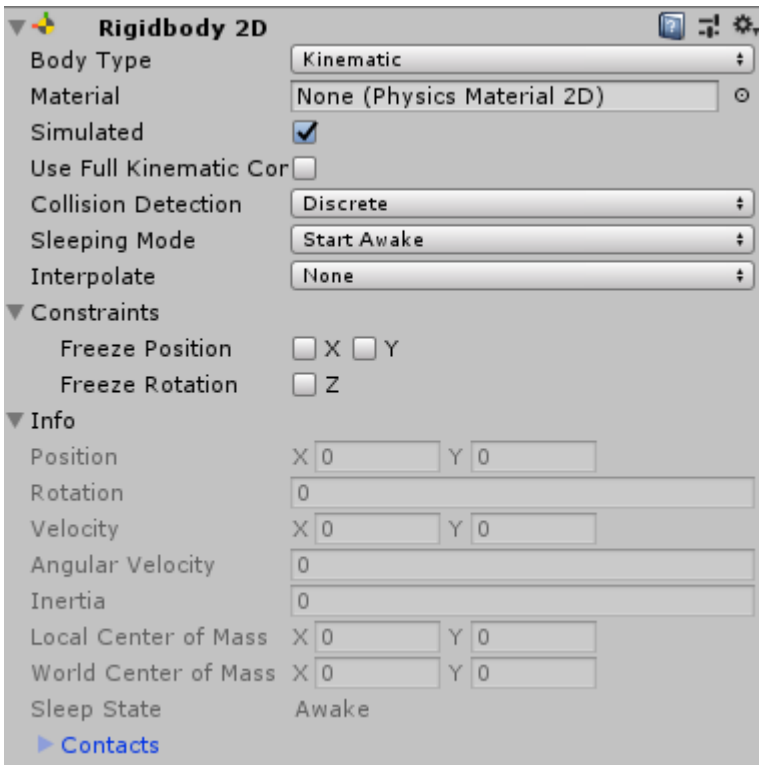

Рисунок 32 – компоненты объекта Player

Компонент Rigidbody даёт игровому физическому движку контроль над объектом.

Параметр Body type определяет то, как физический движок будет взаимодействовать с объектом. Выбор возможен из трёх параметров – Dynamic, Static и Kinematic. В случае нашего проекта, для игрока был выбран тип Kinematic. Данный тип подразумевает движение в рамках игровой сцены/симуляции, но, при этом, под строгим контролем. Данный тип тела наименее требователен к ресурсам системы, поскольку его позиционирование осуществляется исключительно с помощью внутриигровых методов в скриптах. Это значит, что данный объект, например, никак не подвержен силам гравитации.

Наконец, для Kinematic-объект возможны коллизии только с объектами типов Dynamic, но не Static и другими Kinematic.

На рисунке 33 изображен один из элементов алгоритма перемещения Kinematic-объекта Enemy1. Все передвигающиеся объекты в игре реализованы именно с такой физической моделью.

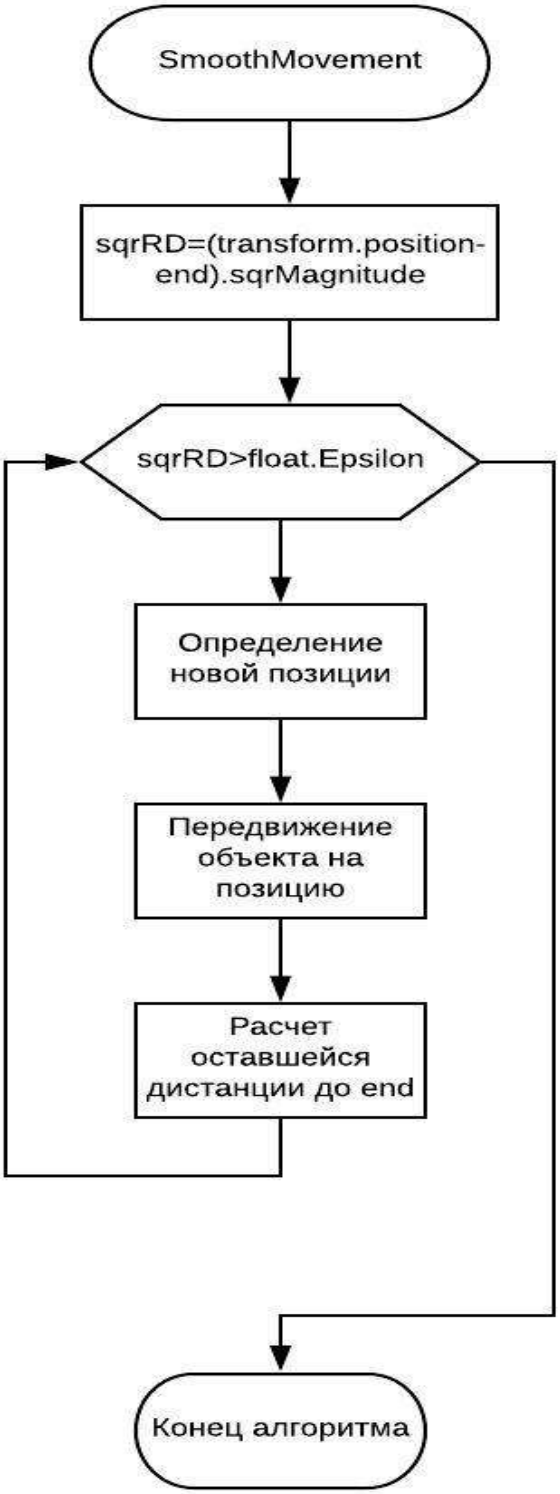

Рисунок 33 – алгоритм передвижения

У компонента Rigidbody есть и другие параметры. Например, если отметить пункт Use Full Kinematic, то коллизии Kinematic-объекта становятся возможны со всеми носителями компонента Rigidbody 2D, вне зависимости от их типа.

За взаимодействие с другими объектами в симуляции отвечает не только компонент Rigidbody, а и компонент Box collider.

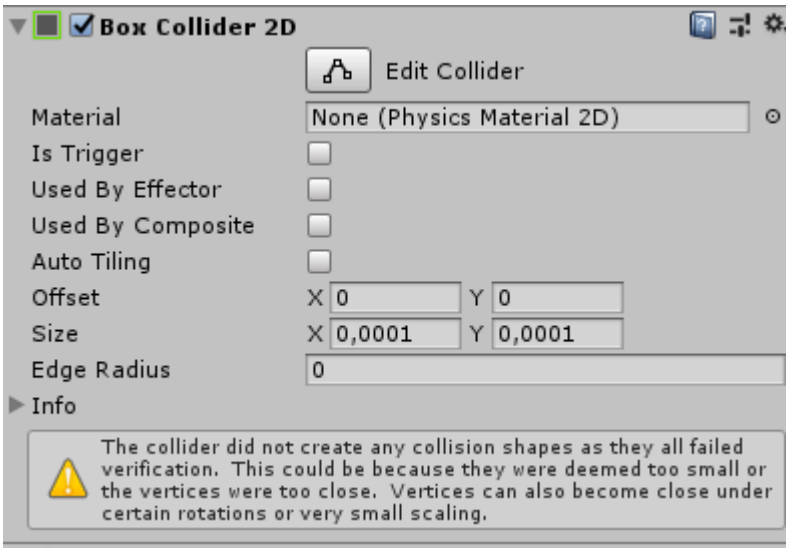

Рисунок 34 – компонент Box Collider

Параметр Material отвечает за настройку эффектов трения объекта.

Параметр IsTrigger делает объект триггером для событий, игнорируемым физическим движком.

Параметр Offset отвечает за положение коллижн-модели в объекте, а Size определяет размер.

Наконец, кроме объекта Player, в сцене используется также множество других объектов и ресурсов. Рассмотрим вкратце некоторые из них.

Рисунок 35 – Fonts

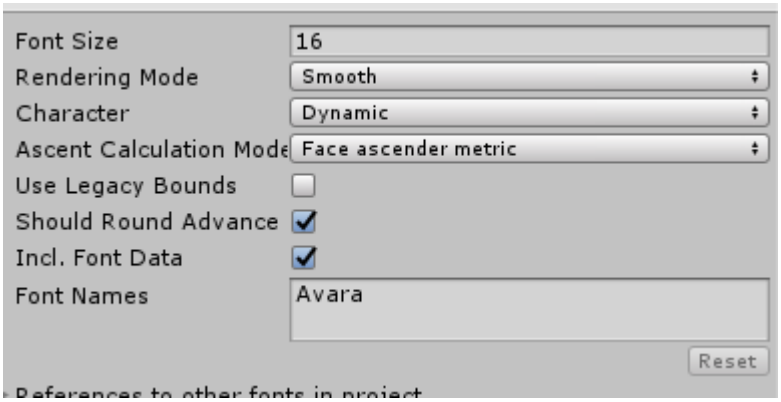

Рисунок 36

Ресурсы оформления, которые содержат используемые в игре шрифты. В движке есть некоторые настройки отображения текста в игре, например, несколько методов рендеринга.

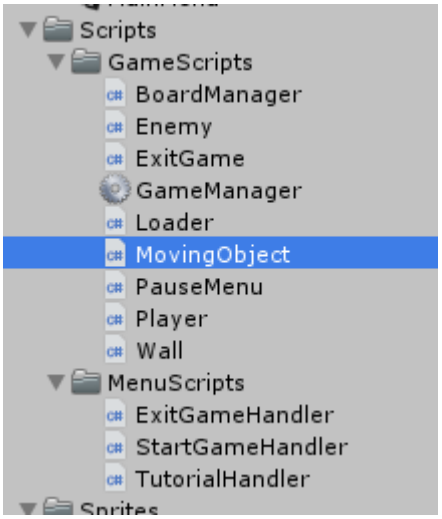

Рисунок 37 – список скриптов

В сцене concept скрипты используются более широко, чем в сцене Main menu. Рассмотрим то, для чего нужны некоторые из них.

1) GameManager

GameManager – один из основных классов игры, управляет всем игровым циклом. Именно в этом скрипте определена очередность передвижения объектов по карте, номер генерируемого уровня, список противников, и некоторые функциональные элементы, например, функция окончания игры.

2) BoardManager

Данный скрипт содержит реализацию описанного алгоритма создания уровня, параметры количества стен, а также некоторые функции, связанные с позиционированием объектов на сцене.

3) Moving Object

Скрипт, отвечающий за поведение движущихся объектов в целом. Именно в нём реализованы описанные алгоритм передвижения по уровню, и некоторые другие функции, связанные с перемещением объектов.

4) Player

Данный скрипт содержит взаимодействие объекта Player с анимациями, определяет количество ресурсов (яблоки), остающихся у игрока, и функции их прироста и убывания, и прочий функционал, связанный с игроком.

5) Enemy

Аналогично скрипту Player, за исключением того, что содержит алгоритм простого ИИ перемещения к игроку.

6) Loader

Содержит метод для загрузки игры в целом.

Скрипты, созданные для движка, взаимодействуют со средой разработке как с помощью графического интерфейса самой инструментальной среды, так и структурно.

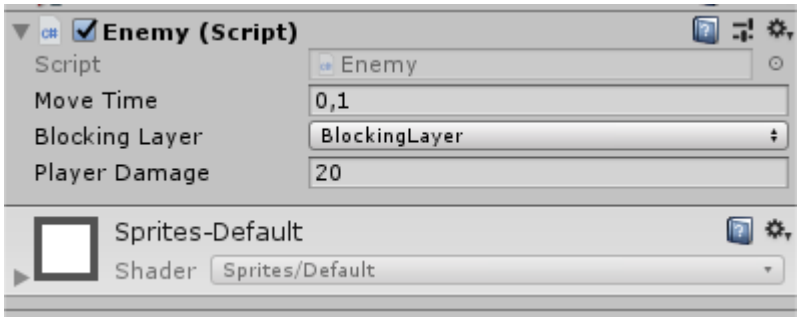

Рисунок 38 – активные элементы скрипта

В частности, на рисунке 38 мы видим скрипт Enemy, присоединённый к префабу Enemy1 в проекте. Поля MoveTime, BlockingLayer и PlayerDamage являются private–полями различных классов, объявленными в программном коде самим пользователем.

С помощью графического интерфейса, пользователь может легко помещать в данные поля не только различные переменные, но и целые объекты в движке (например, префабы).

Например, на рисунке 39 в поле Pause Menu UI с помощью графического интерфейса помещен объект PausePanel.

Рисунок 39 – взаимодействие скрипта с графическим интерфейсом

На рисунке 40 изображена иерархия классов в разработанном приложении.

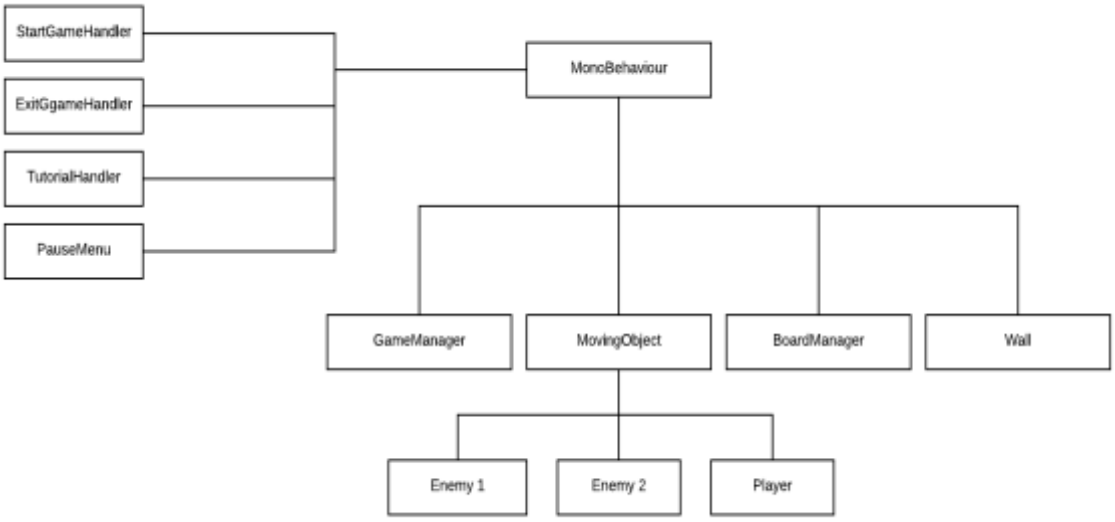

Рисунок 40 – взаимодействие скрипта с графическим интерфейсом

MonoBehaviour представляет собой базовый класс Unity, обеспечивающий связь скрипта с инструментальной средой. Например, метод Awake – один из наследуемых методов от класса MonoBehaviour.

Запуск приложения осуществляется из главного меню.

Рисунок 41 – главное меню

В главном меню пользователь может выбрать одну из следующих опций:

1) Start game – начать игру.
2) Tutorial – просмотреть обучающее сообщение.
3) Выйти из игры.

После запуска игры, игрок видит экран, сообщающий номер уровня, а затем главный игровой экран.

Рисунок 42 – обучающий экран

Рисунок 43 – сообщение номера уровня

Рисунок 44 – главный игровой экран

На главном игровом экране присутствуют следующие объекты:

1) Игрок. Может атаковать противников, может уничтожать разрушаемые препятствия, может собирать один ресурс (яблоки). Игра продолжается до тех пор, пока ресурс больше или равен нулю. Каждая атака противника отнимает у контролируемого игроком персонажа 10 яблок.

   Игрок может как атаковать противников, так и просто избежать их атаки.
2) Ресурсы. Существует только один вид внутриигровых ресурсов: яблоки. Встав на клетку с ресурсом, игрок поднимает 3 яблока.
3) Уничтожаемые объекты. Препятствия, обозначенные коричневыми тайлами земли с изображением ветвей дерева.
4) Неуничтожаемые объекты. Обозначены различными типами камней. Удерживают игрока на уровне.

Рисунок 45 – меню паузы

При нажатии на кнопку Resume, внутриигровое меню будет закрыто, и игра продолжится.

При нажатии на кнопку Restart счётчик уровней сбрасывается до 1, а счётчик еды персонажа до 99.

При нажатии на кнопку Main menu, игрок покинет цену и выйдет на главное меню.

При нажатии на кнопку Exit, приложение будет закрыто.

Заключение

В результате выполнения выпускной квалификационной работы были выполнены следующие задачи.

Был проведен анализ различных приложений. На основании полученной информации составлена объектная модель разрабатываемого приложения-видеоигры, включающая описание объектов на игровом уровне, их поведение, а также механики взаимодействия.

Рассмотрены различные методы геометрического моделирования, используемые в разработке приложений, и примеры их использования в движках 3D-графики, наподобие Unity. Также изучены различные типы геометрических моделей, и рассмотрена специфика их применения в различных ситуациях.

Изучены различные инструментальные среды для разработки приложения, выбрана оптимальная.

Спроектировано, создано и протестировано приложение на движке Unity.

Приложение может быть дополнено и улучшено в дальнейшем, т.к. инструментальная среда предлагает широкие возможности по разработке приложений такого типа. В частности, возможно включить в игру сюжетные сцены, добавить различные игровые механики, и т.д.

Список использованных источников

1. Andrew Troelsen, Philip Japikse, C# 6.0 and the .NET 4.6 Framework (7th Edition), (русский перевод предыдущего издания: Язык программирования C# 5.0 и платформа .NET 4.5, Эндрю Троелсен).
2. Cry Engine vs Unreal vs Unity [Электронный ресурс]. Режим доступа: https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e
3. Difference between Unreal Engine and Cry Engine [Электронный ресурс]. Режим доступа: https://www.educba.com/unreal-engine-vs-cryengine/
4. Epic Games State of Unreal Opening session [Электронный ресурс]. Режим доступа: https://www.youtube.com/watch?v=_veq6Q51Zlg
5. Joseph Albahari, Ben Albahari C# 7.0 in a Nutshell: The Definitive Reference, (русский перевод предыдущего издания: C# 6.0. Справочник. Полное описание языка, Джозеф Албахари, Бен Албахари).
6. Sedgewick R. Algorithms in C, Part 5 - Graph Algorithms (3rd Edition Pt 5.) - R. Sedgewick - Addison-Wesley Professional - 512 с. - 2001 - ISBN 978-0201316636
7. Objectively comparing Unity and Unreal Engine [Электронный ресурс]. Режим доступа:
8. Unity ad GDC Keynote [Электронный ресурс]. Режим доступа: https://www.youtube.com/watch?v=YlcX-O7fRichttps://gametorrahod.com/objectively-comparing-unity-and-unreal-engine/
9. Акопян А.В. Заславский А.А. Геометрические свойства кривых второго порядка - Москва, МЦНМО Геометрические свойства кривых второго порядка - 152 с. - 2011 - ISBN 978-5-94057-732-4

10. Анамова Р.Р., Леонова С.А., Пшеничнова Н.В. Инженерная и компьютерная графика. Учебник и практикум для бакалавриата - Москва, Юрайт - 246 с. - ISBN 978-5-9916-8262-6
11. Большаков В.П., Чагина А.В. Инженерная и компьютерная графика. Теоретический курс и тестовые задания - В.П. Большаков, А.В, Чагина - СПб., БХВ-Петербург - 384 с. - 2016 - ISBN 9785977537681
12. Бонд, Д.Г.. Unity и C#. Геймплей от идеи до реализации [текст] / Бонд Джереми Гибсон, - СПб., Питер - 2019 - 928 с. - ISBN: 978-5-4461-0715-5
13. Васильев А. - C#. Объектно-ориентированное программирование - СПБ, Питер - 320 с. - 2012 - ISBN 978-5-459-01238-5
14. Вицулина Е.В., Журбенко П.А., Гузненков В.Н. Autodesk Inventor 2016 Трехмерное моделирование деталей и электронное оформление чертежей. Учебное пособие - Москва, ДМК Пресс - 124 с. - 2017- ISBN 978-5-97060-514-1
15. Георгиевский О.В., Толкач А.Н. Основы инженерной графики [текст] - Олег Викторович Георгиевский, А.Н. Толкач - Москва, Издательство ассоциации строительных вузов - 304 с. - 2016 - ISBN ISBN 978-5-93093-611-7
16. Голдштейн С., Зурбалев Д., Флатов И. - Оптимизация приложений на платформе .NET с использованием языка C# - Москва, ДМК Пресс - 524 с. - 2017 - ISBN 978-5-94074-944-8
17. Голованов Н.Н. Геометрическое моделирование [текст] / Николай Николаевич Голованов - Москва, Academia, 2011 - 272 с. - ISBN 978-5-7695-7168-8
18. Дегтярев, В.М. Компьютерная геометрия и графика / В.М. Дегтярев. - М.: Academia, 2017. - 200 с. ISBN 978-5-7695-8500-5.

19. Доуни А. Б. - Алгоритмы и структуры данных. Извлечение информации на языке Java. - СПБ, Питер - 240 с. - 2018 - ISBN 978-5-4461-0572-4
20. Королев.А.Л. Компьютерное моделирование. Лабораторный практикум - А.Л. Королев - Москва, Бином. Лаборатория знаний - 296 с. - 2017 - ISBN 978-5-9963-0270-3
21. Лисняк А.А., Чопоров С.В., Гоменюк С.И. - Использование функций В.Л. Рвачева для геометрического моделирования сложной формы - Litres - 120 с. - 2017 - 785457384934
22. Никулин Е.А. Компьютерная графика. Модели и алгоритмы [текст] - Е.А. Никулин - СПб., 2018 - 708 с. - ISBN 978-5-8114-2505-1
23. Патрашов А. Математическое руководство по созданию компьютерных игр. Справочник [текст] - Алексей Патрашов - Москва, Издательские решения - 360 с. - ISBN 9785448322839
24. Руководство по C# — структуры [Электронный ресурс]. Режим доступа:https://professorweb.ru/my/csharp/charp_theory/level9/9_6.php
25. Селезнев В.А. Дмитроченко С.А. Компьютерная графика. Учебник и практикум для академического бакалавриата - Москва, Юрайт, - 218с. - ISBN 978-5-534-01464-8, 978-5-534-07393-5
26. Скит Д. - C# для профессионалов. Тонкости программирования - Москва, Вильямс - 608 с. - 2019 - ISBN: 978-5-8459-1909-0
27. Тозик, В.Т. Компьютерная графика и дизайн: Учебник / В.Т. Тозик. - М.: Academia, 2016. - 672 с. - ISBN 9785769568350
28. Торн А., Основы анимации в Unity - Алан Торн - Москва, ДМК Пресс - 176 с. - 2019 - ISBN 9785970603772
29. Торн А. Искусство создания сценариев в Unity [текст] - Алан Торн - Москва, ДМК Пресс, 2019 - 360 с. - ISBN 978-5-97060-381-9
30. Тюкачев Н. А., Хлебостроев В. Г. - C#. Алгоритмы и структуры данных. Учебное пособие - СПб, Лань - 232 с. - 2017- ISBN 978-5-811-42566-2

31. Уткин А.А. Геометрическое моделирование - Москва, Флинта - 220 с. - 2014 - ISBN 978-5-9765-1956-5
32. Хейлсберг А. и др. Язык программирования C# /. - М.: Питер, 2012. - 784 с. - ISBN 978-5-459-00283-6
33. Хокинг Д. - Unity в действии. Мультиплатформенная разработка на C# - СПб., Питер - 352 с. - 2019 - ISBN 978-5-4461-0816-9