АННОТАЦИЯ

Данный документ посвящен разработке и описанию программы по теме «Детальное проектирование и программная реализация системы, моделирующей процессы создания и взаимодействия группы объектов».

Документация содержит в себе описание всего процесса создания программы, начиная с алгоритмов и логики объектов, заканчивая тестированием программы. Программа разработана и написана на языке C#. Средой разработки являлся игровой движок Unity 2018.3.5f1 и редактор кода Microsoft Visual Studio.

СОДЕРЖАНИЕ

АННОТАЦИЯ 4

СПИСОК ИСПОЛНИТЕЛЕЙ 6

ВВЕДЕНИЕ 7

1. ПОСТАНОВКА ЗАДАЧИ 8
   1. Цель разработки 8
   2. Описательная постановка задачи 8
   3. Условия выполнения программы 8
2. ПРОЕКТНОЕ РЕШЕНИЕ 9
   1. Абстрагирование и выделение объектов 9
   2. Построение иерархии классов 9
   3. Построение информационной модели 12
   4. Описание жизненного цикла программы и одного из объектов 13
   5. Диаграмма переходов состояний 15
   6. Диаграмма потоков данных и действий 18
3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 19
   1. Обоснование выбора языка (системы) программирования 19
   2. Общее описание взаимодействия программных модулей 19
   3. Описание реализации основных классов и их методы 22
   4. Интерфейс пользователя 24
   5. Критерии качества программной системы 34

ВЫВОДЫ 35

ПЕРЕЧЕНЬ ССЫЛОК 36

ПРИЛОЖЕНИЕ А 37

СПИСОК ИСПОЛНИТЕЛЕЙ

Волобуев Юрий Сергеевич – занимался разработкой основной идеи, разработкой объектно-ориентированного подхода, созданием скриптов для врагов, участвовал в создании вспомогательных скриптов и скрипта для главного персонажа и разработке анимации для них, участвовал в создании уровней для игры, редактуре уровней и создании алгоритмов генерации, участвовал в создании пояснительной записки, занимался звуковым сопровождением игры.

Долженко Илья Андреевич – занимался разработкой основной идеи, разработкой иерархии для игры, описанием свойств главного персонажа, функциями управления главным персонажем и разработкой, и созданием GUI. Участвовал в создании скриптов для врагов, участвовал в создании уровней для игры, создании алгоритмов генерации и изображений для врагов, участвовал в создании пояснительной записки, участвовал в разработке звукового сопровождения игры.

ВВЕДЕНИЕ

Целями курсового проектирования является изучение современного подхода к программированию на основе объектно-ориентированной технологии, приобретение навыков написания программ на языке с поддержкой ООП на примере написания программы согласно варианту задания.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

- выбор варианта задания и языка реализации, детализация поставки задачи;

- абстрагирование, разработка классов и их иерархии;

- написание текста (кодирование) разработанных классов на выбранном языке;

- разработка тестовых примеров;

- тестирование и отладка программы;

- разработка программных документов в соответствии с действующими стандартами.

В настоящее время, данный проект является актуальным, так как на данный момент персональные компьютеры распространены повсеместно. Компьютеры являются современной игровой платформой и именно ее мы выбрали для нашего курсового проекта.

Игры в жанре Roguelike все еще востребованы и популярны на сегодняшний день. Несмотря на эволюцию графики в видеоиграх, отличительной чертой данного жанра является графика в стиле Pixel Art.

Курсовой проект состоит из пояснительной записки, включающей в себя аннотацию, введение, основные разделы, выводы, список литературы, приложение и программы на электронном носителе.

1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ

* 1. Цель разработки

Целью курсового проекта является объектно-ориентированное проектирование системы и создания на ее основе игры, пользуясь принципами ООП.

* 1. Описательная постановка задачи

Реализуемая система – игра, подразумевающая управление персонажем и бои с врагами.

Враги делятся на несколько классов и имеют различные атаки. После смерти из них выпадают бонусы, усиливающие персонажа. Целью врагов является уничтожение главного героя.

Персонаж имеет шкалу здоровья, силу удара, скорость передвижения, уникальные способности.

Целью игры является уничтожение всех врагов на карте и поиск выхода из уровня.

Должно быть реализовано главное меню, которое содержит следующие пункты: меню выбора персонажа, меню настроек и кнопку выхода из игры, а также меню паузы с возможностью перезапуска уровня и выхода в главное в меню.

* 1. Условия выполнения программы

Системные требования:

* ОС: Windows XP/Vista/7/8/10
* Процессор: Intel Pentium 4 2.5 GHz
* Оперативная память: 512 Мб
* Видеокарта: DirectX Compatible
* Место на диске: 120 Мб
* Звуковая карта: DirectX Compatible

2 ПРОЕКТНОЕ РЕШЕНИЕ

2.1 Абстрагирование и выделение объектов

Рассмотрим виды объектов, которые потребуются для нашей разработки:

**1)** **Персонаж**

Это объект, которым будет управлять пользователь. Он необходим для того, чтобы мы могли перемещаться по игровому миру и взаимодействовать с другими объектами.

**2)** **Враги**

Это объекты, которые представляют угрозу жизни персонажа. Двигаются по случайной траектории. При появлении персонажа в области его видимости двигается за ним и совершает попытки атаки.

**3)** **Комната**

Эти объекты создаются по шаблону в результате процедурной генерации, являются составными частями игрового уровня.

**4) Камера**

Объект позволяет наблюдать за игровым процессом, двигается за персонажем с небольшой задержкой для обеспечения плавности картинки.

**5) Canvas**

Данный объект связан с интерфейсом пользователя. В нем содержатся панели способностей, мини-карта, шкала здоровья и счетчик монет.

2.2 Построение иерархии классов

В ходе проектирования нашей системы были разработаны иерархии классов, UML-диаграммы которых представлены на рисунках 1, 2.

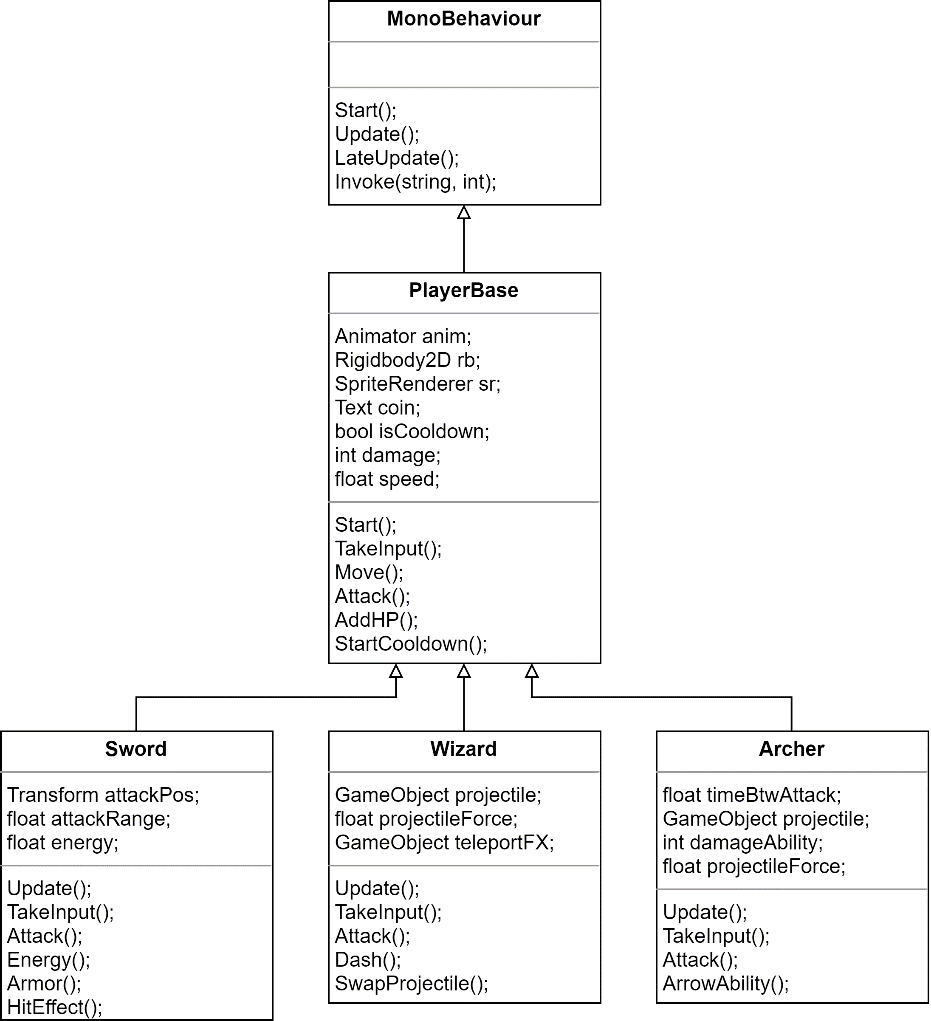


Рисунок 1 – Иерархия классов персонажа

PlayerBase – базовый класс для главного героя, содержит в себе виртуальные функции ввода клавиш и атаки героев, а также общие методы отображения анимации, добавления здоровья и отображения задержки перед применением способностей. Имеет общие поля количества урона и скорости.

Sword – класс наследуется от базового класса PlayerBase, содержит поля значений задержки времени между атаками, расстояние атаки, координаты атаки и энергию. В классе переопределяются функции отслеживания нажатий и атаки. Имеет собственный метод использования энергии для ускорения и уникальную способность.

Wizard – класс-наследник от базового класса PlayerBase, имеет собственные поля скорости снаряда и метод описывающий уникальную способность героя. Переопределяются методы ввода клавиш и атаки.

Archer – класс наследуется от базового класса PlayerBase. Имеет собственные поля урона способности, скорость снаряда и задержки между атаками, а также собственный метод отвечающий за уникальную способность героя. В классе переопределяется метод атаки.

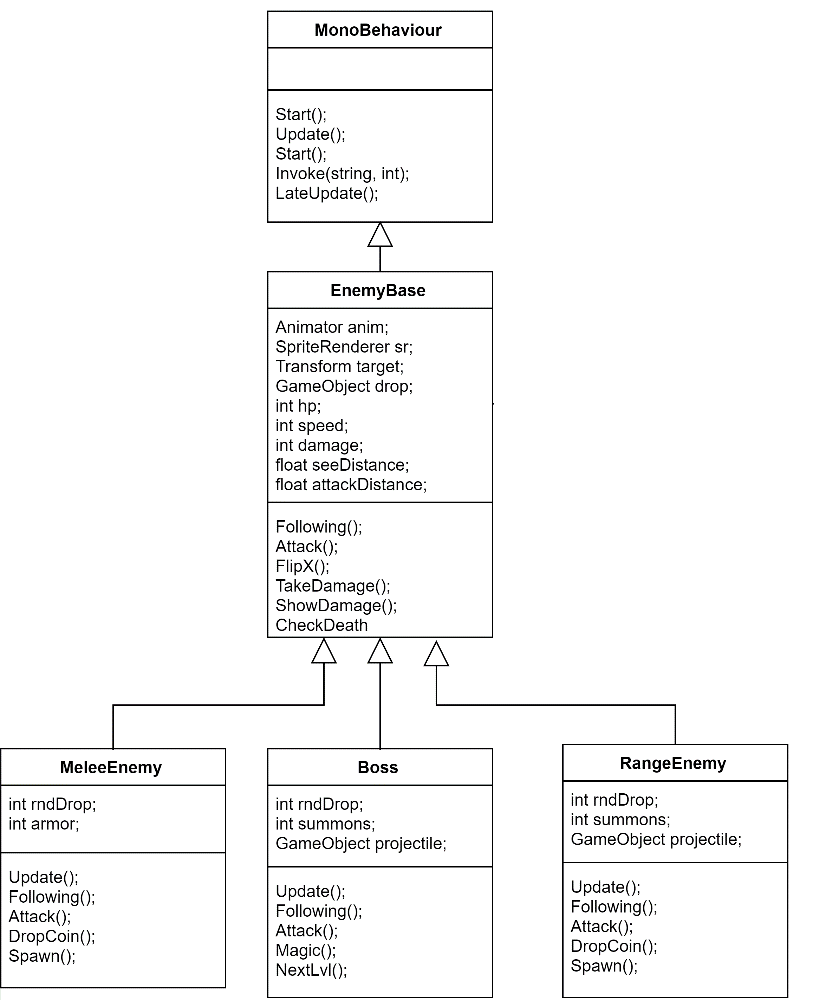


Рисунок 2 – Иерархия классов врагов

EnemyBase – базовый класс врага, содержит основные характеристики врагов (здоровье, скорость, урон, дистанция атаки и видимости), виртуальные методы движения и атаки, а также общие методы отображения объекта, получения урона и его отображение на экране.

MeleeEnemy – класс-наследник от базового класса EnemyBase, содержит целочисленные поля количества брони и монет. В классе переопределяется метод движения и атаки. Содержит собственные методы появления на карте и метод генерации количества вещей, выпадающих после смерти.

RangeEnemy – производный класс от базового класса EnemyBase, содержит целочисленное поле количества монет и игровой объект снаряда. В классе переопределяется метод движения и атаки.

Boss – класс-наследник от базового класса EnemyBase, содержит поле количества вещей/монет выпадающих после смерти и игровой объект снаряда. Имеет собственный методы применения способностей и перехода на следующий уровень. Методы движения и атаки переопределяются.

2.3 Построение информационной модели

На рисунке 3 изображены объекты, располагающиеся на сцене, а также их взаимосвязь.

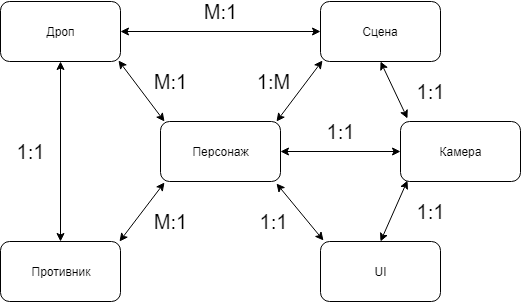


Рисунок 3 – Информационная модель

Дроп – склянки здоровья или монеты, выпадающие с врагов или разбросанные по карте.

Противник – может быть динамическим объектом, который при обнаружении в зоне видимости главного персонажа бежит на него и атакует. Отнимает определенное количество здоровья за определенный промежуток времени. Также на карте генерируются шипы – опасность для главного героя, при столкновении с ними герой теряет здоровье.

Главный герой – основной персонаж, объект одного из трёх классов, которым управляет игрок.

Сцена – генерируемая случайным образом карта уровня, по которой передвигается главный персонаж и где находятся основные объекты, такие как противники, препятствия и монеты.

UI – элементы интерфейса, позволяющие контролировать состояние персонажа, его расположение на карте и использование уникальных способностей.

Камера – объект, движущийся за персонажем и отображающий игровое поле в определенном радиусе.

2.4 Описание жизненного цикла программы и одного из объектов

MonoBehaviour – базовый класс Unity, от которого наследуются все созданные классы, содержит методы и переменные, необходимые для взаимодействия между объектами и компонентами других классов. Основные вещи, достойные внимания, это две функции, определенные внутри класса.

Функция **Update** – это место для размещения кода, который будет обрабатывать обновление кадра для игрового объекта. Это может быть движение, срабатывание действий и ответная реакция на ввод пользователя, в основном всё, что должно быть обработано с течением времени во игровом процессе.

Функция **LateUpdate** вызывается раз в кадр, после завершения Update. Любые вычисления, произведённые в Update будут уже выполнены на момент начала LateUpdate. Часто LateUpdate используют для преследующей камеры от третьего лица. Если вы перемещаете и поворачиваете персонажа в Update, вы можете выполнить все вычисления перемещения и вращения камеры в LateUpdate. Это обеспечит то, что персонаж будет двигаться до того, как камера отследит его позицию.

Функция **Start** будет вызвана Unity до начала игрового процесса (т.е. до первого вызова функции Update), и это идеальное место для выполнения инициализации.

Функция **OnApplicationQuit** вызывается для всех игровых объектов перед тем, как приложение закрывается.

Жизненный цикл класса MonoBehaviour представлен на рисунке 4.

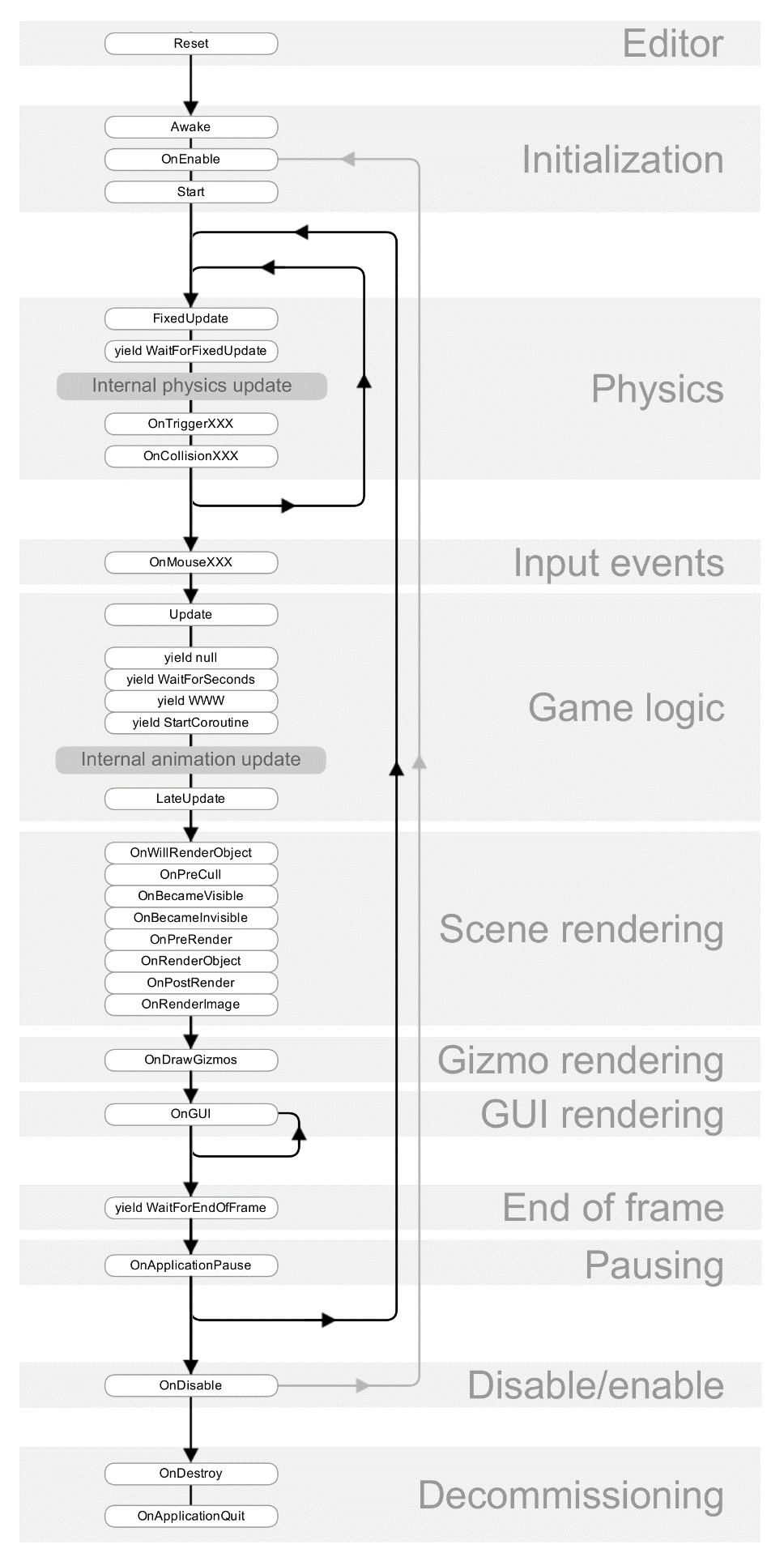


Рисунок 4 – Жизненный цикл класса MonoBehaviour

После запуска игры активируется объект, которым управляет игрок. Он может двигаться, драться с противниками, что вызовет действие проверки здоровья. Если запас здоровья равен нулю, то персонаж уничтожается, и игра начинается заново. Данный цикл изображён на рисунке 5.

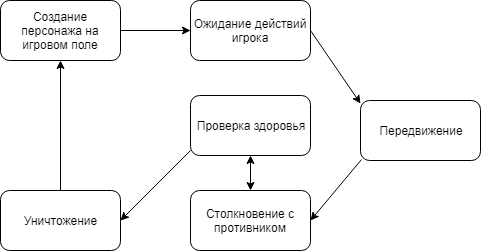


Рисунок 5 – Диаграмма жизненного цикла игрока

2.5 Диаграмма переходов состояний

Состояние *Выбор персонажа* – первое состояние игрового приложения. При переходе в него становится доступно меню выбора персонажа. После выбора персонажа происходит переход в состояние *Игровой процесс*. Диаграмма состояний выбора персонажа представлена на рисунке 6.

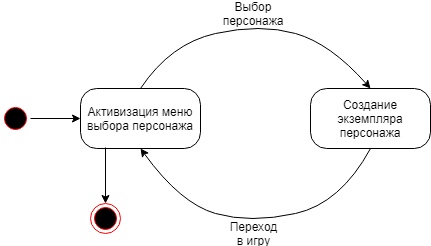


Рисунок 6 – Диаграмма состояний выбора персонажа

Состояние *Игровой процесс* – в этом состоянии игроку доступно передвижение персонажа, битва с врагом и поиск перехода на следующий уровень. Диаграмма состояний игрового процесса представлена на рисунке 7.

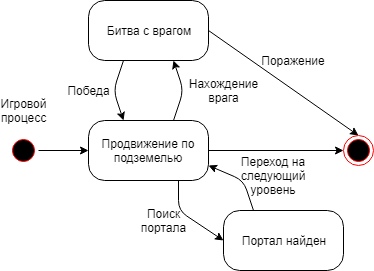


Рисунок 7 – Диаграмма состояний игрового процесса

Состояние *Битва с врагом* – в это состояние игрок переходит, когда встречается с врагом. Персонаж и враг обменивается уроном и каждый кадр проверяется их здоровье. Диаграмма состояний битвы с врагом представлена на рисунке 8.

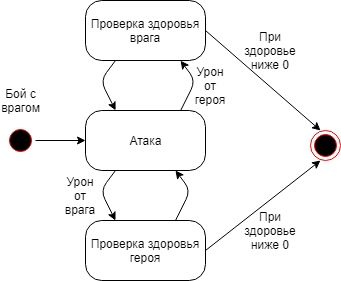


Рисунок 8 – Диаграмма состояний битвы с врагом

Состояние *Движение врага* – если враг не видит персонажа, он двигается по случайным координатам, иначе он получает координаты персонажа и двигается к нему. Диаграмма состояний движения врага представлена на рисунке 9.

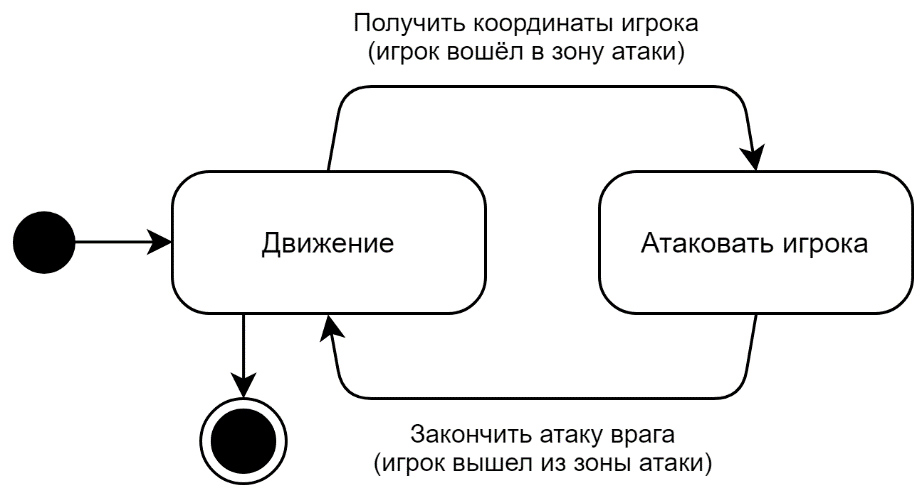


Рисунок 9 – Диаграмма состояний движения врага

Состояние *Появления врага* – система создает точки появления врагов в комнате. Диаграмма состояний появления врага представлена на рисунке 10.

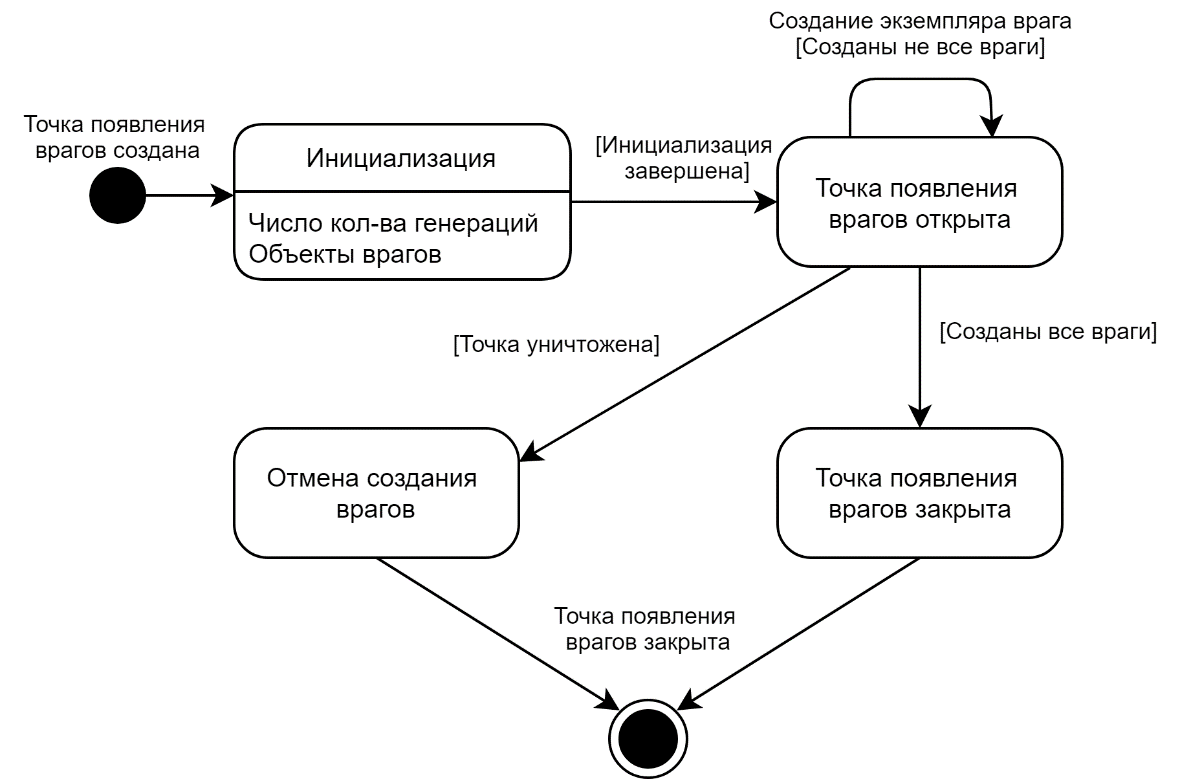


Рисунок 10 – Диаграмма состояний появления врага

Состояние *Босса* – система создает экземпляр класса босс, в этом состоянии объект создает снаряды, летящие в персонажа. Каждый кадр происходит проверка здоровья и если оно равно 0, то объект уничтожается. Диаграмма состояний Босса представлена на рисунке 11.

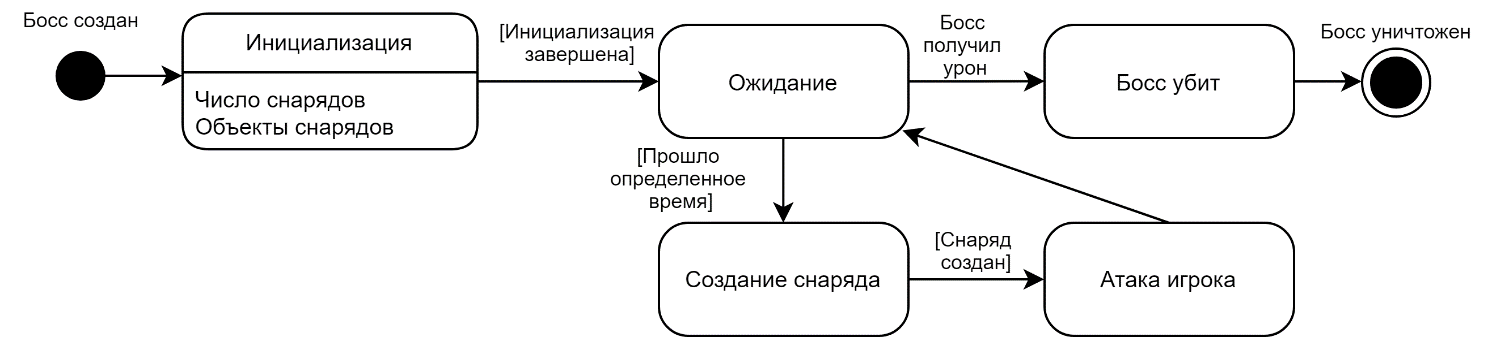


Рисунок 11 – Диаграмма состояний босса

2.6 Диаграмма потоков данных и действий

На рисунке 12 проиллюстрировано какие потоки данных влияют на состояние ожидания и какие данные влияют на другие состояния.



Рисунок 12 – Диаграмма потоков данных для состояния ожидания

Входящими данными являются текущий уровень здоровья и координаты врага. Исходящими данными являются введенные клавиши, которые влияют на дальнейшие действия персонажа.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Обоснование выбора языка (системы) программирования

Основным средством разработки данного проекта является игровой движок Unity. Unity – это мультиплатформенный инструмент для разработки двух- и трёхмерных приложений и игр, работающий под операционными системами Windows и OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Windows Phone, Android, Apple. Есть возможность создавать интернет-приложения с помощью специального подключаемого модуля к браузеру Unity. Приложения, созданные с помощью Unity, поддерживают DirectX и OpenGL.

Разработка велась на языке C#, а в качестве редактора кода был выбран Microsoft Visual Studio. Это линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

3.2 Общее описание взаимодействия программных модулей

Для упрощения поиска и редактирования файлов были созданы папки для разного рода файлов (рис. 13):

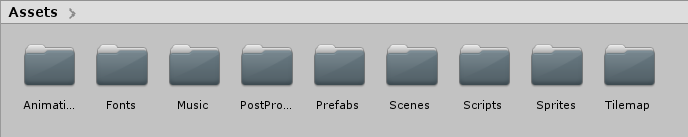


Рисунок 13 – Структура файлов

Animations — файлы анимаций, Fonts — используемые шрифты, Music — музыка и звуковые эффекты, PostProcessing — плагин постобработки изображения, Prefabs — заготовки объектов, Scenes – сцены проекта, Scripts — файлы скриптов (классов), Sprites — изображения игровых объектов и интерфейса, Tilemap – плагин анимации тайлов.

Особенностью Unity является его система управления объектами: каждому объекту можно присвоить какое угодно количество скриптов (имя скрипта соответствует имени класса и по сути это одно и тоже) и управление объектом совершается через эти скрипты (рис. 14).

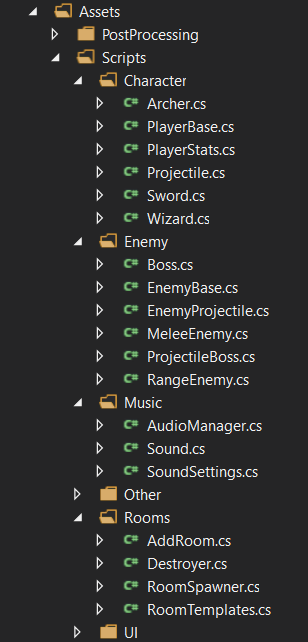


Рисунок 14 – Структура скриптов

В папке Scripts есть несколько типов управляющих скриптов, предназначенных для конкретных объектов. Character — скрипты персонажей, Enemy — скрипты врагов, Music — система управления звуками и музыкой, Other — скрипты побочных игровых объектов, Rooms – скрипты для генерации комнат, UI – скрипты для управления пользовательским интерфейсом.

В папке Prefabs находятся различные шаблоны используемых в проекте объектов (рис. 15):



Рисунок 15 – Шаблоны объектов

После загрузки моделей производятся различные действия с ними: применение скрипта, изменение свойств объекта, редактирование полей скрипта. Для того что бы не производить это вручную или внутри управляющего скрипта можно использовать интерфейс игрового движка Unity (рис. 16).

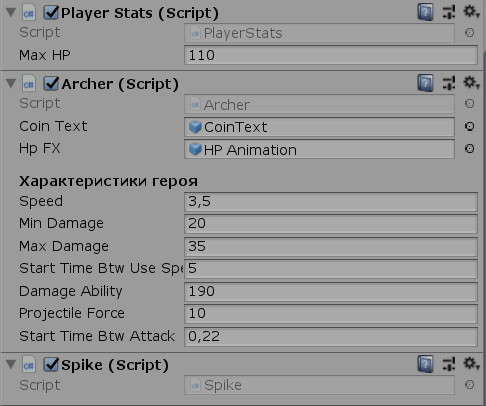


Рисунок 16 – Настройки скрипта персонажа Archer

3.3 Описание реализации основных классов и их методы

1) Класс PlayerBase содержит методы:

* protected virtual void TakeInput() – виртуальная функция, реализующая ввода клавиш управления.
* protected void Move() – функция отвечает за движение персонажа.
* private void AnimationMove(Vector 2 direction) – функция отвечает за анимацию персонажа.
* private void OnTriggerEnter(Colider2D other) – функция отвечает за взаимодействие с монетами.
* protected abstract void Attack() – абстрактная функция атаки, переопределяемая в наследуемых классах.

2) Sword, Archer и Wizard наследуются от базового класса PlayerBase и переопределяют функции TakeInput и Attack.

Класс Sword содержит уникальные методы Energy и Armor и метод отображения атаки HitEffect.

Класс Archer содержит метод ArrowAbility, который описывает уникальную способность выпустить усиленный снаряд.

Класс Wizard содержит уникальный метод Dash, который описывает уникальную способность рывка в направлении движения.

3) Класс EnemyBase содержит методы:

* protected abstract void Following() – абстрактный метод, отвечающий за движение за игроком.
* protected abstract void Attack() – абстрактный метод, отвечающий за атаку.
* protected void FlipX(float other) – метод, отвечающий за поворот спрайта врага.
* protected void TakeDamage(int damage) – метод предназначен для получения урона от игрока.
* protected void ShowDamage(int damage) – метод, предназначенный для отображения получаемого урона.
* protected void CheckDeath() – функция, проверяющая здоровья врага.

4) MeleeEnemy, RangeEnemy и Boss наследуются от базового класса EnemyBase и переопределяют функции Following и Attack.

Класс MeleeEnemy и RangeEnemy содержит разные методы определения координат персонажа и следования за ним.

Класс Boss переопределяет базовую функцию Attack с использованием Coroutine.

5) Классы AddRoom, Destroyer, RoomSpawner отвечают за генерацию игрового уровня, используя массив шаблонов комнат из класса RoomTemplates.

6) Следующие классы не содержат сложных функций и привязаны к простым объектам:

* Camera – отвечает за движение камеры за персонажем.
* Coins – предназначена для учета выпадающих монет.
* Portal – содержит функцию, отвечающая за переход на следующие уровни.
* BottleHP – содержит функцию для использования зелий.
* ArmorAbility – класс, реализующий методы для отображения защиты персонажа.
* Jar – класс отвечает за интерактивные предметы в игровом мире.
* Spike – класс реализует столкновения персонажа с шипами, наносящими урон.

7) Классы, реализующие GUI:

* BarHP – класс, отвечающий за отображения здоровье персонажа.
* Character – реализует выбор персонажа.
* Menu – содержит функции, реализующие главное меню.

8) Классы, отвечающие за звуковые эффекты:

* AudioManager – содержит массив звуков и функции их воспроизведения.
* SoundSettings – реализует меню настройки звука.

3.4 Интерфейс пользователя

Программа имеет собственный установщик, представленный на рисунке 17.

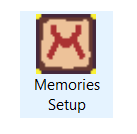


Рисунок 17 – Установщик игры

Далее проиллюстрирован процесс установки (рис. 18-20).

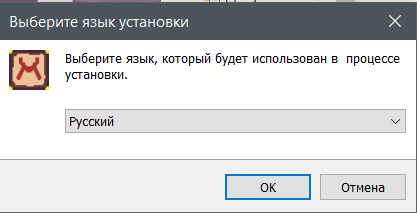


Рисунок 18 – Выбор языка установщика

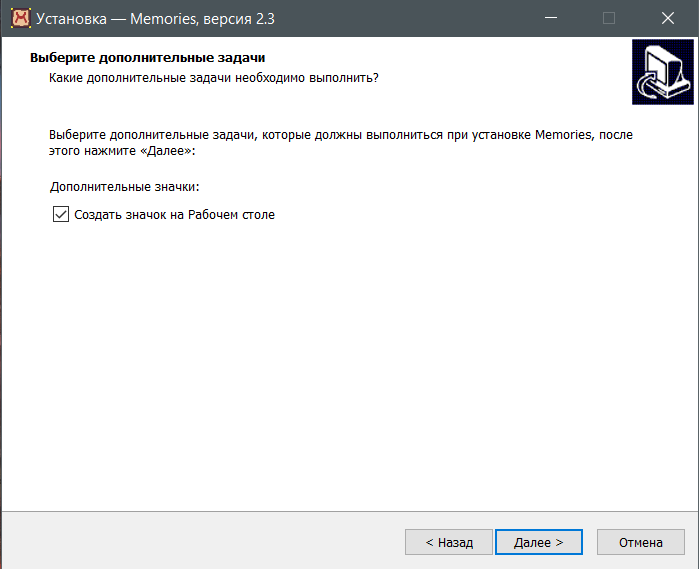


Рисунок 19 – Запрос на создание ярлыка игры

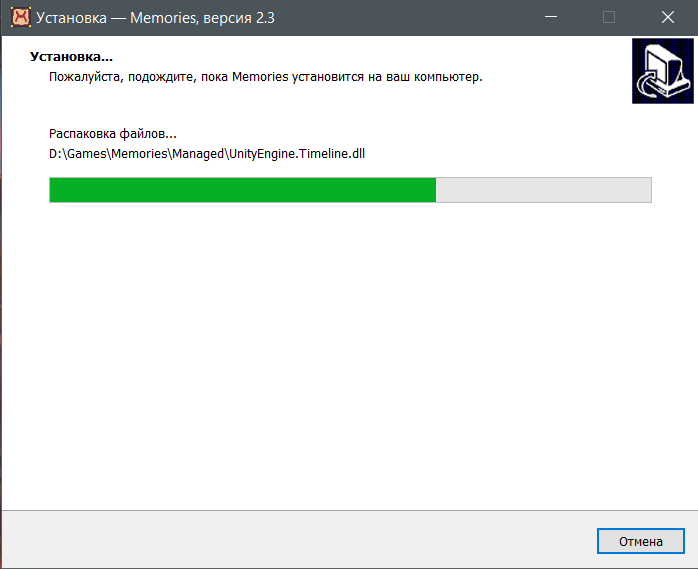


Рисунок 20 – Прогресс установки игры

Файл запускающий игру находится в папке, представленной на рисунке 21. При открытии исполнительного файла Memories.exe запускается игра (рис. 22).

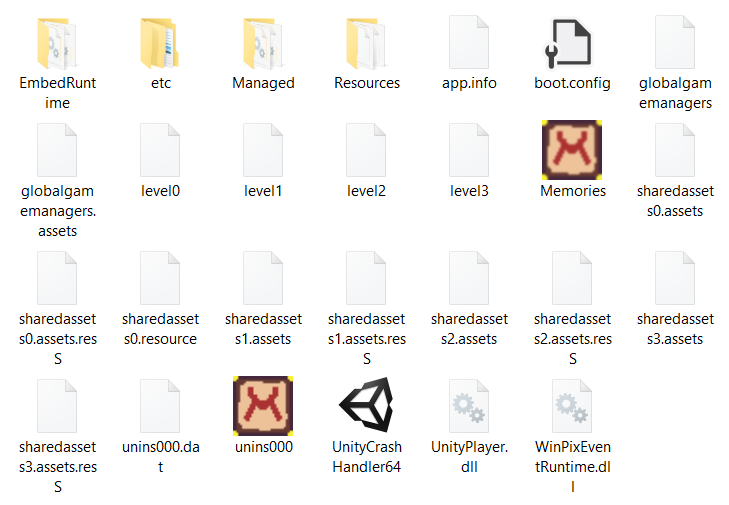


Рисунок 21 – Содержание папки с игрой

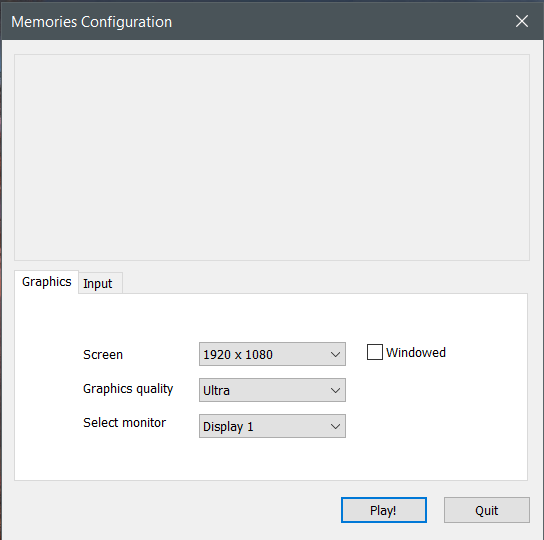


Рисунок 22 – Выбор настроек при запуске игры

Первое что видит игрок – главное меню, представленное на рисунке 23 и содержащее пункты: play, options, quit.



Рисунок 23 – Главное меню

В настройках игрок имеет возможность изменить уровень громкости музыки и звуков (рис. 24).

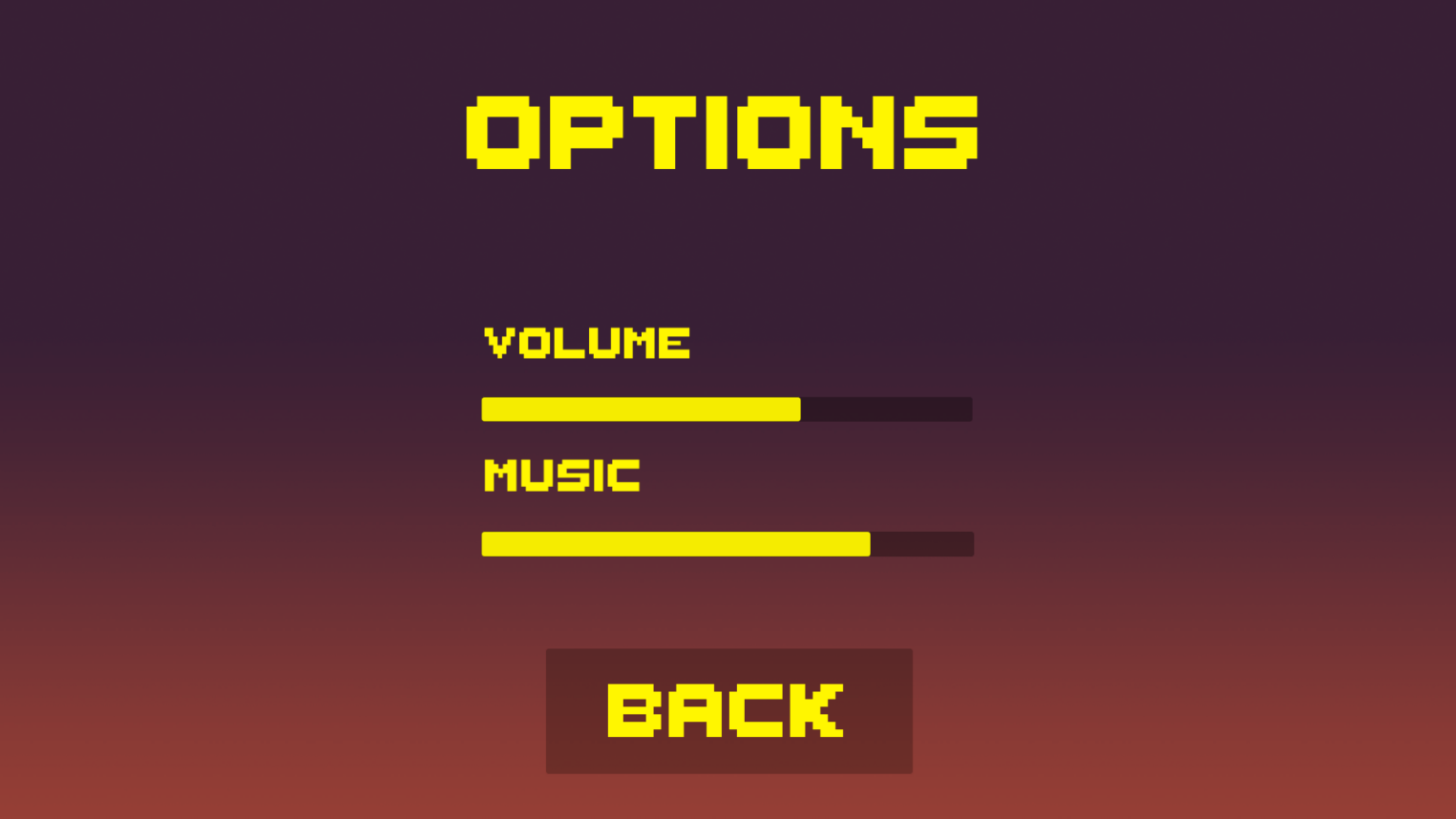


Рисунок 24 – Меню настроек

При нажатии на кнопку “Play” игроку становится доступно меню выбора персонажа (рис. 25). Персонажи имею свои уникальные параметры и умения.



Рисунок 25 – Меню выбора персонажа

После выбора персонажа начинается игровой процесс (рис. 26).



Рисунок 26 – Начало игрового процесса

Нажав клавишу “Esc” игрок попадает в меню паузы, представленную на рисунке 27.



Рисунок 27 – Меню паузы

Для каждого персонажа определена клавиша использования уникальной способности.

Для героя-мечника уникальная способность — это создание защитной сферы, которая снижает весь получаемый урон в 2 раза, действие данной способности представлено на рисунке 28.

Для героя-лучника уникальная способность позволяет запустить золотую стрелу, которая нанесет большой урон, тем самым мгновенно убив любого врага кроме босса.

Герой-маг имеет способность телепортации на короткие расстояния, что позволяет ему избегать вражеские атаки и быстрее преодолевать расстояние (рис. 29).



Рисунок 28 – Использования защиты Мечника



Рисунок 29 – Использование рывка Мага

Игровой процесс подразумевает битву с врагами (рис. 30) и в процессе игрок может получить урон от них.



Рисунок 30 – Битва с врагами

Игроку предоставлена возможность восстановить потерянное здоровье (рис. 31) при помощи склянок здоровья, выпадающих с врагов.



Рисунок 31 – Использование склянок здоровья

Если игрок не успеет восстановить потерянное здоровье его ждет смерть. (рис. 32).

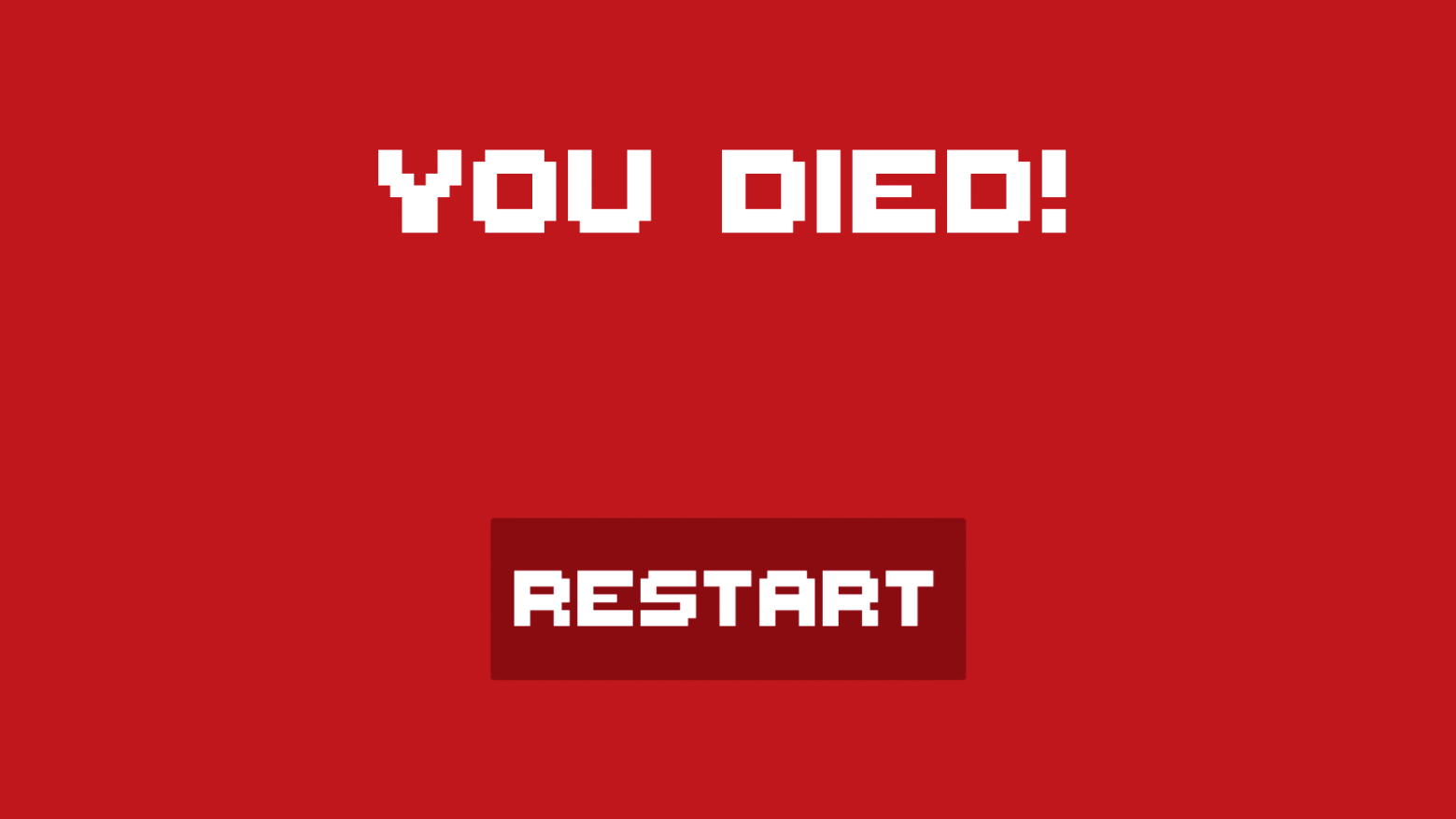


Рисунок 32 – Смерть персонажа

Помимо выпадения с врагов, монеты можно найти на уровне, данный процесс представлен на рисунке 33.



Рисунок 33 – Процесс нахождения монет

Целью игры является поиск портала, который переводит врага на следующий уровень (рис. 34).



Рисунок 34 – Портал

На финальном уровне подземелья игрока ждет битва с боссом, представленная на рисунке 35. После смерти босса закончится игра (рис. 36).



Рисунок 35 – Битва с боссом

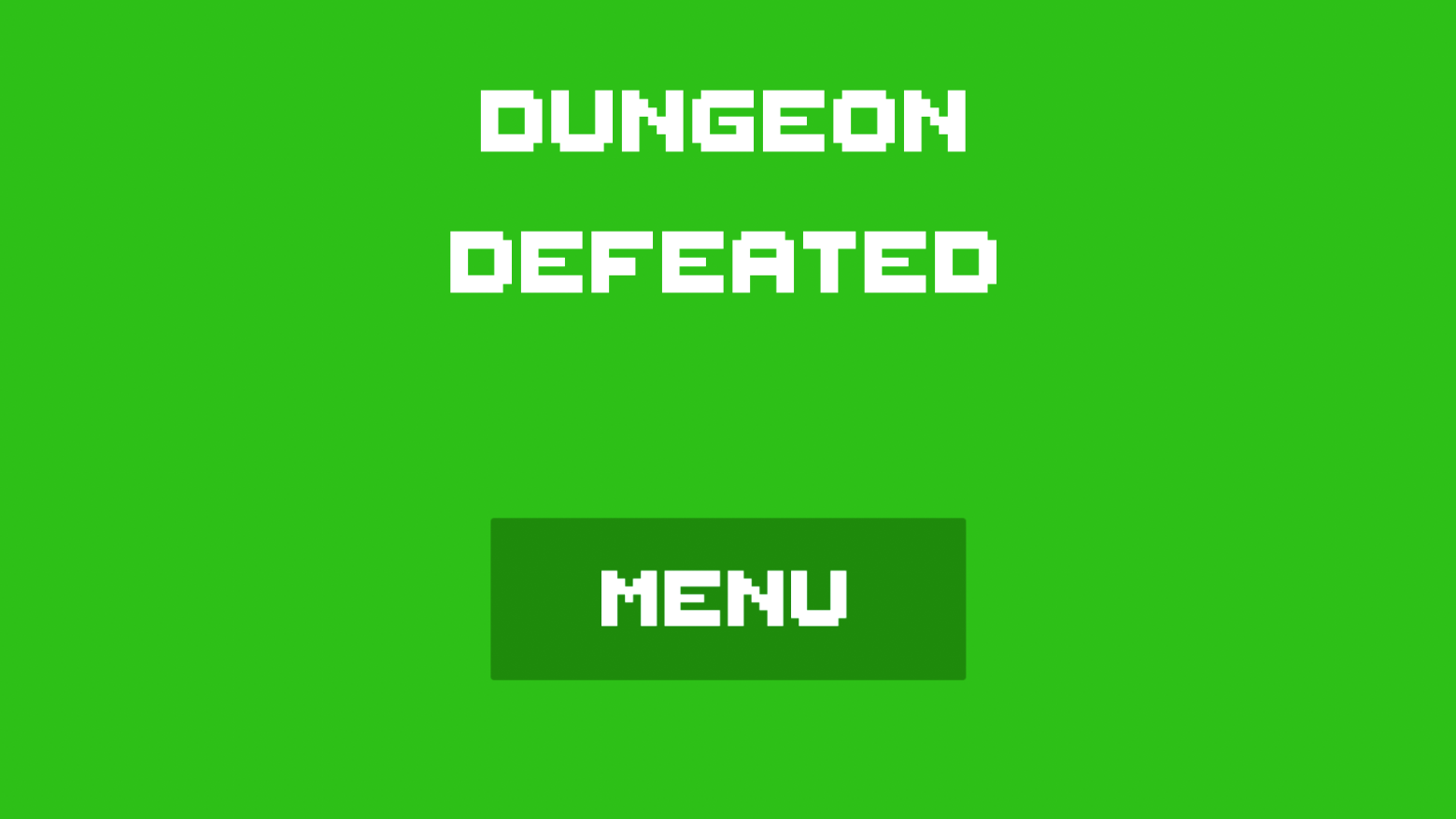


Рисунок 36 – Конец игры

3.5 Критерии качества программной системы

1. Портативность — так как данная программа реализована в среде Unity, несложно портировать игру на Android, iOS, Windows 10, Linux, Xbox, WebGL.
2. Легкость освоения — так как игра представляет собой классического представителя жанра Roguelike, управление не является сложным и в целом игра проста в освоении.
3. Производительность — данное приложение использует 2D графику в стиле Pixel Art, что положительно сказывается на производительности системы, поэтому системные требования минимальны.
4. Удобство сопровождения — данное приложение можно без особых усилий модифицировать, так как подсистемы реализованы в виде отдельных модулей.
5. Надёжность — программа полностью выполняет возложенные на неё функции и безотказно функционирует при любых характеристиках системы.

ВЫВОДЫ

В ходе выполнения курсов работы был глубже изучен современный подход к программированию на основе объектно-ориентированной технологии, были приобретены навыки написания приложения. Для достижения цели на разных этапах проектирования были решены следующие задачи:

1. Разработаны программные документы, в соответствии с действующими стандартами — оформлен отчет по работе, содержащий диаграммы, рисунки и описание работы.

2. Произведено тестирование и отладка программы — приложение не закрывается без команды пользователя, процесс игры предсказуем, скрипты не содержат синтаксических ошибок.

3. Написан текст разработанных классов на выбранном языке;

4. Абстрагированы, выделены классы и их иерархии — получены классы и иерархии, позволяющие реализовать проект согласно ПЗ.

5. Выбран вариант задания, язык реализации, детализирована постановка задачи — выбран вариант игрового проекта, среда разработки Unity и язык программирования C#.

Результатом работы стала игра в жанре Roguelike, подразумевающая процедурную генерацию уровней. На выбор игроку предоставляется три персонажа с различными способностями и параметрами. Игровой процесс заключается в уничтожении врагов, сражении с боссом и поиске перехода на следующий уровень. Игрок может подбирать предметы с врагов, такие как: монеты и склянки с зельем, восстанавливающим здоровье. HUD содержит шкалу здоровья, количество монет, мини-карту и информацию о способностях. Реализовано меню паузы с возможностью перезапуска уровня и выхода в главное в меню. Главное меню содержит следующие пункты: меню выбора персонажа, меню настроек и кнопку выхода из игры.

ПЕРЕЧЕНЬ ССЫЛОК

1. Hocking, J. Unity in Action: Multiplatform game development in C# / J. Hocking. – New York, United States : Manning Publications, 2018. – 400 с.
2. Richter, J. CLR via C# / J. Richter. – Redmond, United States : Microsoft Press, U.S., 2013. – 896 с.
3. Торн, А. Искусство создания сценариев в Unity / А. Торн. – Москва : ДМК-Пресс, 2016. – 360 с.
4. Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – Санкт-Петербург : Питер, 2008. – 464 с.
5. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл. – Москва : Русская редакция, 2017. – 896 с.
6. Schildt, H. C# 4.0 The Complete Reference / H. Schildt. – New York, United States : McGraw-Hill Education, 2010. – 976 с.
7. Полное руководство по языку программирования С# 7.0 и платформе .NET 4.7 [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/tutorial/. – Дата доступа: 24.05.2019.

ПРИЛОЖЕНИЕ А

Код программы