|  |
| --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** |
| **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** |
| **«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»** |

|  |  |
| --- | --- |
| Институт информационных технологий и управления в технических системах | |
| (полное название института) |
|  |
| кафедра «Информационные системы» | |
| (полное название кафедры) |

|  |  |  |
| --- | --- | --- |
| **Пояснительная записка** | | |
|  | | |
| к курсовой работе | | |
|  | | |
| на тему | Разработка игры «Cosmo Defense» в жанре 2D  «Tower Defense» | |  |
|  |  | |
| по дисциплине | | «Объектно-ориентированное программирование» |  |
|  | |  |  |
|  | | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Выполнил: студент | II | курса, группы | | ИС/б-21о | |  | | | | | | Направления подготовки (специальности) | | | | 09.03.02 | |  | | | | | | Информационные системы и технологии | | | | | | (код и наименование направления подготовки (специальности)) | | | | | |  | | | | | | профиль (специализация) | | | Информационные системы и технологии | | |  | | | | | |  | | | | | |  | | | | | | (фамилия, имя, отчество студента) | | | | | |  | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Дата допуска к защите « | | |  | » |  | | | | | 20 | 17 | г. |
|  | | |  |  |  | | | | |  |  |  |
|  | | | | | | | | | | | | |
| Руководитель | |  | | | | | |  |  | | | |
|  | (подпись) | | | | | | |  | (инициалы, фамилия) | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| 20 | | | | | | 17 | г. | | | | | |

АННОТАЦИЯ

Пояснительная записка к курсовому проекту по дисциплине «Основно-ориентированное программирование» для студентов направления 09.03.02 – «Информационные системы и технологии».

Пояснительная записка создана с целью описания программы, написанной в соответствии с поставленной задачей курсового проектирования на тему «Игра в жанре 2D «Tower Defense»». Данная программа представляет компьютерную игру «Cosmo Defense».

В пояснительной записке содержится введение, 3 основных раздела и заключение.

В основных разделах описываются этапы проектирования и программирования проекта.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc485961956)

[1 ПОСТАНОВКА ЗАДАЧИ 7](#_Toc485961957)

[2 ПРОЕКТНОЕ РЕШЕНИЕ 8](#_Toc485961958)

[2.1 Описание жизненного цикла игрового приложения 8](#_Toc485961959)

[2.2 Взаимодействие пользователя с программой 9](#_Toc485961960)

[2.3 Абстрагирование и выделение объектов. Построение иерархии классов 11](#_Toc485961961)

[2.4 Логика функционирования программы 14](#_Toc485961962)

[2.5. Логика поведения игровых объектов 16](#_Toc485961963)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 19](#_Toc485961964)

[3.1 Обоснование выбора средств программирования 19](#_Toc485961965)

[3.2 Описание структуры программы 19](#_Toc485961966)

[3.3 Описание реализации основных классов 23](#_Toc485961967)

[3.5 Критерии качества программной системы 32](#_Toc485961968)

[ЗАКЛЮЧЕНИЕ 33](#_Toc485961969)

[СПИСОК ЛИТЕРАТУРЫ 34](#_Toc485961970)

ВВЕДЕНИЕ

Целью курсового проектирования является закрепление навыков объектно-ориентированного проектирования и программирования и разработка программной системы.

В рамках данного курсового проектирования ведется разработка программного продукта под названием «Cosmo Defense» в соответствии с техническим заданием, утверждённым Севастопольским национальным техническим университетом.

Объектно-ориентированный подход является одним из самых популярным при решении практических задач. Его актуальность объясняется низкими затратами на разработку сложных систем, необходимость которых возникает при наличии в модели большого количества объектов и предполагает наличие сложных связей между ними.

Разрабатываемый программный продукт является продуктом игрового характера. Разработка продукта велась с использованием объектно-ориентированной методологии.

Актуальность разработки игрового программного обеспечения связана с тем, что в течение последнего года, показатель покупок игровых приложений вырос в разы. Эти данные постоянно увеличиваются, и в настоящее время статистика не меняется.

В разделе «Проектное решение» описывается ход проектирования объектной иерархии, информационной модели и жизненного цикла программы, и объектов. В разделе «Программная реализация» содержится описание особенностей программной реализации разрабатываемого продукта, описание интерфейса и критерии качества данного программного продукта.

1 ПОСТАНОВКА ЗАДАЧИ

**1.1 Цель разработки**

Разработать игровую систему на базе объектного подхода, получить навыки разработки объектно-ориентированных проектов.

**1.2 Описательная постановка задачи**

В системе моделируется оборона боевых позиций от сил врага. Действия происходят в атмосфере космо-фентези.

Силы врага представляют собой юниты (космические корабли), Следующие к порталу. Их цель – добраться до точки назначения. Юнит обладает набором характеристик: скорость, здоровье и цена.

Игрок управляет застройкой боевых позиций. Он должен строить башни (корабли своего флота). Башни ведут огонь по юнитам. Юниты появляются в специальной точке, которая называется «Спавн», волнами. Каждая волна содержит 10-50 существ одного типа. Всего 3 волны.

Должно быть предусмотрено три вида обычных башен: стрелковая, замедляющая, парализующая.

Стрелковая башня стреляет по идущим юнитам обычными снарядами. Обладает высокой скоростью атаки. Замедляющая башня стреляет снарядами с замедлением, которые имеют способность наносить урон постепенно и замедляют противника на некоторое время. Парализующая башня наносит стандартный урон, но парализует юнит противника, заставляя его стоять на месте некоторое время.

За убийство юнитов игрок получает золото, которое может тратить на постройку новых башен.

Игрок имеет 10 жизней и за каждое существо, дошедшее до пункта назначения, игрок теряет одну жизнь.

Также должно быть реализовано главное меню и возможность поставить игру на паузу. Игровое поле должно содержать краткую статистику состояния игрового процесса (количество золота, количество убитых юнитов, число оставшихся жизней), а также кнопки постройки новых башен и кнопку готовности к новой волне сил врага.

Входными данными является ввод с клавиатуры и мыши – размещение башен на карте и выбор пунктов меню, нажатие кнопок. Выходные данные выводятся на экран в игровое окно.

2 ПРОЕКТНОЕ РЕШЕНИЕ

**2.1 Описание жизненного цикла игрового приложения**

Жизненный цикл игрового приложения – это непрерывный процесс, который начинается с момента принятия решения о необходимости создания игрового приложения и заканчивается в момент его полного изъятия из эксплуатации [1]. В рамках курсового проектирования данный процесс начинается с получения задания на разработку и заканчивается по выполнению данного задания, однако в дальнейшем готовое приложение можно смело сопровождать, добавляя новый функционал.

Диаграмма, представленная на рисунке 2.1, отражает этапы при разработке игрового приложения.

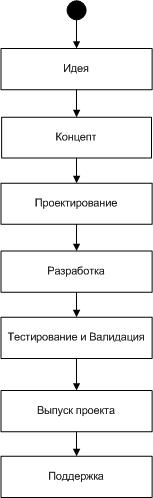


Рисунок 2.1 – Жизненный цикл игрового приложения

Итак, на первом этапе мы сталкиваемся с необходимостью определить конечную цель, то есть найти ответ на вопрос о том, что конкретно необходимо получить по завершению разработки.

После определения основной цели исходя из идеи строится концепт проекта для выбора дальнейшего варианта проектирования. На этом этапе продумывается основная логика взаимодействия в игре.

После того как концепт создан на его основе строится процесс детального проектирования. Крупные фирмы уделяют этому пункту большое внимание, так как при дальнейшей постройке и валидации кода могут появиться ошибки, полученные на этапе проектирования.

Процесс разработки занимает небольшую долю времени из всего времени на разработку игры так как на предыдущем этапе создается полноценный проект (проектное решение), на основе которого можно использовать различные CASE средства или системы автоматического программирования.

Этап тестирования и валидации очень важен, так как на этом этапе написанный ранее код проходит проверку на соответствие проектному решению. Если на этом этапе обнаруживается ошибка, это означает что в проектном решении допущена ошибка и её надо устранить.

Дальше проект выходит в массовое пользование. Собрав определенную статистику от пользователей, компания разработчик может дополнить проект выпустив обновление или дополнение к нему.

**2.2 Взаимодействие пользователя с программой**

На рисунке 2.2 представлена диаграмма взаимодействия, позволяющая рассмотреть возможности пользователя относительно работы с разрабатываемым игровым приложением.

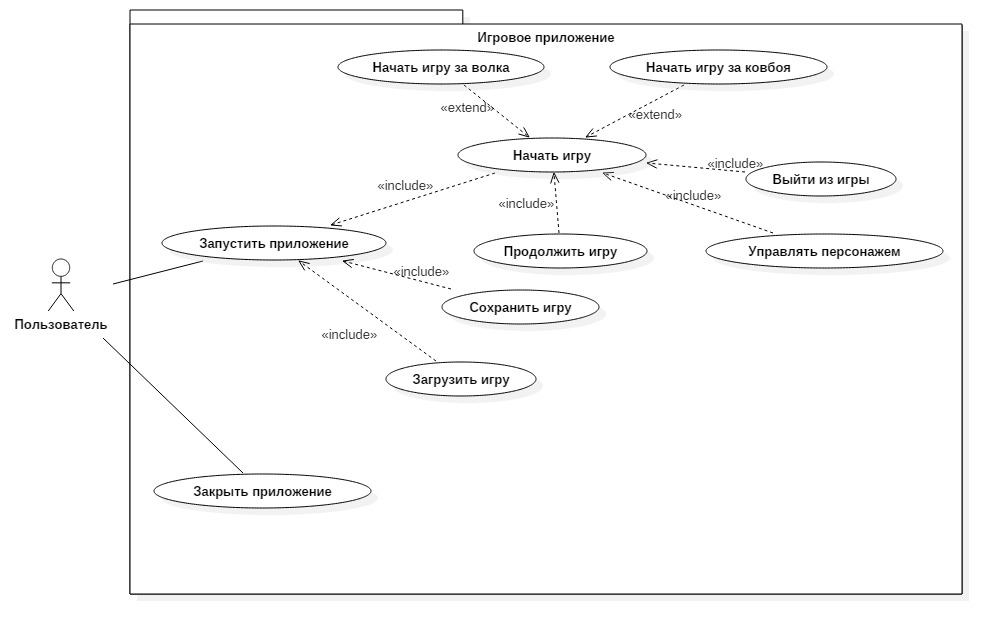


Рисунок 2.2 – Диаграмма прецедентов

Как можно увидеть из данного рисунка, запустив приложение, пользователь имеет возможность произвести следующий ряд действий:

Начать новую игру. Выбирая эту опцию в меню, пользователь переходит к выбору персонажа, после чего – к управлению им.

Продолжить игру. Под этой опцией понимается возобновление игрового процесса после того, как пользователь прерывает его посредством нажатия клавиши Esc, которое приводит к отображению меню. Таким образом, при первоначальном запуске приложения эта опция является неактивной.

Сохранить/загрузить игру. В данном приложении реализованы сериализация и десериализация состояния игрового процесса, поэтому у пользователя есть возможность сохранить свои результаты и, к примеру, в случае проигрыша, вновь подгрузить данные о последней сохраненной точке в игре и продолжить прохождение с данного места.

Также пользователь может просто завершить работу с приложением. В этом случае будет выведено окно, запрашивающее подтверждение этого действия.

**2.3 Абстрагирование и выделение объектов. Построение иерархии классов**

Обращаясь к игровому миру и его содержимому, можно выделить некоторых персонажей и рассматривать их, как отдельные объекты со своими свойствами и поведением. В игре жанра «Tower Defense» основными объектами могут быть башни, юниты, снаряды и спавны. Однако у всех этих объектов есть некоторые общие свойства, такие как, например, позиция или состояние (живы или мертвы), а также схожесть в поведении: движения, взаимодействие с окружающим миром.

Таким образом, можно определить некоторый суперкласс «Графический объект», от которого будут наследоваться отдельные классы, описывающие более детальную логику их обращения и реализации.

Диаграмма классов, описывающих персонажей, представлена на рисунке 2.3.



Рисунок 2.3 – Диаграмма классов игровых персонажей

Класс Tower заключает в себе функциональность башен пользователя. К его функциональности можно отнести то, как они непосредственно ведет себя в игре и в отношении другого окружения и противников.

Класс Weapon реализует всю функциональность базхового снаряда. Под функциональностью можно понимать его движения (следолвание за противником), попадание и нанесение урона. Если у снаряда был какой либо отличный от базового модификатор, то используется логика в зависимости от этого модификатора.

Enemy – класс, реализующий функциональность противника, который передвигается непосредственно от точки спавна в сторону портала, без возможности выйти за пределы своей территории.

Портал – статический элемент на игровой карте, к которому стремятся все противники. Портал обладает свойством количества пройденных юнитов. Если это значение станет равным максимальному или более то игра завершиться поражением.

Также, помимо персонажей, можно выделить компоненты игрового интерфейса. Они имеют достаточное количество общих атрибутов и логика их реализации во многом совпадает, поэтому определим для них суперкласс «InterfaceElement». Для реализации же мира игры выделим класс «Map», который тем или иным образом будет взаимодействовать с классами, описывающие некоторые объекты игрового мира (враги, портал). Данный класс несет в себе крайне весомый функционал для реализации разрабатываемого игрового приложения, так как именно он отвечает за логику следования врагов, счетчик жизней портала. Также возникает потребность в классах, которые будут отвечать за постройки пользователя (Player) и реализацию интерфейса (GameInterface). Диаграмма классов, описывающих игровой уровень, представлена на рисунке 2.4.

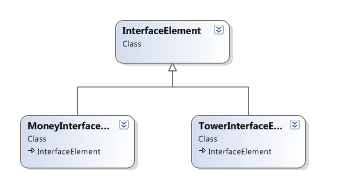


Рисунок 2.4 – Диаграмма классов, описывающих части игрового интерфейса

Класс Camera следит за центром игрового экрана.

Общая диаграмма классов отображена на рисунке 2.5.

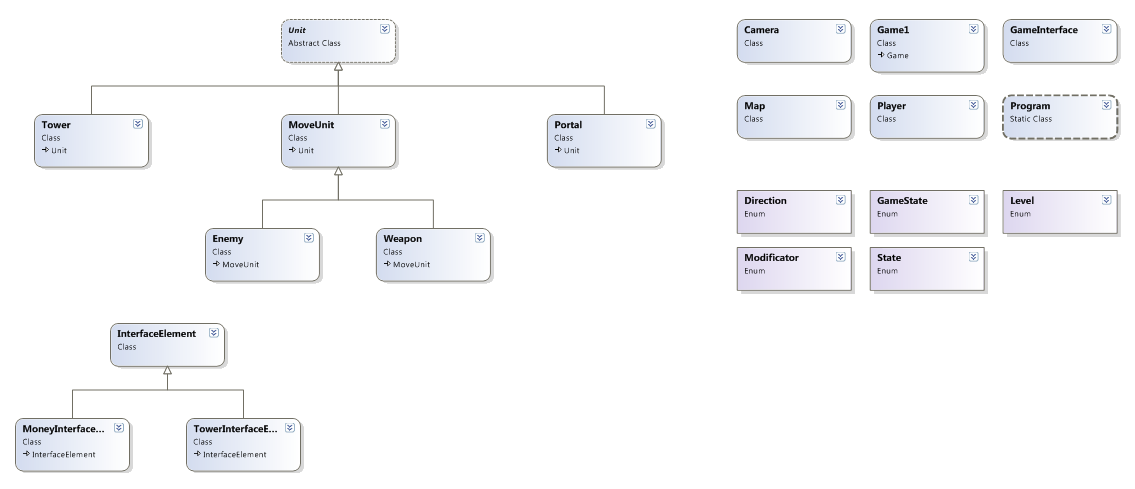


Рисунок 2.5 – Общая диаграмма классов

**2.4 Логика функционирования программы**

На рисунке 2.6 представлена логика функционирования программы относительно действий пользователя.

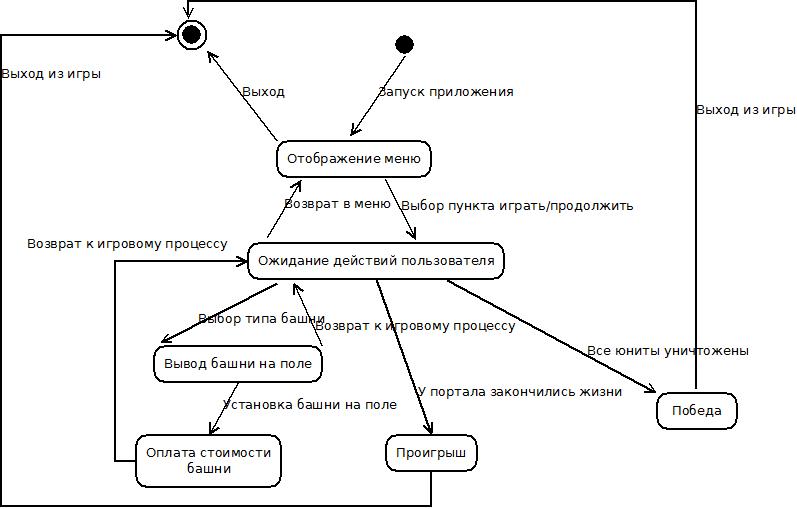


Рисунок 2.6 – Диаграмма состояний игрового процесса

Обращаясь к диаграмме, изображенной на рисунке 2.6, можно увидеть что первым делом, после запуска приложения пользователем, в программе происходит отрисовка меню, предоставляющего такие пункты для выбора: новая игра и выйти.

В случае, если пользователь выберет первый пункт меню, а именно «новая игра», программа перейдет в состояние игры и появиться игровой интерфейс пользователя.

Для взаимодействия с игрой пользователь должен размещать на заданном игровом поле свои башни, для уничтожения врагов надвигающейся волны. Пользователь может выбрать тип устанавливаемой башни, после чего разместить её на поле, потратив средства на постройку. В последствии после размещения башня будет выбирать случайного врага в радиусе и стрелять по нему.

Также может возникнуть ситуация, когда количество счетчик жизней портала будет равен нулю. Тогда на экран будет выведено изображение, символизирующее проигрыш.

**2.5. Логика поведения игровых объектов**

В данном подразделе рассматривается логика поведения игровых объектов.



Рисунок 2.7 – Диаграмма состояний снаряда

Рисунок 2.8 – Диаграмма состояний башни

Рассмотрим данные диаграммы детальней. На рисунке 2.7 представлена диаграмма состояний управляемого героя. Как можно увидеть, его поведение полностью определяется действиями пользователя. Поэтому из своего начального состояния он переходит в состояние ожидания нажатия клавиши. В зависимости от того, какая из кнопок была нажата, персонаж может попасть в два состояния: если нажата клавиша перемещения, то проверяется не выходит ли персонаж за границы уровня и не пересекает ли какой-либо из блоков, если же нажата клавиша для прыжка, то происходит проверка того, не находится ли персонаж уже в состоянии прыжка. В первом случае, если препятствия отсутствует, то координаты героя смещаются, иначе он снова переходит в состояние ожидания нажатия клавиши. Во втором случае, если персонаж еще не совершил прыжок, то по аналогии проводится проверка на наличие каких-либо препятствий и выполнение прыжка при их отсутствии, иначе же – переход в состояние ожидания.

Рассмотрим диаграмму противника, который передвигается лишь в своей ограниченной локации (рисунок 2.8). Из своего начального состояния противник данного типа переходит в состояние проверки пересечения с блоками и границами уровня или своей локации. Если ничего не мешает дальнейшему передвижению, то его координаты смещаются, иначе противник просто меняет направление движения.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

**3.1 Обоснование выбора средств программирования**

Для реализации поставленной задачи было принято решение выбрать в качестве языка программирования С# и при этом использовать технологии XNA Game Studio 4.0. Основными факторами, повлиявшими на выбор данного языка, явились его следующие преимущества:

* удобство разработки;
* скорость работы;
* безопасность и защищенность;
* хорошая читаемость кода и документированность языка;
* кроссплатформенность.

С целью избегания многих технических трудностей, возникающих при написании кода, дополнительно был установлен фреймворк XNA. Конвейер содержимого – одна из самых интересных возможностей XNA, предоставляемых разработчикам приложений [3]. В играх, разрабатываемых без технологий XNA, скорее всего, пришлось бы беспокоиться о методе загрузки игрового содержания (такого как, например, звуки и графика), а также о том, где оно находится содержание, и как программа должна его читать. Конвейер содержания упрощает обработку всего игрового содержания, тем самым облегчая процесс разработки.

**3.2 Описание структуры программы**

Игра реализована с использованием взаимосвязанных между собой классов, которые предоставляют ту или иную функциональность, необходимую для реализации поведения персонажа, отображения меню и всех других разнообразных объектов в игровом приложении в целом. Платформа XNA Game Studio, в свою очередь, предоставляет готовую к использованию базовую структуру игры.

Программная среда загружает контент, обрабатывает и объединяет его, а графический результат выводит на экран монитора. Данная программная реализация предполагает, что подгрузка, обработка и вывод графического 2D изображения на экран выполняется 60 раз в секунду посредством вызова методов Draw и Update, которые будут рассмотрены ниже по тексту.

На рисунке 3.1 представлена структура работы программы.



Рисунок 3.1 – Структура работы программы

Типичные классы для каждого приложения, реализованного с использованием XNA:

* Program. В данном классе находится точка входа XNA-приложения – метод Main, в котором содержится лишь две строки: создание объекта типа Game1 и вызов метода Run() этого объекта для запуска игрового цикла.
* Game1. Данный класс отвечает за общую инициализацию. В нем определяется объект, который будет ссылаться на диспетчер графических устройств, а также объект SpriteBatch, используемый для рисования текста и двумерных изображений. Game1 наследуется от класса Game и реализует следующие методы:
* protected override void Initialize() – инициализация игры;
* protected override void LoadContent() – загрузка графических ресурсов;
* protected override void UnloadContent() – освобождение графических ресурсов;
* protected override void Update(GameTime gameTime) – чтение пользовательского ввода, выполнение вычислений, проверка завершения игры;
* protected override void Draw(GameTime gameTime) – код визуализации.

Также для реализации приложения разработаны классы, описанные выше.

Также стоит отметить, что проект состоит из двух частей. Первая – это реализация различных классов, вторая же – контент игрового приложения. Под контентом здесь понимается следующее:

* спрайты главных героев и их противников;
* изображения блоков и различных фонов;
* звуковые файлы;
* файлы шрифтов.

**3.3 Описание реализации основных классов**

Основные классы игрового приложения описаны в подразделе 2.3. В данном же подразделе рассмотрим то, как они реализованы.

Практически в каждой игре, особенно двумерной, самыми важными компонентами являются так называемые «спрайты» и их поведение на игровом поле, в том числе относительно других объектов. Разрабатываемая игра не является исключением и активно использует спрайты для прорисовки всех основных объектов.

Спрайт является совокупностью изображений, представляющих собой одно полноценное изображение [4]. Для лучшего восприятия того, что же такое спрайт, его структуру можно увидеть на рисунке 3.3. Он отражает поведение одного из главных героев игры (волка) при движении. За представление того, как персонаж будет выглядеть в режиме ожидания, отвечает спрайт, изображенный на рисунке 3.4. Спрайт, отражающий прыжок, представлен на рисунке 3.5.



Рисунок 3.3 – Спрайт главного персонажа (движение)



Рисунок 3.4 – Спрайт главного персонажа (режим ожидания)

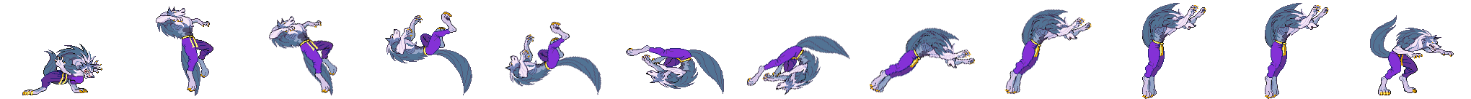


Рисунок 3.5 – Спрайт главного персонажа (прыжок)

Однако спрайты – это лишь набор картинок, а вопрос о том, как они будут отображаться в игре и сменять одна другую, куда сложнее.

Рассмотрим особенности класса GraphicUnit и, в частности, то, какие методы нужны для работы со спрайтами.

Для анимации нам нужно три текстуры и один атрибут типа Rectangle:

* Texture2D realty – пассивное состояние;
* Texture2D movement – движение;
* Texture2D jump – прыжок;
* Rectangle rectangle - прямоугольник для текстур, необходимый для управления ими.

Также для данного класса определены следующие поля:

* SpriteEffects effects – атрибут, необходимый для реализации отображения спрайта по горизонтали;
* int frameHeight – высота одного кадра анимации;
* int frameWidth – ширина одного кадра анимации;
* int currentFrame – номер текущего кадра анимации;
* int timer – время, прошедшее с момента отображения первого кадра анимации;
* int interval – интервал между кадрами (интервал показа каждого кадра анимации);
* float speed – скорость передвижения персонажа;
* bool stateAnimation – состояние персонажа: пассивное/активное;
* bool animationRight – направление движения персонажа.

Для работы со спрайтами определены следующие методы:

* public int FrameCount() – подсчет количества кадров анимации;
* public void MovementFrame(GameTime gameTime) – увеличение кадра анимации на один, в случае движения объекта.

Рассмотрим небольшой аспект физики игры, а именно гравитацию, действующую на персонажа во время его прыжка. Для реализации прыжка нам нужна текстура, представленная на рисунке 3.5 и некоторые атрибуты:

* float ySpeed – «вертикальная скорость» прыжка;
* float maxYSpeed – максимальная «вертикальная скорость» прыжка;
* float gravity – сила тяжести, действующая на персонажа во время прыжка;
* bool stateJump – состояние персонажа: в прыжке/не в прыжке.

Перейдем к основным методам, реализующим прыжок и гравитацию.

* public virtual void Jump() – смена состояния на «прыжок» в случае, если он еще не был совершен;
* public void Gravity(GameTime gameTime) – метод, отвечающий за прыжок (работа с гравитацией).

Также для персонажа необходимо определить методы, отвечающие за его передвижение.

* public void Run(bool right) – смена состояния персонажа на "бег";
* public void VacationStop() – смена состояния персонажа на "отдых";
* public void BorderScreen() – метод, передвигающий персонажа, а точнее той «коробки», в которую он помещается;

public Rectangle GetBoundingRect(Rectangle rectangle) – обрезка прямоугольника, в который помещается персонаж.

Рассмотрим класс Level, который является родителем классов, отвечающих за работу с объектами, отображаемыми на уровне.

Некоторые из его методов:

* public Rectangle GetScreenRectangle(Rectangle rectangle) – Перевод из «глобальных» координат в «локальные». «Глобальными» будем считать те, которые не помещаются на экран (то есть всю рабочую область), «локальными» – те, которые ограничены размерами экрана;
* public void Scroll(int dx) перемещение экрана относительно начала.

**3.4 Интерфейс пользователя**

Для запуска данного игрового приложения компьютер пользователя должен отвечать минимальным техническим характеристикам программы, а также иметь установленное дополнительное программное обеспечение, такое как Microsoft .NET Framework, Microsoft XNA Framework Redistributable 4.0. и DirectX.

Итак, при запуске игры появляется окно, которое можно увидеть на рисунке 3.6. Для продолжения пользователю необходимо нажать произвольную клавишу, при этом начнется первая стадия по подготовке к началу игры.



Рисунок 3.6 – Экран при первоначальном запуске игры

После нажатия любой клавиши, на изображение, составленное из мелких шариков, начинает действовать гравитация и он начнет падать вниз. Через некоторое время появится надпись «Press SPACE to Start» (рисунок 3.7), то есть для продолжения пользователю теперь требуется нажать пробел.



Рисунок 3.7 – Предложение нажать «пробел» для продолжения

После нажатия пользователем всех необходимых клавиш, на экране появляется главное меню игры (рисунок 3.8), состоящее из следующих пунктов:

* New game – начать новую игру;
* Resume – вернуться к игре;
* Save – сохранить игру;
* Load – загрузить игру;
* Exit – выйти из игры.



Рисунок 3.8 – Меню игры

При нажатии пункта меню New game появляется окно выбора персонажа (рисунок 3.9). Здесь пользователю также требуется ввести свой ник в соответствующее поле, после чего при помощи курсора выбрать конкретного героя. Если же не ввести никнейм и при этом щелкнуть на изображение персонажа, пользователь увидит сообщение об ошибке (рисунок 3.10).



Рисунок 3.9 – Выбор персонажа



Рисунок 3.10 – Ошибка: пользователь не ввел данные

При соблюдении всех инструкций начинается сам процесс игры (рисунок 3.11). Пользователь получает возможность управления главным персонажем для того, чтобы собирать предметы на уровне и избегать противников. При этом, если управляемый персонаж столкнется с одним из противников, игра на текущем уровне начнется с самого начала. Для того, чтобы пройти первый уровень, необходимо собрать на нем все подбираемые предметы, не столкнувшись при этом с противником. После того, как будет набрано максимальное количество очков в текущем уровне, пользователь сможет начать прохождение второго, структура которого изображена на рисунке 3.12.



Рисунок 3.11 – Первый уровень игры



Рисунок 3.12 – Второй уровень игры

Во время игры пользователь имеет возможность приостановить ее посредством нажатия клавиши «Esc» на клавиатуре. Пауза в игре предполагает открытие меню (рисунок 3.13). Однако в данном случае пункты меню «Resume» и «Save», которые являлись неактивными при первом запуске, становятся доступными для нажатия. Для возврата к игре достаточно выбрать пункт «Resume».



Рисунок 3.13 – Открытие меню при паузе в игре

Когда же пользователь соберет все предметы на всех доступных уровнях, он увидит окно с поздравительным текстом, сообщающим о том, что игра пройдена (рисунок 3.14). Нажимая клавишу «Esc», пользователь снова оказывается в главном меню.



Рисунок 3.14 – Вид экрана при прохождении игры

Завершить игру можно в главном меню либо посредством выбора пункта меню «Exit», либо же нажатием «Esc». При этом пользователь увидит окошко, запрашивающее подтверждение данного действия (рисунок 3.15).



Рисунок 3.15 – Диалоговое окно при выходе из игры

**3.5 Критерии качества программной системы**

Функциональная пригодность:

Данное приложение является игрой в жанре 2D , что полностью удовлетворяет функционалу поставленной задачи.

Надежность:

При тестировании данного продукта возникали ошибки, но в последующем были устранены. При финальном тесте критичных ошибок выявлено не было.

Корректность:

Как уже было отмечено ранее, критичных ошибок в данном продукте не выявлено, но иногда можно столкнуться с некоторыми багами (к примеру, персонаж может застрять в текстуре), которые, скорее всего, приведут к тому, что игровой процесс придется перезапустить.

Сопровождаемость:

Обеспечена модульность данного продукта за счет разбиения сущностей по классам. Это придает приложению гибкость и возможность изменять какие-либо составляющие. К примеру, в дальнейшем могут быть добавлены новые персонажи, новые типы противников, различные объекты на уровне, количество уровней. В общем и целом, игра полностью готова к сопровождению.

Применимость:

Управление в данной игре каждый способен освоить интуитивно, оно достаточно тривиально и не должно вызывать затруднений.

Эффективность:

Игровой программный продукт будет показывать высокую производительность при наличии следующей рекомендуемой конфигурации для пользователя и разработчика:

* настольный ПК
* Windows 7, Windows Vista, Windows XP
* процессор Intel или AMD, 1.5 GHz или лучше
* встроенная видеокарта Intel или NVIDIA Geforce
* 1 Гб ОЗУ
* установенное дополнительное ПО (Microsoft .NET Framework, Microsoft XNA Framework Redistributable 4.0. и DirectX).

ЗАКЛЮЧЕНИЕ

Современный мир информационных технологий трудно представить себе без такого программного продукта как компьютерные игры. Поэтому без сомнений можно утверждать, что вопрос их написания является актуальным.

В связи с вышесказанным, темой курсового проекта была выбрана разработка игры в жанре 2D-платформер. В итоге был спроектирован и реализован весь задуманный функционал. Для этого, в первую очередь, были проанализированы общие требования к программному продукту и разработана архитектура будущего приложения, после чего была разработана диаграмма классов, визуально демонстрирующая зависимости между ними. Также для лучшего понимания были разработаны диаграммы состояний персонажей и диаграмма прецедентов.

Этап разработки был самым объемным, в ходе которого функционал программы постоянно инкрементировался. Сначала, был подобран графический материал (спрайты, фоны, шрифты и т.д.), а также музыкальное сопровождение, после чего подгружен в разрабатываемое приложение. Затем была реализована игровая физика персонажей и методы обработки их спрайтов. Параллельно с данным функционалом разрабатывался интерфейс главного меню.

Был приобретен опыт разработки игрового приложения, работы в команде из двух человек, применения основных этапов итеративных методологий для достижения эффективного, производительного и минимизирующего время разработки результата.

В дальнейшем развитии данного проекта будет идти работа по расширению и совершенствованию функциональных возможностей и игровой логики, для того, чтобы пользователи могли получать еще больше игровых возможностей.

СПИСОК ЛИТЕРАТУРЫ

1. Скопин, И. Н. Модели жизненного цикла программного обеспечения // sntbul.bmstu.ru: Виртуальный компьютерный музей [Электронный ресурс] URL: http://www.computer-museum.ru/books/n\_collection/models.htm (дата обращения: 05.04.2017)

2. Программирование на C, C# и Java // vscode.ru: Технология разработки программного обеспечения [Электронный ресурс] URL: <http://vscode.ru/articles/tehnologiya-razrabotki-po.html> (дата обращения: 05.04.2017)

3. Горнаков, С.Г. Разработка игр под Windows в XNA Game Studio Express/ С.Г. Горнаков. – Москва: ДМК Пресс, 2008. – 384 с.

4. C# Game Programming: For Serious Game Creation. Boston: Course Technology PTR, 2010. – p. 439