## СОДЕРЖАНИЕ

## Введение 4

## 1 Технический проект 7

## 1.1 Классификация игровых приложений 7

## 1.2 Постановка задачи 10

## 1.3 Выбор средств разработки игр 12

## 1.4 Требования к программному обеспечению и техническим средствам 18

## 1.5 Проектирование и моделирование 18

## 1.5.1 Сценарий и стратегия игры 18

## 1.5.2 Функции персонажей 19

## 1.5.3 Основные функции игрового приложения 20

## 1.5.4 Графика 21

## 2 Рабочий проект 23

## 2.1 Стандартные объекты игрового приложения 23

## 2.2 Пользовательские объекты игрового приложения 25

## 2.3 Установка и настройка программы 29

## 2.4 Работа с игровым приложением 30

## 3.1 Исходные данные для расчета затрат 36

## 3.2 Расчет затрат на проектирование системы 38

## 3.3 Расчет затрат на внедрение 40

## 3.4 Расчет прибыли разработчика системы 41

## 3.5 Расчет затрат, связанных с покупкой и использованием системы 41

## Заключение 43

## Список использованных источников 45

## Приложения

## А – Код основных классов игрового приложения 46

## Б – Макеты графических объектов 52

ВВЕДЕНИЕ

Темой дипломного проекта является «Разработка и создание 2D игры «Space Shooter».

Игры – это неотъемлемая часть нашей жизни, с их помощью мы узнаем что-то новое, развиваемся, учимся и общаемся. Игра – это вид деятельности, воплощающий в себе отношение к окружающей, прежде всего социальной, действительности, имеющий специфическое содержание и строение.

Игровая деятельность выполняет следующие функции:

* развлекательную;
* коммуникативную – освоение диалектики общения;
* самореализации в игре как полигоне человеческой практики;
* диагностическую – выявление отклонений от нормативного поведения, самопознание в процессе игры;
* функцию коррекции – внесение позитивных изменений в структуру личностных показателей;
* межнациональной коммуникации – усвоение единых для всех людей социально-культурных ценностей;
* социализации – включение в систему общественных отношений, усвоение норм человеческого общежития.

Игровые жанры определяются не по сюжету, а по игровым действиям, которые наиболее часто совершает игрок. Основных игровых жанров не так много.

Наиболее популярными сейчас являются квесты, RPG-игры, различные стратегии и шутеры. Также большим спросом пользуются 2D и 3D платформеры для различных операционных систем (ОС).

Исходя из выше изложенного, а также потому что видеоигры являются самым популярным развлечением во всём мире, написание данного дипломного проекта целесообразно.

Целью данного дипломного проекта является разработка 2D игрового приложения.

Объектом исследования будут компьютерные 2D игры.

Предметом данного дипломного проекта будут особенности реализации игрового приложения для ОС Windows, а также средства реализации программного продукта.

Для достижения поставленной цели будут рассмотрены и решены следующие вопросы и задачи:

* выбран жанр игры;
* спроектированы структура и функциональность игры;
* спроектированы уровни и персонажи;
* изучены и выбраны средства разработки компьютерных игр;
* описаны минимальные требования к техническим и программным средствам;
* разработана графическая составляющая игры (дизайн уровней и персонажей, пользовательский интерфейс);
* реализована структура и функциональность игрового приложения;
* создана обучающая зона;
* создан инсталлятор;

Прежде чем начать создавать новую игру, необходимо определиться с игровым движком, то есть с программным компонентом, позволяющим создавать и запускать видеоигры. Он предоставляет разработчикам инструменты для создания большинства компонентов игры, а потом позволяет собрать их воедино. Движок игры затрагивает все компоненты игры, начиная от рендеринга, физики, звукового оформления, скриптинга, создания искусственного интеллекта и заканчивая сетевыми аспектами. Если что-то создать с его помощью невозможно, то это создается в специализированной программе и потом импортируется в игру.

С выходом бесплатных версий Unreal Engine и Source 2 появилось ещё больше возможностей делать собственные игры. Но выбор движка, подходящего под потребности и навыки – дело не самое простое. Чтобы серьёзно углубиться в большинство этих программных средств, потребуются как минимум базовые навыки программирования. Но в некоторых случаях можно будет обойтись даже без них и сразу взяться за дело.

Теоретическими основами и методами решения поставленных задач будут методологии проектирования функциональности приложения, особенности реализации программного кода с помощью конкретного языка программирования, принципы и алгоритмы применения инструментальных средств на различных этапах разработки программного продукта.

Выполнение работы требует творческого подхода и применения дополнительных знаний из области графики.

Ресурсы игры будут создаваться вручную соответствующими средствами и с использованием уже существующих графических объектов.

Данный программный продукт будет предназначен для широкого круга пользователей и предполагает коммерческое использование.

1 ТЕХНИЧЕСКИЙ ПРОЕКТ

1.1 Классификация игровых приложений

Жанры игр формировались бесструктурно и интуитивно на протяжении длительного времени. Разработчики игр проводили смелые эксперименты, создавая новые игровые механики. Неудавшиеся эксперименты забывались, а удачные игры становились примером для других разработчиков. Они копировали популярную игровую механику, добавляя немного идей от себя, таким образом, вокруг наиболее популярных игр образовывались целые классы схожих между собой игр, эти классы и стали называть игровыми жанрами. Игровые жанры определяются не по сюжету, а по игровым действиям, которые наиболее часто совершает игрок.

Разделение на жанры очень полезно на практике. Игры определенного жанра интересны уже сформировавшейся игровой аудитории. Разработчик объявляет, что выпускает игру в таком-то жанре, и игроки уже примерно представляют себе, что будет происходить в игре, даже без уточняющих комментариев разработчиков.

Основных игровых жанров не так много.

Квест – жанр, где герой перемещается по вымышленному миру, подбирает или достает иными способами всевозможные предметы, позволяющие достичь основной игровой цели – как правило, кого-либо спасти, куда-нибудь добраться. Обычно делается вид на главного героя сбоку, фон и персонажи рисованные. Этот жанр имеет наибольшее количество своих представителей из-за относительной технической легкости создания – никаких особенно трудоемких задач при разработке среднестатистического квеста решать не приходится.

Подгруппой жанра «Квест» можно считать так называемые «Приключения», где больше активных действий игрока (по сравнению с квестами), эпизодически требуется моментально реагировать на события.

RPG (Role Playing Games – ролевые игры) – имеют давнюю многолетнюю историю. Появились они в виде настольных игр, а также были распространены в качестве военно-спортивных игр на пересеченной местности, чаще всего лесной, с использованием деревянных мечей, луков и так далее. С появлением компьютеров они просто были перенесены на соответствующую платформу с некоторыми доработками. Чем-то RPG напоминают квесты – в них часто приходится выполнять различные задания. Этот жанр ориентирован на людей, имеющих склонность к древнему оружию, магии, вообще атрибутике средних веков, литературе жанра «фэнтези». Считается, что игрок управляет одновременно группой персонажей, которая может состоять не только из людей, а из эльфов, гномов, халфлингов, оборотней. В наборе команды, снаряжении персонажей различными доспехами и оружием, а также в наращивании их количественных характеристик и состоит главная притягательность игр жанра RPG.

В RPG требуется не такой сложный движок, главное – проработанность сюжета, артефактов, магии, персонажей. Существует несколько приличных готовых сюжетов, до сих пор не реализованных. Безусловно, жанр заслуживает пристального внимания для российского разработчика.

Стратегии – наиболее распространен и характеризуется тем, что игрок управляет большим количеством войск (юнитов). В качестве юнитов могут выступать танки, пехота, космические корабли, отряды конницы. Стратегические игры подразделяются на пошаговые, когда сначала игрок делает ход всеми своими юнитами, потом передаёт ход компьютеру, и на так называемые «Real-time» (в реальном времени), когда действие происходит непрерывно для всех играющих. «Real-time» – более жизненный и более интересный в многопользовательском режиме, зато пошаговые стратегии более спокойные и имеют свой круг поклонников, и в них можно организовать игру нескольких человек всего на одном компьютере.

Симуляторы – бывают самыми разными в плане имитируемого объекта: автомобиля, самолета, вертолета и т.д. Иногда имитация может оказаться достоверно реализована, однако для самолетных и вертолетных симуляторов характерно очень сложное управление, в котором может быть задействована практически вся клавиатура. Из-за этого больше половины среднестатистических пользователей просто не могут элементарно освоить игру, что, естественно, отрицательно сказывается на ее популярности. Кроме того, симуляторы требуют опять-таки разработки мощного движка.

Шутер – в таких играх не очень важен сюжет, а нехватка патронов и вовсе отсутствует. Главная цель – уничтожение большого количества врагов. Существует огромное количество видов шутеров, наиболее распространены FPS, TPS, TDS, Scroll Shooter.

FPS (First Person Shooter) – боевик «от первого лица». В этом случае игрок наблюдает за происходящим «из глаз» и может покидать тело персонажа только в момент внутриигровых заставок. Особенность жанра заключается в погружения в игровую атмосферу за счёт дополнительных эффектов (тряска экрана, графические эффекты «моргания» и «ослепления» и т.п.). Игры данного жанра чаще всего разрабатываются новичками, из-за относительно лёгкого исполнения. При разработке более сложных игр затрачивается больше времени и сил, но и игра в итоге становится интереснее.

TPS (Third Person Shooter) – боевик от третьего лица. В отличие от FPS, персонажа видно полностью или частично, со спины. Это даёт больше возможностей для дополнительных особенностей (QuickTime Event, акробатические трюки и т.п.) Часто в играх этого жанра можно переключатся из режима TPS в FPS и обратно.

Scroll Shooter (Shoot'em Up) – игрок, управляющий персонажем с оружием или техническим средством, должен уничтожить большое количество врагов с помощью стрельбы. Представители данного жанра бывают платформенными (с соответствующими признаками «Run-and-Gun») и летательными (авиа или космическими). Вторые часто отличаются автоматическим продвижением вперёд, а иногда и бесконечностью (случайно появляющиеся враги и препятствия, зависящие от продолжительности времени игры, от этого и цель меняется на более аркадную, т.е. набор очков).

TDS (Top Down Shooter) – шутер с видом сверху. Игры данного жанра чаще всего разрабатываются новичками из-за относительно лёгкого исполнения. Другими словами это «Shoot'em Up» с видом сверху [4].

Платформер – жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Противники всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Иногда противник может быть нейтрализован либо прыжком ему на голову, либо из оружия, если им обладает герой. Смерть живых существ обычно изображается упрощённо или символически (существо исчезает или проваливается вниз за пределы экрана). Игры подобного жанра характеризуются нереалистичностью, рисованной мультяшной графикой. Героями таких игр обычно бывают мифические существа (к примеру, драконы, гоблины) или антропоморфные животные [2].

Практика показывает, что очень часто можно добиться совсем неплохих успехов, удачно смешивая различные жанры. Но такие варианты жанров довольно редки, и их крайне сложно выполнить удачно.

В рамках дипломного проекта будет разрабатываться игра в жанре «Платформер» для одиночного игрока, в 2D-стиле.

1.2 Постановка задачи

Необходимо разработать игру в жанре «Шутер» для одиночного игрока, в 2D-стиле, для ОС Windows.

Приложение должно иметь следующую структуру:

* главное меню;
* игровой уровень;
* ознакомление с рекордами

Приложение должно иметь дружественный интерфейс и обеспечивать:

* интерфейс главного меню (новая игра, продолжить, выход);
* сохранение и отображение статистики;
* подсчёт секунд, минут, часов за прохождение

Нужно разработать правила игры, базовые концепции и механики:

* перемещение персонажа в пространстве;
* систему урона игровым объектам;
* систему контроля персонажа (счётчики здоровья);
* систему контроля игры (общий таймер игры, создание противников и ведение игры);
* систему искусственного интеллекта (перемещение противников и нейтральных персонажей);

Также надо разработать игровую графику:

* элементы пользовательского интерфейса (иконки, кнопки, шрифты, шкалы прогресса);
* игровых и нейтральных персонажей, а также противников;
* элементы окружения.

Также нужно разработать инсталляцию приложения.

Входными и выходными данным для приложения будет количество набранных секунд, минут, часов.

Способ решения поставленных задач зависит от выбора среды разработки и языка программирования.

Тестирование будет проводиться на персональном компьютере с   
64-разрядной ОС Windows 7, процессором Intel Core i3, CPU 3.70 GHz, 4 Гб RAM, дисплеем SVGA с разрешением не менее 1024х768 точек на дюйм, оснащенном клавиатурой и мышью.

1.3 Выбор средств разработки игр

Существует множество игровых движков, позволяющих быстро и эффективно создать игровое приложение. Одни из них перечисляются ниже.

CryENGINE – это чрезвычайно мощный игровой движок, созданный компанией-разработчиком Crytek, впервые представленный в игре Far Cry. Он предназначен для разработки под персональный компьютер (ПК) и консоли, включая PlayStation 4 и Xbox One. Его графические возможности превосходят Unity и Unreal Development Kit (UDK), и время от времени стоят на шаг впереди Unreal Engine 4 (UE4): новейшее освещение, реалистичная физика, продвинутые системы анимации и многое другое. Последней игрой на CryENGINE была Ryse: Son of Rome. По аналогии с UDK и UE4 в CryENGINE встроены мощные и интуитивные функции, касающиеся работы с дизайном уровней.

Продуктивное использование CryENGINE потребует определённого времени на его изучение, и могут возникнуть затруднения при отсутствии опыта работы с другими движками. Если не нужна графика уровня Crysis 3 или Ryse: Son of Rome, стоит присмотреться к чему-то более дружелюбному к пользователю.

Ценовая модель CryENGINE несколько отличается от конкурентов. За использование движка нужно платить $9,90 в месяц. Он не полностью бесплатный, как UE4 или Unity 5, зато не требует выплаты роялти, так что $9,90 – это всё, что придётся платить Crytek. В зависимости от размера студии и команды, отсутствие роялти может быть огромным преимуществом.

Если нет никакого опыта в программировании, лучше начать с простейших инструментов. Из таковых самые популярные и общепризнанные – это Stencyl и GameMaker. Оба просты в освоении новичками, и на их основе уже создано несколько качественных игр.

Stencyl позволяет делать игры без программирования. Интерфейс полностью основан на перетаскивании, игры можно выпускать под Windows, Mac, Linux, iOS, Android и Flash. Здесь используется LEGO-подобный подход к построению кода посредством размещения блоков. Stencyl задуман для простого создания 2D-игр, поэтому чаще служит основой паззлам и сайд-скроллерам. Что-нибудь сложное сделать будет проблематично, так что, для разработки чего-то более сложного, вроде RPG, лучше выбрать другой движок. С помощью Stencyl на свет появилось несколько популярных игр, в том числе Impossible Pixel и Zuki’s Quest. Также в нём есть встроенное обучение, которое расскажет обо всём, что нужно знать.

GameMaker – бесплатная утилита для новичков, с помощью которой можно делать игры под Windows, Mac, iOS и Android. Как и в Stencyl, тут почти всё работает на перетаскивании, но есть ещё и возможность писать свои скрипты и подключать сторонние библиотеки. Бесплатная версия при экспорте накладывает водяные знаки, но, тем не менее, GameMaker отлично подходит для первого знакомства с процессом разработки игр и содержит встроенное обучение основам. Тут нет такового жанрового ограничения, как в Stencyl, и можно найти отдельные руководства для разных типов игр. В GameMaker была разработана оригинальная версия Spelunky и Hotline Miami.

Cocos2D – это программа с открытым исходным кодом для создания 2D-игр. Игры можно выпускать под Windows, Mac, Android, iOS, Windows Phone или на веб-платформе. Большая часть того, что придётся делать в Cocos2D – это работа с C++ (также есть поддержка Lua и JavaScript), так что нужно освоить этот язык программирования, прежде чем вообще браться за Cocos2D. Однако, при знании упомянутых языков, программа становится довольно простой в использовании. В ней есть полноценная интегрированная среда разработки, и она совершенно бесплатна. На Cocos2D создано немало успешных игр различных жанров, включая Badland, имеющую множество наград.

Для создания трёхмерных игр существуют два наиболее популярных инструмента – UE4 и Unity. У обоих есть свои сильные и слабые стороны и разные моменты в лицензионных соглашениях, с которыми стоит ознакомиться перед принятием окончательного решения.

Unity позволяет делать трёхмерные и двумерные игры практически для любой платформы, включая Windows, Mac, Xbox, Playstation, Android, iOS. Он поддерживает игровые ресурсы, созданные в 3ds Max, Maya, Softimage, Cinema 4D, Blender. Unity использует C#, наряду с собственным языком программирования. Если сравнивать Unity и Unreal, первый проще в освоении. У него есть богатый набор готовых решений игровых механик и встроенная библиотека игровых ресурсов, в которой довольно просто за ними следить. Unity – лучший движок для первых проектов, потому что его проще понять и изучить, чем Unreal. Ещё Unity поддерживает альтернативные модели оплаты прямо в движке, включая несколько free-to-play моделей монетизации.

Ключевые плюсы использования Unity вообще и C# в частности:

* использование .NET Framework;
* возможность сделать сборку сразу на iPhone и для веб-плеера;
* уже готовые сценарии поведения на C# можно будет повторно использовать, например, при создании порта игры на XBox при помощи XNA (с изменениями, но все же);
* удобный интерфейс и не слишком требовательные к ресурсам ПК редакторы;
* приемлемая цена.

Также в Unity можно использовать ряд встроенных и подключаемых инструментов:

* Post-Processing Stack – этот инструмент является источником мощного функционала для дизайнеров, которые хотят улучшить визуальный ряд своей игры. Post-Processing Stack служит для добавления пост-эффектов. В прошлых версиях Unity уже была реализована такая возможность: пост-эффекты можно было добавлять на камеры в виде отдельных компонентов, но Post-Processing Stack – это единый механизм, в котором собраны и доступны для вызова все эффекты, такие как сглаживание, смазывание, свечение и тому подобное. Удобство этого инструмента в том, что больше не нужно обращаться к каждому из них отдельно. Вместо этого представлен стек эффектов, которые можно менять как угодно;
* Lighting Box – это платный инструмент из магазина инструментов Unity, созданный для настройки графики в игре. Разрабатывался с целью упрощённой настройки цвета и света в игре. Инструмент так же предполагает предустановленные настройки цветовых схем популярных сред разработки игр, таких как Unreal Engine и Cry Engine.
* Cinemachine – набор умных камер, с помощью которых можно делать кат-сцены с использованием камер, ролики на движке, а также связывать все это с возможностями Timeline;
* Unity Anima2D – широкий инструмент для работы со скелетной 2d анимацией, включая инверсную кинематику и автоматическую генерацию весов. Раньше он был платным, но на данный момент Unity сделала его бесплатным [9].

Для написания скриптов используется редактор UniSciTE на базе Scintilla, но также есть возможность использовать Visual Studio.

По сравнению с Unity, в Unreal Engine процесс создания 2D игр более трудоёмкий и требует большее количество ресурсов системы, что не нежелательно при создании небольших проектов. Поэтому для разработки игрового приложения будет использоваться среда разработки Unity 2017 из-за её удобства в освоении.

Программа может быть написана на таких языках программирования, как C++, С# или Java.

Выбор был остановлен на языке программирования C# в среде программирования Microsoft Visual Studio 2017.

Visual Studio обеспечивает поддержку новейших языковых функций на протяжении всего процесса разработки. Данная среда разработки поддерживает C#, Visual Basic, C++, TypeScript, F# и даже сторонние языки, например, JavaScript.

Актуальность языка С# обуславливает следующее:

* интегрировал в себе преимущества языков Java и С++, при этом в объединённом языке исключены некоторые спорные директивы, макросы, отменены глобальные переменные;
* создавался параллельно с каркасом .Net Framework и в полной мере учитывает все его возможности – как FCL, так и CLR;
* благодаря каркасу .Net Framework, ставшему надстройкой над ОС, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода;
* является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами;
* является мощным объектным языком с возможностями наследования и универсализации;
* сохранив основные черты С++, язык стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства С++ как указатели, адресация, разыменование, адресная арифметика;
* мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C# [11].

В качестве редактора изображений для создания векторных изображений будет использоваться Adobe Photoshop CS6. Это многофункциональный графический редактор, разработанный и распространяемый фирмой Adobe Systems. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты.

Для создания инсталляции была выбрана программа Smart InstallMaker. Это условно-бесплатная утилита с закрытым исходным кодом, которая предоставляет пользователям мощный и простой в использовании инструмент для создания высокоэффективных инсталляторов в ОС Microsoft Windows.

Для данного игрового приложения не было выбрано не одного из выше перечисленных движков, что имеет свои кардинальные плюсы, нежели, если бы программа имела игровой движок.

Как правило, движки используют для того, чтобы написать крупное игровое приложение, не затрачивая времени на реализацию уже имеющихся в игровом движке встроенных функций, что позволяет сократить срок написания многоуровневого, сложного приложения, облегчить задачу как новичкам, так и опытным программистам.

Как я упомянул выше – движки используются либо дилетантами, либо опытными программистами, которые желают разработать очень крупное приложение, тратя на это минимальное количество времени, так как в бизнесе время играет важную роль. Но целесообразно ли выбирать движок для приложения, имеющего 5 игровых уровня? Думаю, ответ очевиден.

Таким образом, выбор игрового движка для маленького проекта – дело ресурсозатратное не только на стадии разработки приложения, но и на стадии его эксплуатации, потому что любая подключаемая библиотека или фреймворк имеет определённый вес, который ухудшает производительность, но за то облегчает работу «конструкторным программистам».

1.4 Требования к программному обеспечению и техническим средствам

Для обеспечения корректности работы игрового приложения, необходимо чтобы компьютер был оснащен:

* процессором Intel Core 2 Duo 2, 2 GHz;
* оперативной памятью не менее 2 ГБ;
* дисплеем не менее 1024х768 точек;
* местом на жестком диске 600 МБ;
* манипулятором «мышь»;
* клавиатурой.

Так же на компьютере должна быть установлена операционная система Windows 7/ 8.1/ 10.

Для разработки приложения на компьютере должно быть установлено следующее программное обеспечение:

* DirectX 9.0c;
* Adobe Photoshop CS6;
* Visual Studio 2017;
* Smart InstallMaker.

Для обеспечения корректной работы игрового приложения, необходимо чтобы компьютер был оснащен: ОЗУ 100 Мб и не менее 50 Мб свободной памяти.

1.5 Проектирование и моделирование

1.5.1 Сценарий и стратегия игры

После третьей мировой войны прошло тысячи лет, человеческая цивилизация распространилась не только по всему земному шару, но и по всей галактике. Люди научились строить космические корабли, выращивать растения и лечить рак.

К сожалению, человеческая болезнь, которая имеет симптомы ненависти, злобы и войны привела к очередному столкновению между двумя цивилизациями.

Стратегия игры заключается в том, чтобы манипулируя космическим кораблём продержаться самое длительное время, обороняясь.

Дополнительно можно смотреть статистику, нажимая кнопку «Tab»

1.5.2 Функции персонажей

В приложении будет два типа персонажей:

* герой, управлять которым будет пользователь;
* противники, управление которыми будет осуществляться компьютером.

У героя будут следующие состояния:

* бездействие;
* атака;
* передвижение;

Если герой погибает, то происходит добавление его статистики в базу данных: имя и время которое он смог держать оборону (секунды, минуты, часы).

У героя и противников возможны следующие функции:

* получить урон;
* умереть;
* атаковать;

1.5.3 Основные функции игрового приложения

Игровое приложение будет иметь следующие функции:

* возможность начать новую игру, нажав кнопку Play (Играть) в главном меню, при этом будут обнуляться счётчик времени;
* объекты будут физически взаимодействовать друг с другом;
* производиться подсчёт количества набранного времени;
* здоровье персонажа будет меняться при столкновении с врагами и когда игрок попадает под наносящую урон зону (будут вычитаться очки здоровья);
* здоровье персонажа будет меняться при собирании предметов восстанавливающих очки здоровья.

Для сохранения данных можно использовать четыре вида файлов:

* JSON – формат записи объектов, позволяет хранить данные в удобно читаемом виде при маленьком размере, также поддерживает добавление свойств, что позволяет мигрировать с сохранением старых версий на новые;
* XML – формат записи объектов, при котором достигается хорошая читаемость, версионность, но размер намного выше, чем в JSON, из-за повторяющихся названий тегов;
* PlayerPrefs – это класс из стандартной библиотеки Unity, который позволяет сохранять данные в долговременную память устройства. Он часто используется разработчиками для хранения различных настроек, достижений, прогресса игрока и другой информации о состоянии игры. На ОС Windows эти данные сохраняются в системном реестре;
* бинарный – формат записи, который не читаем людьми, имеет маленький размер, но не поддерживает легкую версионность, так как формат файла четко задан.
* БД – реализационная база данных, представляющая из-себя иерархию таблиц.

Выбор был остановлен на формате БД.

1.5.4 Графика

Разработка и проработка графических объектов игрового приложения разделена на следующие направления:

* создание и прорисовка героя (рисунок Б.1);
* создание и прорисовка противников и их действий (рисунок Б.2);
* проектирование игровой зоны (рисунок Б.3);

2 РАБОЧИЙ ПРОЕКТ

2.1 Стандартные объекты приложений

На основе технического проекта было разработано игровое приложение «Space Shooter» для операционной системы Windows 7.

Для создания игры была использована среда разработки Visual Studio 2017, язык программирования С# а также ресурсы, разработанные в программе Adobe Photoshop CS6.

В программе используются следующие системные классы С#:

* System – содержит фундаментальные и базовые классы, определяющие часто используемые типы значений и ссылочных данных, события и обработчики событий, интерфейсы, атрибуты и исключения обработки;
* System.IO – содержит типы, поддерживающие ввод и вывод, включая возможности чтения и записи данных в потоках как синхронно, так и асинхронно, реализации пользовательского ведения журнала и обработки входящих и исходящих потоков данных в последовательных портах;
* System.Collections – содержит функции для работы с массивами, списками и прочими коллекциями;
* System.Collections.Generic – системный класс, предоставляющий функции для работы с динамическими типами [10].

В программе используется объектно-ориентированный подход, при котором используется множество классов, каждый объект является сочетанием множества скриптов, которые добавляют им функциональности и могут подключаться и отключаться по требованию.

Самые часто используемые методы:

* void moveFighters(); – функция, которая служит для передвижения вражеских кораблей;
* drawPlayerShip(int k); - функция, которая служит для отрисовки игрока
* drawFighters(); - функция, которая служит для отрисовки вражеских кораблей
* drawPlayerShipShields(); - функция, которая служит для отрисовки жизней игрока
* drawPowerUps(); – функция, которая служит для отрисовки предмета для пополнения жизни
* MovePowerUp(); - функция, которая служит для движения предмета, для пополнения жизни
* moveDeathStar(); - функция, которая служит для движения вражеского корабля

2.2 Пользовательские объекты приложения

Сцены в программе содержат объекты игры. Они могут использоваться для создания главного меню, отдельных уровней и для других целей. Можно считать каждый файл сцены отдельным игровым уровнем. В каждой сцене можно разместить объекты окружения, заграждения, декорации, по кусочкам создавая дизайн и саму игру.

В отдельной сцене храниться контроллер игры и вся система графического интерфейса уровня.

В игре есть три основные сцены:

* главное меню;
* графический интерфейс;
* уровень.

Сцена главное меню появляется при загрузке игры.

Стартовый экран состоит из трех пунктов:

* Играть;
* Рекорды;
* О программе;
* Выход;

Сцена «уровень» содержит игровой мир, маркеры начала, врагов и точки сохранения. В проекте на каждый уровень приходится по сцене типа «уровень», поэтому все уровни в дальнейшем будут упоминаться как одна сцена.

Схема взаимодействия сцен приложения представлена на рисунке 1.



Рисунок 1 – Общая схема взаимодействия основных сцен

Сцены являются аналогами окон в Windows и содержат в себе так называемые игровые объекты, к которым могут применяться различные модификаторы поведения и которые могут взаимодействовать друг с другом с помощью привязанных к ним модификаторов.

Для добавления особой функциональности игровым объектам были созданы следующие скрипты:

* Player – класс, описывающий героя (приложение А);
* Ship – класс, описывающий корабль;
* Game – класс, описывающий методы передвижения и прорисовки;
* PowerUP – класс, описывающий предмет для пополнения жизни;
* Missile – класс, описывающий ракеты;
* DataManager – класс, описывающий методы для работы с БД;
* Explotion – класс, описывающий взрыв при столкновении;
* Validation – класс, описывающий методы проверки ввода данных;
* Records – форма с рекордами;
* NameAndDiffChoice – форма ввода имени и выбора сложности;
* Menu – форма выбора;

Контроллер героя реализован классом Player, который содержит всю информацию об игроке и предоставляет управление пользователю, следя за нажатиями кнопок.

Переменные класса:

* string Name – переменная, хранящая имя;
* string Diff – переменная, хранящая сложность;
* int Seconds – переменная, хранящая секунды;
* int Minutes – переменная, хранящая минуты;
* int Hours – переменная, хранящая часы

Игровое приложение будет иметь следующие функции:

* возможность начать новую игру(кнопка Play(Играть) в главном меню);
* данные автоматически сохраняются после завершения игры в БД;
* здоровье персонажа меняется при столкновении с врагами путем вычитания здоровья в момент, когда игрок сталкивается с противником или снарядом;

Листинг 1 – Функция нанесения урона персонажу врагами:

private void checkPlayerMissileColitionWithFighters()

{

bool breakOut = false;

for (int m = 0; m < playermissiles.Count; m++)

{

for (int x = 0; x < xNumberOfFighters; x++)

{

for (int y = 0; y < yNumberOfFighters; y++)

{

if (playermissiles[m].xGraphicLocation + 10 > fighter[x, y].xGraphicLocation && fighter[x,y].xGraphicLocation + 150 > playermissiles[m].xGraphicLocation &&

playermissiles[m].yGraphicLocation + 20 > fighter[x, y].yGraphicLocation && fighter[x,y].yGraphicLocation + 20 > playermissiles[m].yGraphicLocation)

{

explotion.Add(new Explotion(fighter[x,y].xGraphicLocation, fighter[x,y].yGraphicLocation, 30, 30, 1));

g.FillRectangle(black, playermissiles[m].xGraphicLocation, playermissiles[m].yGraphicLocation, 30, 30);

fighter[x, y].shields = fighter[x, y].shields - 10 \* cannonCount;

if (fighter[x, y].shields < 0)

{

fighter[x,y].xGraphicLocation = 2000;

fighter[x,y].shields = 100;

}

playermissiles.Remove(playermissiles[m]);

breakOut = true;

}

if (breakOut == true) break;

}

if (breakOut == true) break;

}

}

}

2.3 Установка и настройка программы

Для установки программы необходимо запустить файл инсталляции   
Setup.exe. В результате в выбранном пользователем каталоге будет создана папка «Space Shooter», в которой будут размещены исполняемый файл программы (Space Shooter.exe). Также будет создан ярлык для запуска приложения на рабочем столе. Ярлык для деинсталляции программы будет размещен в папке программы.

Для нормальной работы программы необходимо:

* 200мб свободного места на жестком диске;
* OC Windows 7, 8.1, 10;
* DirectX 9.0c;
* разрешение экрана не менее 1024x768 точек;
* манипулятор «Мышь»;
* клавиатура.

Для создания инсталляции была использована программа Smart InstallMaker 5.04.

В ходе инсталляции пользователю показываются следующие окна:

* выбор папки установки;
* создание дополнительных ярлыков – позволяет создать ярлык на рабочем столе или отказаться от создания ярлыка;
* ход выполнения процесса установки;
* завершение установки – позволяет выбрать файлы, которые нужно запустить после установки.

После запуска программы инсталляции нужно следовать указаниям в появляющихся окнах.

2.4 Работа с приложением

Для запуска игрового приложения необходимо щёлкнуть на ярлык игры «Space Shooter.exe». После запуска приложения открывается главное меню (рисунок 2).

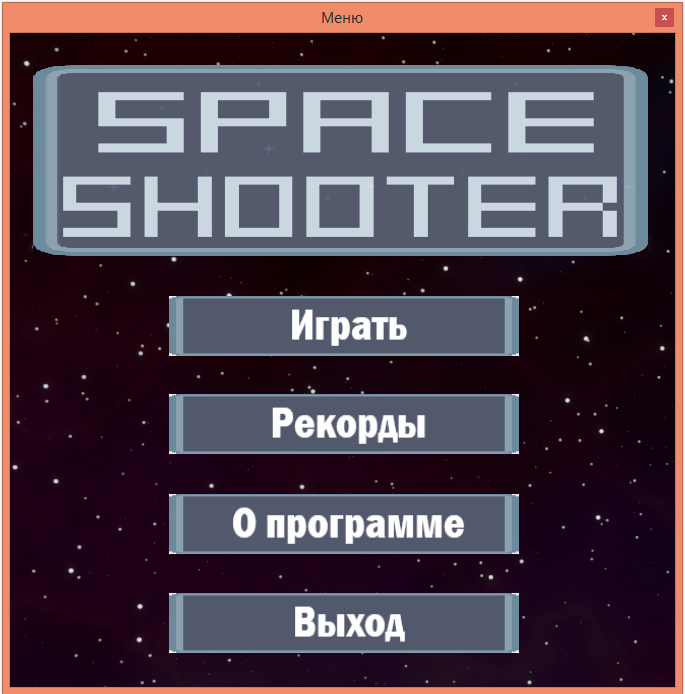


Рисунок 2 – Главное меню приложения

При нажатии в главном меню на кнопку «Играть» (Play), откроется меню выбора сложности и имени (рисунок 3).

Для перемещения персонажа используются клавиши «W», «A», «S», «D», а также клавиши со стрелочками.

Главный персонаж может стрелять по врагам, для этого необходимо нажать клавишу «Space».

После выбора имени и сложности откроется сама игра (рисунок 4)

Для завершения игры необходимо умереть.

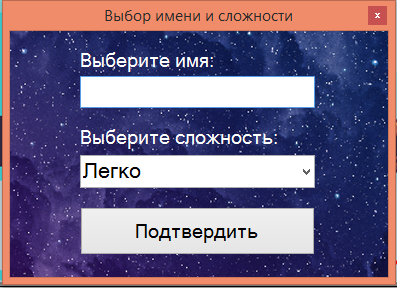


Рисунок 3 – Меню выбора сложности и имени



Рисунок 4 – Окно игры

На уровне нужно остерегаться столкновений с снарядами и вражескими кораблями, которые наносят урон.

Шкала здоровья персонажа отображается в левом верхнем углу экрана в виде толстой полосы.

За собирание предмета в виде лампы, также начисляется ХП (PowerUP), количество которых отображается в левом верхнем углу экрана.

Клавишей «Esc» можно открыть меню паузы (рисунок 5).



Рисунок 5 – Пауза

При этом игра останавливается и предлагается продолжить игру, путём нажатия клавиши «ESC»

ЗАКЛЮЧЕНИЕ

В результате выполнения данного дипломного проекта было разработано игровое приложение «Space Shooter» в жанре «Платформер» для одиночного игрока, в 2D-стиле, для ОС Windows. Также было разработано документальное сопровождение программы, включающее в себя технический и рабочий проекты.

Данная программа включает в себя обучающую и четыре игровых зоны. Она позволяет хорошо провести время и потренировать реакцию. Удобный интерфейс программы позволяет легко ориентироваться в приложении, не требуя от пользователя каких-либо специальных навыков.

Приложение обеспечивает следующие возможности:

* начать игру;
* сохранение статистики игрока;

Основными достоинствами программы является:

* наличие простого и удобного интерфейса;
* данный программный продукт будет предназначен для широкого круга пользователей;
* имеет инсталляцию;
* наличие нормализованной БД.

В ходе выполнения дипломного проекта были достигнуты поставленные цели и решены следующие задачи:

* изучены средства разработки компьютерных игр;
* спроектированы уровни и персонажи;
* выбраны средства реализации;
* описаны минимальные требования к техническим и программным средствам;
* разработаны структура и функциональность программы;
* разработаны графическая составляющая игры (дизайн уровней и персонажей, пользовательский интерфейс);
* создан инсталлятор;

К недостаткам данного игрового приложения можно отнести малое количество уровней для прохождения, небольшое разнообразие врагов, а также отсутствие возможности настройки приложения, в будущем это планируется исправить.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Антонов Л.С., Кадомцев С.Б. Api функции. М.: Просвещение, 2016. 336 с.
2. Мэйн С. Основы Windows Communication Foundation для .NET Framework 3.5. СПб.: БХВ-Петербург, 2015. 306 с.
3. Петров А.В. Учебник юного программиста. М.: Просвещение, 2015. 290 с.
4. Рихтер Д. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. СПб.: Питер, 2017. 407 с.
5. Сибраро П., Клайс К., Коссолино Ф., Грабнер Йохан. WCF4: Windows Communication Foundation и .NET 4. СПб.: БХВ-Петербург, 2015. 478 с.
6. Торн А. Основы анимации в Unity. М.: ДМК, 2016. 176 с.
7. Троелсен Э. Язык программирования С# 5.0 и платформа .NET 4.5 СПб.: БХВ-Петербург, 2017. 430 с.
8. Хокинг Д. Unity в действии. Мультиплатформенная разработка на C#. СПб.: Питер, 2016. 336 с.
9. Библиотека классов .NET Framework. URL: https:// msdn.microsoft.com/ru-ru/library/mt472912(v=vs.110).aspx (дата обращения: 05.04.2019).
10. Официальная документация Unity. URL: https://docs.unity3d.com/ru/ (дата обращения: 05.04.2019).
11. Официальный сайт Smart Install Maker. URL: http:// ru.sminstall.com (дата обращения: 15.05.2019).

ПРИЛОЖЕНИЕ А

Код основных классов игрового приложения

Класс игрока

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SpaceShooterGame

{

class Player

{

public string Name { get; private set; }

public string Diff { get; private set; }

public int Seconds { get; private set; }

public int Minutes { get; private set; }

public int Hours { get; private set; }

public Player(string name, string diff, int seconds, int minutes, int hours)

{

Name = name;

Diff = diff;

Seconds = seconds;

Minutes = minutes;

Hours = hours;

}

}

}

Класс коробля

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SpaceShooterGame

{

class Ship

{

private float cxGridLocation;

private float cyGridLocation;

private float cxGraphicLocation;

private float cyGraphicLocation;

private float cShields;

private float cDamage;

public Ship(float cxGridLocation, float cyGridLocation, float cxGraphicLocation, float cyGraphicLocation, float cShields, float cDamage)

{

this.cxGridLocation = cxGridLocation;

this.cyGridLocation = cyGridLocation;

this.cxGraphicLocation = cxGraphicLocation;

this.cyGraphicLocation = cyGraphicLocation;

this.cShields = cShields;

this.cDamage = cDamage;

}

public float xGridLocation

{

set { cxGridLocation = value; }

get { return cxGridLocation; }

}

public float yGridLocation

{

set { cyGridLocation = value; }

get { return cyGridLocation; }

}

public float xGraphicLocation

{

set { cxGraphicLocation = value; }

get { return cxGraphicLocation; }

}

public float yGraphicLocation

{

set { cyGraphicLocation = value; }

get { return cyGraphicLocation; }

}

public float shields

{

set { cShields = value; }

get { return cShields; }

}

public float damage

{

set { cDamage = value; }

get { return cDamage; }

}

}

}

Класс пополнения жизни

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SpaceShooterGame

{

class PowerUp

{

private float cx;

private float cy;

private float cShields;

private int cCannonCount;

private Bitmap cpicture;

public PowerUp(float cx, float cy, float cShields, int cCannonCount, Bitmap cpicture)

{

this.cx = cx;

this.cy = cy;

this.cShields = cShields;

this.cCannonCount = cCannonCount;

this.cpicture = cpicture;

}

public float x

{

set { cx = value; }

get { return cx; }

}

public float y

{

set { cy = value; }

get { return cy; }

}

public float shields

{

set { cShields = value; }

get { return cShields; }

}

public int cannonCount

{

set { cCannonCount = value; }

get { return cCannonCount; }

}

public Bitmap picture

{

set { cpicture = value; }

get { return cpicture; }

}

}

}

Класс взрыва

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SpaceShooterGame

{

class Explotion

{

private float cxLocation;

private float cyLocation;

private float cxLength;

private float cyHeight;

private int cCount;

public Explotion(float cxLocation, float cyLocation,float cxLength, float cyHeight, int cCount)

{

this.cxLocation = cxLocation;

this.cyLocation = cyLocation;

this.cxLength = cxLength;

this.cyHeight = cyHeight;

this.cCount = cCount;

}

public float xLocation

{

set { cxLocation = value; }

get { return cxLocation; }

}

public float yLocation

{

set { cyLocation = value; }

get { return cyLocation; }

}

public float xLength

{

set { cxLength = value; }

get { return cxLength; }

}

public float yHeight

{

set { cyHeight = value; }

get { return cyHeight; }

}

public int count

{

set { cCount = value; }

get { return cCount; }

}

}

}

Класс проверки ввода данных

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SpaceShooterGame

{

class Validation

{

public bool ValidateName(string name)

{

bool flag = true;

if (name.Length < 3) flag = false;

if (name.Length > 15) flag = false;

if (string.IsNullOrWhiteSpace(name)) flag = false;

return flag;

}

}

}

Класс работы с базой данных

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using SpaceShooterGame.EntityModel;

using System.Windows.Forms;

namespace SpaceShooterGame

{

class DataManager

{

public void Fill(DataGridView data)

{

using (var ctx = new MyDbContext())

{

var all = (from ps in ctx.PlayerScore

join p in ctx.Player on ps.id\_player equals p.id

join s in ctx.Score on ps.id\_score equals s.id

orderby s.seconds + s.minutes + s.hours descending

select new

{

Имя = p.name,

Рекорд = s.hours + " часов, " + s.minutes + " минут, " + s.seconds + " секунд",

Сложность = s.difficulty

}).ToList();

data.DataSource = all;

}

}

private string DifficultyConverter(string combo)

{

string word = "";

switch (combo)

{

case "Все": word = "Все"; break;

case "Только легкие": word = "Легко"; break;

case "Только средние": word = "Средне"; break;

case "Только сложные": word = "Тяжело"; break;

case "Только очень сложные": word = "Очень тяжело"; break;

}

return word;

}

public void Filter(DataGridView data, ComboBox combo)

{

string choice = DifficultyConverter(combo.Text);

using (var ctx = new MyDbContext())

{

if(choice == "Все")

{

Fill(data);

}

else

{

var all = (from ps in ctx.PlayerScore

join p in ctx.Player on ps.id\_player equals p.id

join s in ctx.Score on ps.id\_score equals s.id

where s.difficulty == choice

orderby s.seconds + s.minutes + s.hours descending

select new

{

Имя = p.name,

Рекорд = s.hours + " часов, " + s.minutes + " минут, " + s.seconds + " секунд",

Сложность = s.difficulty

}).ToList();

data.DataSource = all;

}

}

}

private void AddRecordOfExistedPlayer(string playerName, string diff, int sec, int min, int h)

{

using (var ctx = new MyDbContext())

{

var player\_id = ctx.Player.Select(s => s)

.Where(i => i.name == playerName)

.First();

var score = new Score

{

difficulty = diff,

seconds = sec,

minutes = min,

hours = h

};

ctx.Score.Add(score);

ctx.SaveChanges();

var score\_id = ctx.Score

.Select(s => s.id).Max();

var playerScore = new PlayerScore

{

id\_player = player\_id.id,

id\_score = score\_id

};

ctx.PlayerScore.Add(playerScore);

ctx.SaveChanges();

}

}

private void AddRecordOfNotExistingPlayer(string playerName, string diff, int sec, int min, int h)

{

using (var ctx = new MyDbContext())

{

var player = new EntityModel.Player

{

name = playerName

};

ctx.Player.Add(player);

ctx.SaveChanges();

var player\_id = ctx.Player

.Select(s => s.id).Max();

var score = new Score

{

difficulty = diff,

seconds = sec,

minutes = min,

hours = h

};

ctx.Score.Add(score);

ctx.SaveChanges();

var score\_id = ctx.Score

.Select(s => s.id).Max();

var playerScore = new PlayerScore

{

id\_player = player\_id,

id\_score = score\_id

};

ctx.PlayerScore.Add(playerScore);

ctx.SaveChanges();

}

}

public void Add(string playerName, string diff, int sec, int min, int h)

{

using (var ctx = new MyDbContext())

{

var IsPlayerExist = ctx.Player

.Select(s => s)

.Any(n => n.name == playerName);

if (

IsPlayerExist)

AddRecordOfExistedPlayer(playerName, diff, sec, min, h);

else AddRecordOfNotExistingPlayer(playerName, diff, sec, min, h);

}

}

public void Search(string playerName, DataGridView data)

{

using (var ctx = new MyDbContext())

{

var all = (from ps in ctx.PlayerScore

join p in ctx.Player on ps.id\_player equals p.id

join s in ctx.Score on ps.id\_score equals s.id

where p.name == playerName

orderby s.seconds + s.minutes + s.hours descending

select new

{

Имя = p.name,

Рекорд = s.hours + " часов, " + s.minutes + " минут, " + s.seconds + " секунд",

Сложность = s.difficulty

}).ToList();

data.DataSource = all;

}

}

}

}

ПРИЛОЖЕНИЕ Б

Макеты графических объектов

F:\visual studio\Space shooter\bin\Debug\Resources\SpaceShip1.png F:\visual studio\Space shooter\bin\Debug\Resources\SpaceShip2.png

Рисунок Б.1 – Анимация главного героя

F:\visual studio\Space shooter\bin\Debug\Resources\Fighter1.png F:\visual studio\Space shooter\bin\Debug\Resources\Fighter2.png

Рисунок Б.2 – Анимация противника

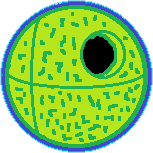


Рисунок Б.3 – Мертвая звезда

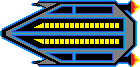


Рисунок Б.4 – Уничтожитель звёзд

F:\visual studio\Space shooter\bin\Debug\Resources\FireBall.png

Рисунок Б.5 – Огненный шар

F:\visual studio\Space shooter\bin\Debug\Resources\fireExplotion1.png F:\visual studio\Space shooter\bin\Debug\Resources\fireExplotion2.png

Рисунок Б.6 – Взрыв

F:\visual studio\Space shooter\bin\Debug\Resources\Missile.png

Рисунок Б.7 – Ракета

F:\visual studio\Space shooter\bin\Debug\Resources\PowerUp.png

Рисунок Б.8 – Предмет пополнения жизни

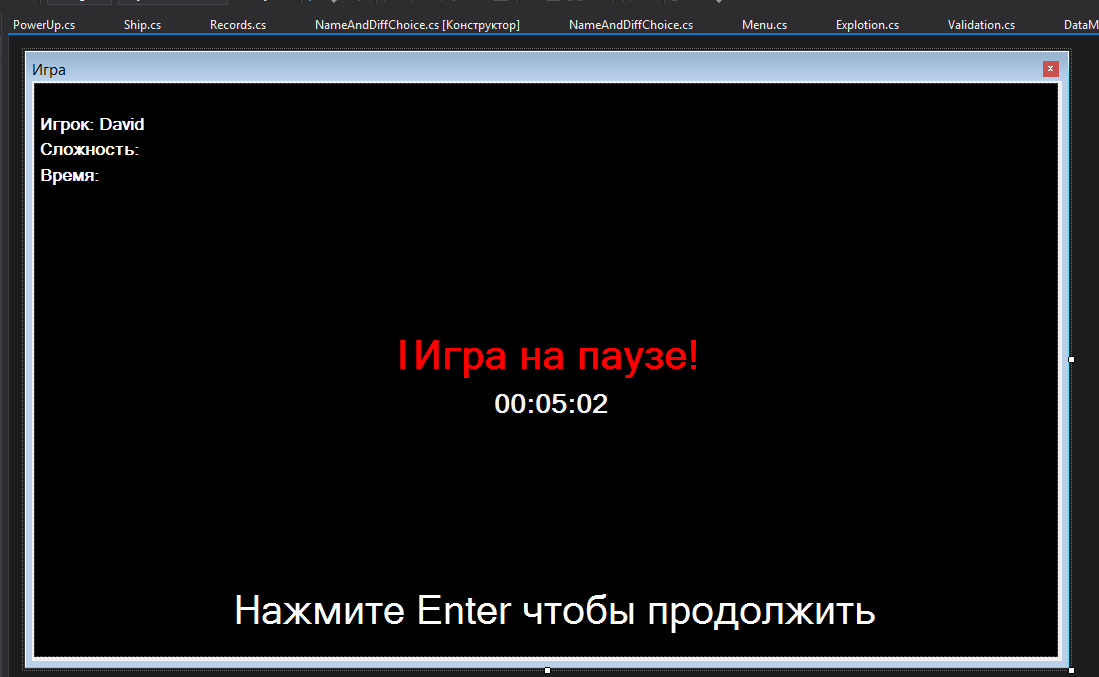


Рисунок Б.9 – Проектирование игровой зоны