**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc40809808)

[1 ТЕХНИЧЕСКИЕ СРЕДСТВА РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ 5](#_Toc40809809)

[1.1 Графическая библиотека DirectX 5](#_Toc40809810)

[1.2 Графическая библиотека OpenGL 6](#_Toc40809811)

[1.3 Отличия графических библиотек OpenGL и DirectX 8](#_Toc40809812)

[1.4 Игровой жанр «Платформер» 9](#_Toc40809813)

[1.5 Обзор существующих игр жанра «Платформер» 9](#_Toc40809814)

[1.6 Технические требования к курсовому проекту 14](#_Toc40809815)

[2 АРХИТЕКТУРА ИГРОВОГО ПРИЛОЖЕНИЯ «ПЛАТФОРМЕР» 15](#_Toc40809816)

[2.1 Архитектура игрового приложения 15](#_Toc40809817)

[2.2 Структура игрового приложения «Платформер» 16](#_Toc40809818)

[2.3 Обзор взаимодействия классов игрового приложения 1](#_Toc40809818)8

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «ПЛАТФОРМЕР» 21](#_Toc40809819)

[3.1 Реализация игрового приложения «Платформер» 21](#_Toc40809820)

[3.2 Пользовательский интерфейс 29](#_Toc40809821)

[ЗАКЛЮЧЕНИЕ 33](#_Toc40809822)

[Список используемых источников 34](#_Toc40809823)

[Приложение А Листинг программы «Платформер» 35](#_Toc40809824)

[Приложение Б Иерархическая схема классов приложения 119](#_Toc40809825)

[Приложение В Руководство пользователя 120](#_Toc40809826)

[Приложение Г Руководство программиста 122](#_Toc40809827)

[Приложение Д Руководство системного программиста 123](#_Toc40809828)

## ВВЕДЕНИЕ

Актуальность игровых приложений уже не требует никаких доказательств, а их популярность растет с каждым днём. Едва ли в современном мире найдется хоть один человек, который не знал бы об этом чуде технического прогресса.

Индустрия компьютерных игр – одна из наиболее востребованных и быстро развивающихся отраслей компьютерных технологий. Разумеется, что с возникновением новых технологий, творческий потенциал игры становится возможным реализовать. Таким образом, с каждым годом на рынок выходит всё больше креативных и реалистичных видеоигр.

В основном, игровая индустрия считается развлекательной сферой, однако её развитие поспособствовало появлению важных симуляторов для обучения профессиям, спасающим жизни людей.

Казалось, что компьютерные игры всегда присутствовали в жизни людей, и трудно представить, что они появились только около 50-ти лет назад.

Первые игры считались уникальными, неповторимыми, их было достаточно просто отличить от других. Но со временем стало появляться много аналогов и схожих видеоигр, поэтому их принято классифицировать.

Существует большое количество подходов к классификации игр: по используемой платформе, по визуальному представлению, по количеству игроков, множество авторских подходов, а также по жанру.

Жанровая классификация игр стала активно развиваться в 90-е годы с появлением целых серий видеоигр. Одними из наиболее популярных считаются игры в жанрах:

* аркады;
* головоломки;
* симуляторы;
* стратегии;
* action;
* tower defense;
* платформеры;
* RPG;
* песочницы.

Из вышесказанного следует, что разработка компьютерных игр является актуальным и перспективным родом деятельности, а специалисты в этой области очень востребованными. В связи с этим, будет разрабатываться игровое приложение в одном из популярных жанров «Платформер» на языке программирования С# с использованием объектно-ориентированного подхода и графической библиотеки OpenGL.

## ТЕХНИЧЕСКИЕ СРЕДСТВА РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ

## 1.1 Графическая библиотека *DirectX*

Microsoft DirectX – набор API функций для реализация разнообразных задач связанных с программированием, который позволяет создавать визуальные и звуковые эффекты в мультимедийных приложениях. Широко используется для разработки игровых приложений. За счёт данного набора API функций увеличивается качество и производительность обработки игр, использующих 3D графику, анимацию, стереозвук и другие элементы мультимедиа, а также увеличивается безопасность и производительность системы в общем.

Современные технологии прогрессируют достаточно быстро. С каждой новой версией DirectX становится возможным добавление новых элементов в игры и оптимизация старых.

По сути, Microsoft DirectX представляет собой набор программных библиотек. В случае их отсутствия в системных папках, какие-то эффекты в играх не будут отображаться, а возможно и такое, что игра вообще не запустится. На данный момент, минимальный набор этих файлов присутствует в установленной Windows. А если вдруг нет, то его можно скачать бесплатно на официальном сайте Microsoft.

DirectX поддерживается видеокартами по такому принципу: чем новее видеокарта, тем выше версию графической библиотеки она поддерживает. Следует отметить, что все предыдущие версии всё так же будут поддерживаться. Однако нужно понимать, что от версии Windows тоже зависит, какая версия DirectX будет поддерживаться.

Практически все части DirectX API представляют наборы «объектной модели компонентов».

В целом, Microsoft DirectX подразделяется на следующие компоненты:

* Direct3D – это компонент, предоставляющий возможность отображает трехмерную графику;
* Direct2D – интерфейс вывода двухмерной графики;
* DirectDraw – компонент для отображения двухмерных визуальных эффектов;
* DirectSound – компонент для увеличения обработки звуковых эффектов в фильмах и компьютерных играх;
* DirectInput – интерфейс для обработки данных, поступающих с клавиатуры, мыши и джойстика;
* DirectPlay – интерфейс сетевой коммуникации игр;
* DirectMusic – интерфейс для воспроизведения музыки;
* DirectShow – интерфейс, используемый для ввода/вывода аудио и/или видео данных;
* DirectX Instruments – технология для синтеза звука;
* DirectSetup – компонент, который отвечает за установку DirectX;
* DirectX Media Objects – компонент, который реализует функциональную поддержку потоковых объектов.

В основе данной графической библиотеки функции для отрисовки, которые запускают графический конвейер. На рисунке 1.1 показаны стандартные блоки конвейера.

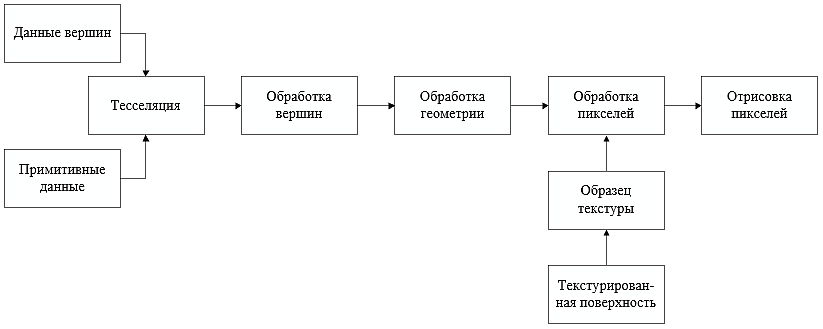


Рисунок 1.1 – Архитектура DirectX

Графический конвейер позволяет эффективно обрабатывать и отображать сцены Direct3D на дисплее, используя преимущества доступного оборудования.

## 1.2 Графическая библиотека *OpenGL*

OpenGL – спецификация, которой определяется программный интерфейс API для разработки приложений в области двумерной и трехмерной графики. Она включает в себя более 300 функций для отрисовки сложных сцен из простых примитивов. Графическая библиотека OpenGL используется для создания видеоигр, виртуальной реальности, а также для визуализации научных исследований.

Название библиотеки расшифровывается, как Open Graphics Library, что в переводе на русский означает «Открытая графическая библиотека». Ее открытость обосновывается тем, что она доступна для пользователя и разработчика без лицензионных отчислений.

OpenGL можно использовать с наиболее удобным языком, так для большинства языков есть привязки к этой графической библиотеке. Например, привязки OpenGL есть у C#, C++, Python, Java и у других.

Основные возможности библиотеки OpenGL:

* геометрические и растровые примитивы;
* использование В-сплайнов;
* видовые и модельные преобразования;
* работа с цветом (RGBA или выбор цвета из палитры);
* удаление невидимых линий и поверхностей;
* z-буферизация;
* одинарная и двойная буферизация;
* наложение текстуры;
* сглаживание пикселей;
* возможность задать источники света;
* атмосферные эффекты для наибольшей реалистичности, такие как дым, туман и другие;
* прозрачность объектов;
* использование списков изображений.

В основе OpenGL функции для отрисовки, которые запускают графический конвейер. На рисунке 1.2 показаны стандартные блоки конвейера.

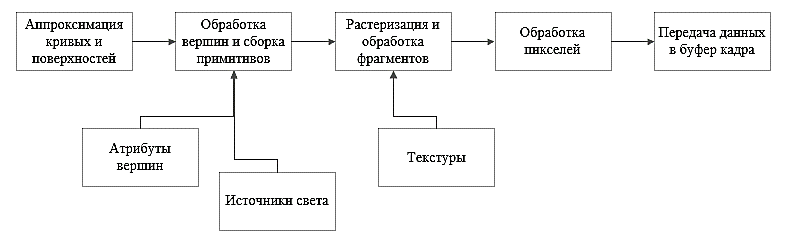


Рисунок 1.2 – Архитектура OpenGL

Функции OpenGL реализованы в модели клиент-сервер. Приложение выступает в роли клиента, так как оно обрабатывает команды, а сервер OpenGL интерпретирует и выполняет их.

На данный момент графическая библиотека OpenGL поддерживается на большинстве популярных аппаратных и программных платформах.

## 1.3 Отличия графических библиотек *OpenGL* и *DirectX*

Основные отличия рассматриваемых графических библиотек приведены в таблице 1.1.

Таблица 1.1 – Сравнение OpenGL и DirectX

|  |  |  |
| --- | --- | --- |
| Признаки для сравнения | OpenGL | DirectX |
| Операционная система | Кроссплатформенная библиотека, можно использовать в большинстве популярных ОС. | Операционная система Windows. |
| Библиотеки | У OpenGL только графическая библиотека. | DirectX содержит сразу несколько библиотек, а не только графическую, как OpenGL. В состав DirectX: основная графическая библиотека, вспомогательная графическая библиотека, библиотека для работы со звуком и другое. |
| Поддержка расширений | По сути, практически весь функционал OpenGL – расширения. Реализация OpenGL позволяет определять расширения к основной спецификации. | DirectX фиксирован в пределах одной мажорной версии. Какие-либо изменения или дополнения происходят только при выпуске следующей версии. |

Из данной таблицы следует, что обе библиотеки имеют свои преимущества и недостатки, но они не существенны. Выбор библиотеки нужно осуществлять, в зависимости от сферы применения API.

В разрабатываемом игровом приложении будет выбрана графическая библиотека OpenGL соответственно заданию.

## 1.4 Игровой жанр «Платформер»

Игры жанра «Платформер» характеризуются наличием разнообразных опор, называющиеся платформами, которые позволяют персонажу передвигаться по игровому миру. Как правило, герою игры нужно совершать прыжки, перескакивать от платформы к платформе, собирать предметы или сражаться с врагами, чтобы достигнуть конца уровня.

Большинство игр жанра «Платформер» обладают нереалистичной, мультипликационной графикой, поэтому чаще всего героями данного игрового жанра чаще всего являются персонажи из сказок и мультфильмов.

Начало 1980-х – период появления первых платформеров. Тогда игровые консоли были недостаточно мощными, чтобы отображать трехмерную графику. Такие игры в 3D появились только к 1990-м годам.

В играх данного жанра встречаются предметы, которые называются power-up, которые наделяют персонажа какими-либо временными способностями, например, ускорение, увеличение высоты прыжка и другое. Так же возможно получение коллекционных предметов и оружия. Противники могут быть многообразными в одной игре, различных типов, с различными способностями. Созданы они, разумеется, для интереса прохождений более сложных уровней. Чем выше уровень, тем сильнее противники. Игрок же должен уклоняться от их нападений, иногда даже защищаться в ответ. Если же игрок не станет уклоняться или защищаться, то его здоровье будет падать, или он вовсе умрёт.

Как правило, уровень пройти возможно при нахождении скрытого выхода, если выход не скрыт, тогда при нахождении загадочного ключа от этого самого выхода. Это всё так же создано для разнообразия и подогрева интереса игрока.

## 1.5 Обзор существующих игр жанра «Платформер»

Считается, что родоначальниками игр жанра «Платформер» является игры Space Panic и Apple Panic. Space Panic лишен возможности прыжка. Вместо этого персонаж роет ямы, в которые ему необходимо заманить пришельцев и нанести удар, чтобы выкинуть их из ямы и экрана, сохранив запас кислорода. Apple Panic – неавторизованная версия Space Panic для Apple II. Внешний вид данных игр представлен на рисунках 1.3-1.4.





Рисунок 1.3 – Внешний вид игры Space Panic



Рисунок 1.4 – Внешний вид игры Apple Panic

За ними последовала игра Donkey Kong (1981 год), аркадная игра, которая была создана фирмой «Nintendo». Вскоре процесс прохождения уровня, в основном, стал горизонтальным с длинным прокручивающимся миром. Считается, что начало этому положила игра Pitfall!, выпущенная фирмой Activision в 1982 году для консолей Atari 2600. Внешний вид игры с новым прокручивающимся миром представлен на рисунке 1.5.



Рисунок 1.5 – Внешний вид игры Pitfall!

Наиболее популярной и переворотной игрой для фирмы «Nintendo» стал платформер Super Mario Bros (1985 год). Игра была выпущена для приставки Nintendo Entertainment System. Она наполнена большими и сложными уровнями, став примером для последующих создателей игр. Внешний вид данной игры представлен нп рисунке 1.6.



Рисунок 1.6 – Внешний вид игры Super Mario Bros

Термин «трёхмерный платформер» обозначает или геймплей, включающий три измерения, или использование 3D полигонов в реальном времени для отрисовки уровней и героев, или и то и другое. Появление трёхмерных платформеров изменило конечные цели некоторых платформеров. В большинстве 2D платформеров игроку нужно достичь одной цели на уровне. Однако во многих 3D платформерах в каждом уровне нужно собирать множество пазлов, разгадывая головоломки, собирать звезды, части ключей и другие предметы. Это сделано для наиболее эффективного использования трёхмерных областей и вознаграждения игрока за тщательное исследование уровней. Однако некоторые игроки не оценили это, считая это очень скучным занятием, особенно, когда разработчики придумывали очень сложные головоломки или места, где спрятаны части ключей или других предметов. Однако не все трёхмерные платформеры были такими. Например, игра Crash Bandicoot оставалось верной традиции двумерных платформеров, в ней использовались достаточно плоские уровни, в конце которых располагалась игровая цель. Внешний вид данной игры представлен на рисунке 1.7.



Рисунок 1.7 – Внешний вид игры Crash Bandicoot

Также существуют изометрические платформеры, которые являются поджанром 2D и 3D платформеров. Такие платформеры отображают трехмерную сцену при помощи двумерной графики. Игровой мир отображает с жёстко ориентированной камерой без учёта перспективы. Ранними примерами изометрических платформеров являются Congo Bongo (1983 год) и 3D Ant Attack (1983 год). Внешний вид данных игр представлен на рисунках 1.8-1.9.



Рисунок 1.8 – Внешний вид игры Congo Bongo



Рисунок 1.9 – Внешний вид игры 3D Ant Attack

Так же ожидается появление первого четырехмерного платформера Miegakure. Способ отображения четырёх измерений – 3D-сечения. Игровой процесс в принципе схож с классическим 3D платформером, однако по нажатию кнопки одно из измерений меняется местами с её четвертым измерением, осуществляя перемещение в четвёртом измерении. Внешний вид будущей игры представлен на рисунке 1.10.



Рисунок 1.10 – Внешний вид игры Miegakure

## 1.6 Технические требования к курсовому проекту

В ходе аналитического обзора технических средств решения данной задачи и рассмотрения существующих аналогов игр жанра «Платформер» будет создано игровое приложение на языке программирования C# WPF с использованием графической библиотеки OpenGL, реализующее традиционные концепции данного жанра. Приложение будет разработано для одного игрока, и он будет считаться победителем, достигнув конца уровня. Для разнообразия и сложности прохождения уровня будут также реализованы противники с различными типами атаки при помощи паттерна объектно-ориентированного программирования «Декоратор». При попадании врагов в главного героя игры, у него будут сниматься очки здоровья. Если они достигнут нуля, то игрок будет считаться проигравшим. Игрок классически будет прыгать по платформам, которые в свою очередь будут реализованы при помощи паттерна «Фабричный метод». Также у игрока будут сниматься очки здоровья при падении с платформ, находящихся на большой высоте, и при попадании на объект «шипы». Внешний вид игры будет разработан при помощи OpenTK – расширенной низкоуровневой библиотеки, упрощающей работу с OpenGL, использующейся для создания игр, научных приложений или других проектов, требующих 2D и 3D графики, аудио или вычислительных функций.

## АРХИТЕКТУРА ИГРОВОГО ПРИЛОЖЕНИЯ «ПЛАТФОРМЕР»

## 2.1 Архитектура игрового приложения

При проектировании архитектуры игрового приложения был использован метод декомпозиции. Метод декомпозиции – метод разделения целого на части для наиболее наглядного видения того, что требуется сделать для решения общей задачи. То есть для проектирования архитектуры данного приложения главная задача была разделена на подзадачи, с помощью которых в итоге была решена главная задача проектирования. Разумеется, каждый элемент декомпозиции является уникальным для возможности сформировать и отличить составляющие объекта один от другого. Элементы, которые были получены в результате декомпозиции, связаны с функциями отдельных подзадач единой общей задачей.

Обобщенная функциональная схема игрового приложения представлена на рисунке 2.1.



Рисунок 2.1 – Обобщенная схема приложения

Для реализации игровой логики необходимо создать собственный игровой движок. Игровой движок – базовая программная реализация функционала игрового приложения. Основная задача, которую должен выполнять игровой движок с графической стороны, это умение работать со спрайтовой графикой, то есть он должен уметь отображать спрайты и двухмерные анимации. Со стороны функционала приложения основной задачей игрового движка является умение просчитывать физику движения игровых объектов. Для этих целей будет создана библиотека классов GameEngine.

## 2.2 Структура игрового приложения «Платформер»

Игровое приложение должно предоставлять следующий функционал:

* удобное и понятное меню игры;
* передвижение игрока по игровому полю;
* реализация 3-х типов врагов, снимающих очки здоровья у игрока;
* реализация объекта «шипы», снимающих жизнь;
* снятие очков здоровья при падении с высоких платформ;
* переход на следующий уровень;
* возможность пройти игру заново при потере всех жизней;
* завершение игры при победе игрока.

Для реализации игровой логики с использованием игрового движка будет создана новая библиотека классов GameLogic.

Потом будет создан основной проект приложения Windows Presentation Foundation. Он будет обеспечивать реализацию удобного меню приложения.

Так же при написании игрового приложения будут использованы паттерны объектно-ориентированного программирования: паттерн «Декоратор» для реализации различных типов врагов и паттерн «Фабричный метод» для реализации платформ. Данные паттерны будут так же реализованы в библиотеке классов GameLogic.

Паттерн «Декоратор» – это структурный паттерн, который предоставляет возможность динамически добавлять объектам дополнительную функциональность, помещая их в объекты-обертки. Оборачивать объекты можно сколько угодно раз, благодаря общему интерфейсу, как для объектов, так и для оберток. Помимо расширения функциональности, декоратор используется и для реализации обязанностей, которые могут быть сняты с объекта. Однако снятие декораций должно происходить в порядке, обратном наложению [книжечка].

Схема реализации паттерна «Декоратор» представлена на рисунке 2.2.

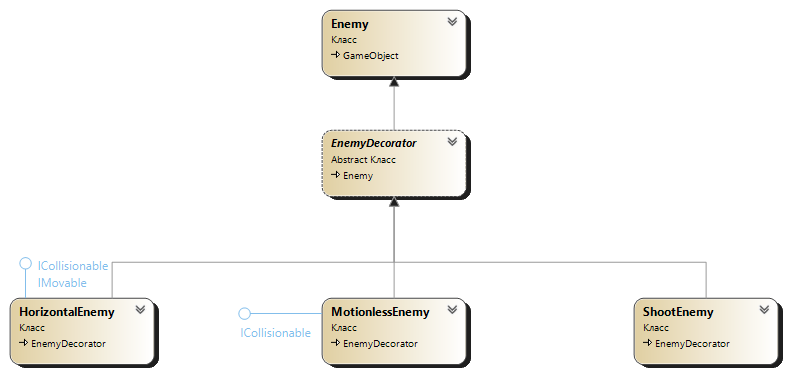


Рисунок 2.2 – Схема паттерна «Декоратор»

Класс Enemy содержит в себе все основные характеристики для врагов, а также виртуальный метод Update, который переопределяется в классах-наследниках от декоратора.

Паттерн «Фабричный метод» – это порождающий паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам [метанит].

Паттерн применяется в случаях:

* когда классу заранее неизвестно, объекты каких классов ему нужно создавать;
* когда класс спроектирован так, чтобы объекты, которые он создает, специфици­ровались подклассами;
* когда класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс при­нимает эти обязанности на себя.

Схема реализации паттерна «Фабричный метод» представлена на рисунке 2.3.

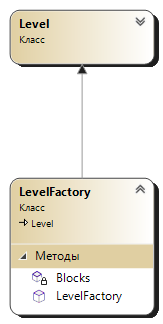


Рисунок 2.3 – Схема паттерна «Фабричный метод»

В данном проекте реализация паттерна «Фабричный метод» имеет неклассическую реализацию, в следствие нестандартной генерации уровней. Уровни проектировались в специальной для этой цели программе, бесплатном графическом редакторе тайловых карт Tiled Map Editor, а она в свою очередь генерировала xml-файл, в котором находились данные о том, на каком месте в матрице блоков находится тот или иной блок, обозначенный своей цифрой. Фабричный метод Blocks считывает и определяет какой тип блока находится под данной цифрой. Так как в xml-файле находятся данные не только о блоках, то класс LevelFactory отвечает полностью за обработку этого файла. Проектирование одного из уровней в данной программе представлено на рисунке 2.4.

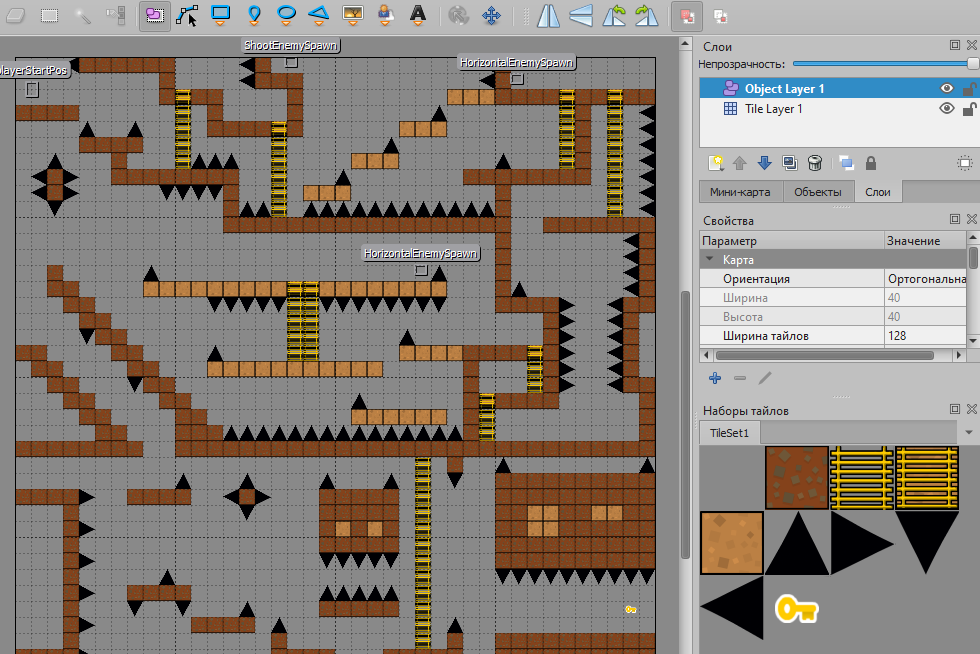


Рисунок 2.4 – Окно редактора Tiled Map Editor

## 2.3 Обзор взаимодействия классов игрового приложения

В процессе разработки игрового приложения была сформирована иерархическая структура классов, которые описывают все возможности своих сущностей.

Для лучшей читаемости кода и удобства были разработаны две библиотеки классов GameEngine и GameLogic. Библиотека классов GameEngine содержит основные классы для функционирования игрового движка, а библиотека классов GameLogic содержит реализацию игровых объектов, паттернов и другие классы игровой логики.

Далее рассматривается библиотека классов GameEngine и взаимодействие её классов.

Классы библиотеки GameEngine:

1. Класс GameObject – базовый абстрактный класс для всех игровых объектов, как движущихся, так и статических. В нем находятся поля position и size, которые определяют положение и размер объектов на игровой сцене;
2. Класс GameComponent – базовый абстрактный класс для всех компонентов игрового движка;
3. Класс Input – класс для работы с событиями клавиатуры;
4. Класс ContentPipe – класс для создания текстуры на основе добавленного изображения и вывода класса Texture2D со всеми данными о текстуре;
5. Класс Spritebatch – класс для работы с отрисовкой текстур. Содержит статический метод Draw для рендеринга спрайтов;
6. Класс Texture2D – класс, содержащий все свойства текстуры, а именно номер, высоту и ширину текстуры;
7. Класс View – класс для работы с камерой, в котором содержится метод перемещения камеры по игровому миру и другие необходимые математические методы.

Схема игрового движка представлена на рисунке 2.5.

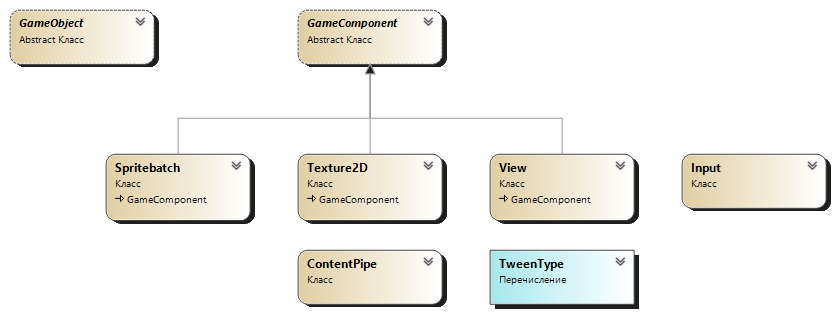


Рисунок 2.5 – Схема игрового движка

Следующей рассматривается библиотека классов GameLogic и взаимодействие её классов.

В данной библиотеке содержатся паттерны объектно-ориентированного программирования «Декоратор» и «Фабричный метод», а также логика игровых объектов.

Паттерн «Декоратор» используется для генерации различных типов врагов для дальнобойной и ближней атаки. В данном случае реализованы 3 типа врагов: HorizontalEnemy – враг с ближней атакой, двигающийся горизонтально, ShootEnemy – враг, атакующий игрока пулями с дальнего расстояния и MotionlessEnemy – не двигающийся враг, размеры которого меняются со временем – увеличиваются и уменьшаются.

Паттерн «Фабричный метод» используется для чтения различных типов блоков, а именно Solid – твердый непроходимый блок, Empty – пустота, Platform – блок-платформа, Ladder – блок-лестница, LadderPlatform – блок-лестница с платформой, SpikeUp, SpikeDown, SpikeLeft, SpikeRight – блоки «шипы», направленные в различные стороны, Key – блок-ключ, являющийся переходом на новый уровень.

Классы библиотеки GameLogic:

1. Класс Player – класс для главного персонажа. В нем содержатся методы для управления персонажем, прописана физика столкновения игрока с другими объектами на игровом поле, отрисовка различных спрайтов при движении и при нахождении персонажа на лестнице, то есть своего рода анимация;
2. Класс Enemy – общий класс для реализации врагов. В нем содержатся общие для всех врагов характеристики;
3. Класс EnemyDecorator – сам декоратор, реализуется посредством абстрактного класса и имеет тот же базовый класс, что и декорируемые объекты. Наследники данного класса представляют дополнительный функционал, которым должен быть расширен объект Enemy;
4. Класс HorizontalEnemy – класс, представляющий дополнительный функционал для Enemy в качестве горизонтального движения от блока к блоку;
5. Класс ShootEnemy – класс для врага, стреляющего в игрока, дополнительный функционал отсутствует помимо загрузки нового спрайта;
6. Класс MotionlessEnemy – класс, представляющий дополнительный функционал для Enemy в качестве увеличения размеров в зависимости от времени;
7. Класс Block – класс для работы с блоками, где содержаться свойства и метод для определения типа блока;
8. Класс Bullet – класс для пули, которой стреляет один из врагов. В нем содержатся методы для реализации движения пули, физики столкновения пули с игроком и другими игровыми объектами;
9. Класс Level – общий класс для генерации всех уровней;
10. Класс LevelFactory – сама фабрика для различных блоков.

Схема взаимодействия объектов игровой логики представлена на рисунке 2.6.

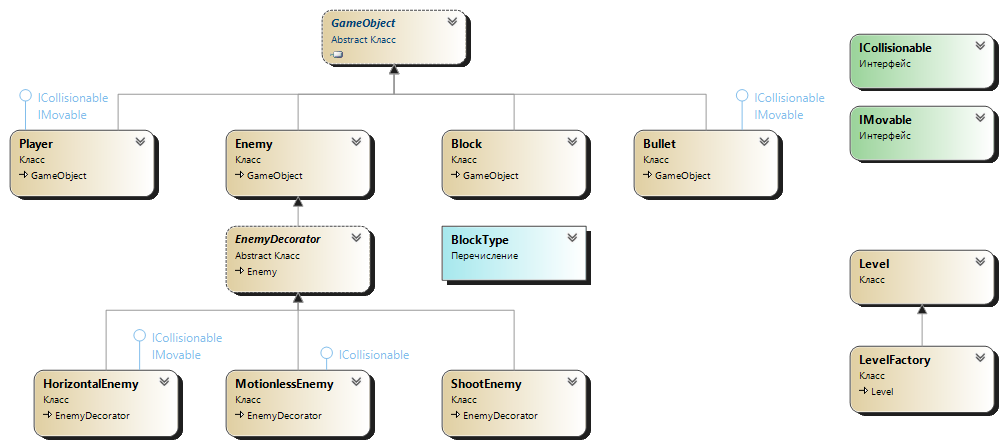


Рисунок 2.6 – Схема взаимодействия объектов игровой логики