Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Игра «Галактика»

Выполнила:

студентка группы 953503

Зорко П.А.

Научный руководитель:

ассистент кафедры информатики

Удовин И.А.

Минск 2020**Содержание**

[**Введение** 3](#_Toc41741445)

[**1 Описание среды разработки** 5](#_Toc41741446)

[1.1 Среда разработки 5](#_Toc41741447)

[1.2 Среда разработки Microsoft Visual Studio 5](#_Toc41741448)

[1.3 Описание языка программирования С# 6](#_Toc41741449)

[1.4 Описание игрового движка 8](#_Toc41741450)

[**2 Разработка программы** 9](#_Toc41741451)

[2.1 Этапы разработки игры 9](#_Toc41741452)

[2.2 Описание игры 9](#_Toc41741453)

[2.3 Структура проекта. Основные методы 10](#_Toc41741454)

[2.4 Работа со спрайтами 13](#_Toc41741455)

[2.5 Работа со шрифтами 13](#_Toc41741456)

[2.6 Описание основных игровых элементов 14](#_Toc41741457)

[2.7 Взаимодействие объектов 18](#_Toc41741458)

[2.8 Описание экранов игры 20](#_Toc41741459)

[**Заключение** 24](#_Toc41741460)

[**Список используемых источников** 25](#_Toc41741461)

Введение

Ни для кого не секрет, что видеоигры прочно заняли свою позицию в современной индустрии развлечений. Разработка игр может оказаться не только увлекательным, но и прибыльным делом. Первые примитивные компьютерные и видеоигры были разработаны в 1950-х и 1960-х годах. Они работали на таких платформах, как осциллографы, университетские мейнфреймы и компьютеры EDSAC. Самой первой компьютерной игрой стал симулятор ракеты, созданный в 1942 году Томасом Голдсмитом Младшим (англ. Thomas T. Goldsmith Jr.) и Истл Рей Менном (англ. Estle Ray Mann). Позже, в 1952 году, появилась программа "OXO", имитирующая игру "крестики-нолики", созданная А.С. Дугласом как часть его докторской диссертации в Кембриджском Университете. Игра работала на большом университетском компьютере, известном как EDSAC (Electronic Delay Storage Automatic Calculator). В настоящее время, разработка игры – это многомиллионный процесс, в котором задействована целая команда разработчиков, сложные современные технологии и даже маркетинговые ходы.

Данная работа посвящена разработке двумерной компьютерной игры «Галактика». Проект относится к жанру shoot 'em up. Жанр shoot 'em up – это разновидность жанра шутер, который, в свою очередь, является разновидностью экшн-игр. Обычно в этих играх используется вид сверху или со стороны, а игроки используют оружие дальнего боя для поражения удалённых целей. Аватар игрока обычно представлен в виде транспортного средства, на которое постоянно нападают враги. Поэтому основной целью игрока является максимально быстрое уничтожение любого движущегося объекта, который может представлять угрозу. В каких-то играх персонаж может выдержать несколько попаданий, тогда как в других единственное попадание приводит к его уничтожению. Основами мастерства игрока в таких играх являются хорошая реакция и способность запоминать последовательности вражеских атак. Особенностью некоторых игр является крайне большое количество снарядов, выпускаемых врагами, последовательности запуска которых игроку требуется запоминать для «выживания» его персонажа. Ещё одна особенность игр в этом жанре – весьма быстротечный игровой процесс. [4]

В играх жанра shoot 'em up обычно представлено большое разнообразие вражеских персонажей. Каждый из этих врагов может придерживаться особой линии поведения, свойственной его типу, либо атаковать в формациях, что игрок должен научиться предугадывать. Основы геймплея этих игр имеют тенденцию к упрощению. В играх shoot 'em up редко встречается реалистичная физика. Так, персонажи могут мгновенно изменять направление движения без какой-либо [инерции](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B5%D1%80%D1%86%D0%B8%D1%8F), а снаряды двигаются по прямой на постоянных скоростях. Как результат игрок может стрелять, не останавливаясь, при этом его оружие будет наносить урон только подходящим целям.

В данной курсовой работе ставятся следующие задачи:

1. В программе должно присутствовать меню, в котором можно просматривать данные о разработчике и игре, рекорды и непосредственно переход к самой игре.
2. В ней должны использоваться многоцветные анимированные спрайты, прокручивающееся игровое поле со звёздами, музыкальная тема и фоновая музыка, графические значки, которые показывают число жизней игрока.
3. В программе игроку за каждого убитого врага должны начисляться очки.
4. Должна присутствовать возможность сохранения лучшего результата в текстовый файл и вывод этого результатов на экран при последующем запуске программы.
5. Описание среды разработки
   1. **Среда разработки**

Интегрированная среда разработки (англ. Integrated development environment – IDE) – комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Среда разработки включает в себя:

* [текстовый редактор](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80),
* транслятор ([компилятор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) и/или [интерпретатор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80)),
* средства автоматизации сборки,
* [отладчик](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA).

Иногда содержит также средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя.

ИСР предназначенные для нескольких языков программирования: [IntelliJ IDEA](https://ru.wikipedia.org/wiki/IntelliJ_IDEA), [NetBeans](https://ru.wikipedia.org/wiki/NetBeans), [Eclipse](https://ru.wikipedia.org/wiki/Eclipse_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)), [Qt Creator](https://ru.wikipedia.org/wiki/Qt_Creator), [Geany](https://ru.wikipedia.org/wiki/Geany), [Embarcadero RAD Studio](https://ru.wikipedia.org/wiki/Embarcadero_RAD_Studio), [Code::Blocks](https://ru.wikipedia.org/wiki/Code::Blocks), [Xcode](https://ru.wikipedia.org/wiki/Xcode), [Microsoft Visual Studio](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio). [7]

Выбор остановим на среде разработки **Microsoft Visual Studio**.

* 1. **Среда разработки Microsoft Visual Studio**

**Microsoft Visual Studio** – линейка продуктов компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), включающих [интегрированную среду разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как [консольные](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [приложения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), так и приложения с [графическим интерфейсом](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), в том числе с поддержкой технологии [Windows Forms](https://ru.wikipedia.org/wiki/Windows_Forms), а также [веб-сайты](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82), [веб-приложения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), [веб-службы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0) как в [родном](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), так и в управляемом кодах для всех платформ, поддерживаемых [Windows](https://ru.wikipedia.org/wiki/Windows), Windows Mobile, [Windows CE](https://ru.wikipedia.org/wiki/Windows_CE), [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework), [Xbox](https://ru.wikipedia.org/wiki/Xbox), [Windows Phone](https://ru.wikipedia.org/wiki/Windows_Phone) [.NET Compact Framework](https://ru.wikipedia.org/wiki/.NET_Compact_Framework) и [Silverlight](https://ru.wikipedia.org/wiki/Silverlight). [3]

**Visual Studio** включает в себя [редактор исходного кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0) с поддержкой технологии [**IntelliSense**](https://ru.wikipedia.org/wiki/IntelliSense) и возможностью простейшего [рефакторинга кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3). Встроенный [отладчик](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и дизайнер [схемы базы данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). **Visual Studio** позволяет создавать и подключать сторонние дополнения ([плагины](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD)) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем [контроля версий исходного кода](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8) (как, например, **Subversion** и **Visual SourceSafe),** добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов [процесса разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (например, клиент **Team Explorer** для работы с **Team Foundation Server**).

Для создания игры было решено использовать объектно-ориентированный язык программирования (ООЯП) **C#**.

* 1. **Описание языка программирования С#**

**C#** – современный объектно-ориентированный и типобезопасный язык программирования. **C#** относится к широко известному семейству языков **C**, и покажется хорошо знакомым любому, кто работал с **C**, **C++**, **Java** или **JavaScript**. [2]

**C#** является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. **C#** подходит для создания и применения программных компонентов.

Вот лишь несколько функций языка **C#**, обеспечивающих надежность и устойчивость приложений:

* Сборка мусора автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами.
* Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и их восстановлению.
* Типобезопасная структура языка делает невозможным чтение из неинициализированных переменных, индексацию массивов за пределами их границ или выполнение непроверенных приведений типов.
* В **C#** существует единая система типов. Все типы **C#**, включая типы-примитивы, такие как **int** и **double**, наследуют от одного корневого типа **object**.
* **C#** поддерживает пользовательские ссылочные типы и типы значений, позволяя как динамически выделять память для объектов, так и хранить упрощенные структуры в стеке.
* Чтобы обеспечить совместимость программ и библиотек **C#** при дальнейшем развитии, при разработке **C#** много внимания было уделено управлению версиями.
* **C#** включает функции, поддерживающие приемы функционального программирования, такие как лямбда-выражения. [8]

Кроме того, **C#** обеспечивает полную поддержку объектно-ориентированного программирования, включая инкапсуляцию, наследование и полиморфизм.

Инкапсуляция означает, что группа связанных свойств, методов и других членов рассматривается как единый элемент или объект.

Наследование описывает возможность создания новых классов на основе существующих классов.

Полиморфизм означает, что можно иметь несколько взаимозаменяемых классов, даже если каждый класс реализует одни и те же свойства или методы разными способами. [1]

* 1. **Описание игрового движка**

Последним важным выбором на этапе планирования стал выбор игрового движка (ИД).

Игровой движок (англ. game engine) – это центральный программный компонент компьютерных видеоигр и других интерактивных приложений с графикой, обрабатываемой в реальном времени. [6]

Он обеспечивает основные технологии, упрощает разработку и часто дает игре возможность запускаться на нескольких платформах. Игровой движок подразумевает целый комплекс прикладных программ, включающий движок рендеринга (визуализатор) для 2D или 3D графики, физический движок, звук, скриптинг, анимацию, искусственный интеллект, сетевой код, streaming, управление памятью, threading и графические сцены. То есть, все части кода, написанные программистами при разработке игры, являются компонентами движка. Выбор пал в пользу игрового движка **MonoGame**.

**MonoGame** – это набор инструментов, который по уровню сложности находится где-то между игровой подсистемой и более серьезным API наподобие DirectX. Он предоставляет простой в использовании конвейер содержимого, а также все функции для создания упрощенных игр, работающих на ряде платформ. И самое приятное: приложения **MonoGame** пишутся на чистом языке **C#**. А это значит, что можно опубликовать их в **Microsoft Store** или в аналогичных ресурсах. **MonoGame** станет отличным выбором для людей, ранее не знакомых с программированием игр.

1. Разработка программы
   1. **Этапы разработки игры**

На первом этапе была разработана концепция игры, дизайн персонажей, выбраны средства для реализации проекта, подготовлен план, по которому будет создаваться игра. Все текстуры и аудиофайлы были взяты из открытых источников в интернете. Картинки обработаны в программе **Adobe Photoshop CC**.

Затем приступим к непосредственной разработке приложения. Результатом этапа кодирования является готовая программа.

На третьем этапе занимаемся отладкой программы, то есть поиском и устранением ошибок. Последние делятся на две группы: алгоритмические и синтаксические (ошибки в тексте исходной программы). Из этих двух групп ошибок наиболее легко устранить синтаксические ошибки, тогда как алгоритмические ошибки определить достаточно трудно.

Следующим важным этапом является тестирование программы, поскольку в большинстве случаев программы создаются не для личного применения. На этапе тестирования проверяем поведение программы при разных игровых ситуациях.

* 1. **Описание игры**

Общий цикл работы программы можно представить следующим образом. После запуска возникает окно приложения с главным меню. Начать игру можно нажатием клавиши **Enter**. На поле генерируются космический корабль игрока и вражеские корабли.

Волна за волной армии противника атакуют корабль игрока, который может двигаться влево-вправо и верх-вниз (при этом он не может выходить за границы экрана). Движение корабля игрока осуществляется с помощью стрелок вверх, вниз, вправо, влево. Игрок может выпускать снаряды с помощью клавиши пробел (или левой кнопки мыши). На фоне прокручивается звёздное небо.

Задачей игрока является уничтожение как можно большего количества кораблей противника. За каждого поверженного противника игроку начисляются очки. Количество очков зависит от типа корабля противника. Так игрок может получить 10, 20 или 30 очков за убитого врага.

В начале игры у каждого игрока имеется 3 игровых жизни. Игрок теряет жизнь при столкновении с кораблем противника или при попадании в него вражеского выстрела. Игра заканчивается, когда игрок теряет все жизни. Лучший счет игрока записывается.

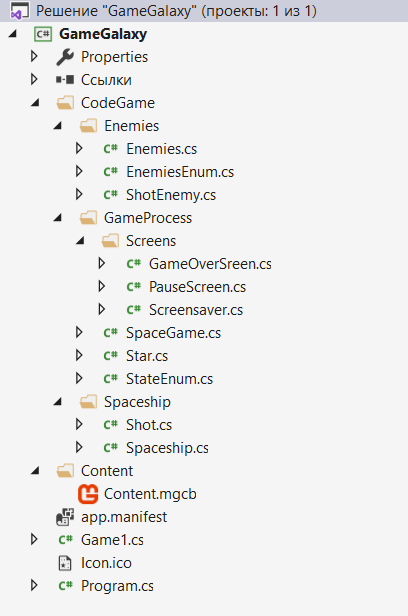
Также в игре предусмотрена функция паузы. При нажатии клавиши **P** вы можете поставить игру на паузы, а затем при нажатии клавиши **Enter** можете продолжить игровой процесс либо закончить игру при нажатии клавиши **Escape** (однако, при этом ваш рекорд не будет сохранен).

* 1. **Структура проекта. Основные методы**

Всего в рабочем проекте курсовой работы присутствует более 10 различных файлов, распределённых по папкам в зависимости от их предназначения. Каждый файл содержит свой отдельный класс или перечисление. Опишем элементы по умолчанию, представленные в окне **Обозреватель решений**:

* **Program.cs.** Этот файл в большинстве проектов не будет изменен. Он служит загрузчиком для игрового класса.
* **Game1.cs.** Это сердце или, скорее, мозг игры. Вся игровая логика передается сюда.
* **Content Folder.** Здесь хранится весь игровой контент, например, звуки, 2D- и 3D-графика и шрифты. Объект **Content** предоставляет доступ к содержимому игры. Следует отметить, что добавление в проект различных текстур, аудиофайлов, шрифтов и т.п. происходит через **MonoGame Pipeline** – утилиту, которая преобразует исходные файлы в файлы **.xnb**, с которыми уже идет непосредственная работа приложения. [5]

Наглядно структуру проекта представлена на рисунке 1.

  
Рисунок 1 – Обозреватель решений

В классе **Game1** файла **Game1.cs** используются следующие методы:

* **protected override void Initialize().** Этот метод позволяет до запуска игры настроить всё, что не связанно с непосредственно содержимым игры (именно настроить, а не загрузить их извне). Запускается один раз самым первым из методов.
* **protected override void LoadContent().** Этот метод загружает содержимое (например, текстуры, звук, шрифты) в память перед началом игры. Как и **Initialize**(), он вызывается один раз при запуске приложения.
* **protected override void Update(GameTime gameTIme).** Этот метод вызывается один раз для каждого цикла игры. Здесь мы обновляем состояния любого объекта или переменной, используемых в игре. Сюда входят такие свойства объектов, как положение, скорость или цвет. Кроме того, здесь обрабатывается ввод данных пользователем. Иными словами, этот метод обрабатывает всю игровую логику, кроме рисования объектов на экране.
* **protected override void Draw(GameTime gameTime).** Этот метод отвечает за рисование объектов на экране с учетом положения, заданного с помощью метода **Update**.
* **protected override void UnloadContent().** При выходе из игрового цикла управление передается в метод **UnloadContent()**. В этом методе происходит выгрузка ранее использованных ресурсов. После этого игра завершается, и приложение закрывается. [9]

В итоге имеем игровую петлю такого вида (Рисунок 2):

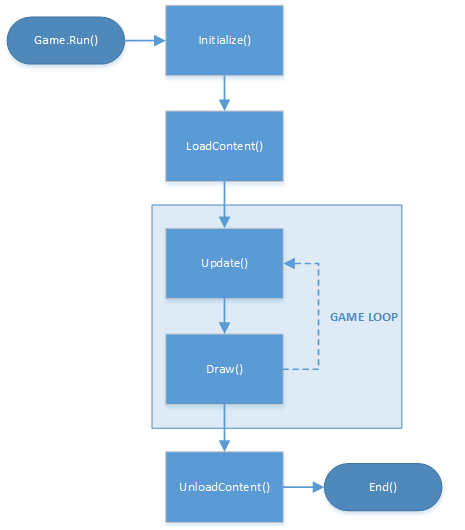


Рисунок 2 – Игровой цикл

* 1. **Работа со спрайтами**

Спрайт – графический объект в [компьютерной графике](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0). Чаще всего – [растровое](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) [изображение](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B5_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), которое можно [отобразить](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3) на экране. Наблюдение спрайта в трёхмерном пространстве под несоответствующим углом приводит к разрушению иллюзии. То есть легче всего воспринимать спрайт как перемещающуюся в пространстве проекцию какого-то объёмного тела так, чтобы разница была незаметна. [10]

В классе **Game1** используем объекты следующих классов:

* **GraphicsDeviceManager**: Находится в системе **Microsoft.Xna.Framework namespace** – это абстракция графического оборудования, на котором работает игра, что позволяет сосредоточиться на создании логики игры, а не тратить время на написание низкоуровневых драйверов для конкретных аппаратных функций. Класс **GraphicsDeviceManager** обрабатывает инициализацию оборудования и предоставляет вам доступ к таким вещам, как ориентация дисплея, полноэкранный режим, размер буфера и многое другое.
* **SpriteBatch**: находится в системе **Microsoft.Xna.Framework.Graphics namespace**. Класс **SpriteBatch** используется для объединения нескольких вызовов **Draw** в один блок работы, который будет отправлен на графический процессор устройства. Это намного эффективнее, чем отправлять команды рисования по отдельности, что приводит к более высокой (и, следовательно, более гладкой) частоте кадров.
  1. **Работа со шрифтами**

Шрифты, как и любой другой контент для игры, должны быть загружены через утилиту **Pipeline**. Однако в данном случае, загружается не конкретный шрифт, а создается новый элемент для описания используемого шрифта.

Довольно просто поместить изображение некоторого статического текста на экран, но для этого необходимо создать объект класса **SpriteFont**.

**SpriteFont** состоит из двух файлов. Первый - это **XML-файл**, который содержит определение способа отображения **SpriteFont**. Второй файл представляет собой отдельное графическое изображение, содержащее каждый символ шрифта, с размером, указанным в файле **XML**.

Изображение загружается с помощью **MonoGame** (в методе **LoadContent()**), а затем нарезается на отдельные двумерные изображения (известные как спрайты), которые затем можно использовать для отображения текста на экране с помощью метода **DrawString ()** объекта **SpriteBatch**.

* 1. **Описание основных игровых элементов**

Для описания корабля игрока, кораблей врагов и выстрелов были созданы классы **Spaceship**, **Enemy**, **Shot** и **ShotEnemy**.

**Класс Spaceship.** Данный класс предназначен для описания и управления спрайтом-кораблем главного героя. В нем разработаны следующие основные методы:

* Методы **Up**(), **Down()**, **Left()**, **Right()** обеспечивают передвижение корабля

public void Up()

{

if (this.\_position.Y > 0)

{

this.\_position.Y -= \_speed;

}

}

public void Down()

{

if (this.\_position.Y < SpaceGame.Height - 91)

{

this.\_position.Y += \_speed;

}

}

public void Left()

{

if (this.\_position.X > 0)

{

this.\_position.X -= \_speed;

}

}

public void Right()

{

if (this.\_position.X < SpaceGame.Width - 80)

{

this.\_position.X += \_speed;

}

}

* Метод **DrawSpaceship**() отвечает за рисование корабля игрока на экране с учетом заданного положения

public void DrawSpaceship()

{

SpaceGame.SpriteBatch.Draw(Texture2DSpaceship, \_position, \_color);

}

**Класс Enemy**. Данный класс создан для описания кораблей противника. В нем разработаны следующие основные методы:

* Метод **Move**() обеспечивает передвижение корабля противника

public void Move()

{

\_position.Y += \_speed;

if (\_position.Y > SpaceGame.Height)

{

RandomSet();

}

}

* Метод **DrawEnemy**() отвечает за рисование корабля противника на экране с учетом заданного положения

public void DrawEnemy()

{

switch(\_enemy)

{

case Enemies.Enemy1:

Texture2DEnemy = Texture2DEnemy1;

break;

case Enemies.Enemy2:

Texture2DEnemy = Texture2DEnemy2;

break;

case Enemies.Enemy3:

Texture2DEnemy = Texture2DEnemy3;

break;

}

SpaceGame.SpriteBatch.Draw(Texture2DEnemy, \_position, \_color);

}

* Метод **RandomSet**() присваивает начальные координаты кораблю противника

public void RandomSet()

{

\_position = new Vector2(SpaceGame.GetIntRandom(100,

SpaceGame.Width - 100), SpaceGame.GetIntRandom(-500, -100));

}

**Классы Shot и ShotEnemy**. Данные классы предназначены для описания выстрелов корабля игрока и кораблей противника соответственно. В нем реализованы следующие методы:

* Метод **UpdateShot**() отвечает за обновление координат выстрела

public void UpdateShot()

{

if (\_position.Y >= 0)

{

\_position += \_direction;

}

}

* Метод **DrawShot**() отвечает за рисование выстрела на экране с учетом заданного положения

public void DrawShot()

{

SpaceGame.SpriteBatch.Draw(Texture2DShot, \_position, \_color);

}

Для имитации движения в космическом пространстве был разработан класс **Star**, в котором реализованы методы, отвечающие за задание звезде начальной координаты и цвета, а также методы, обеспечивающие изменение координаты звезды и ее прорисовку.

Основной игровой процесс описан в классе **SpaceGame**. В данном классе создаются объекты классов **Star**, **Spaceship**, **Enemy**, **Shot** и **ShotEnemy**.

Метод **Initialization(SpriteBatch spriteBatch, int width, int height)** класса **SpaceGame** задает начальные координаты игрового поля, а также начальные координаты корабля игрока, кораблей врагов, скорость движения, начальные координаты и цвет каждой звезды.

Метод **Draw**() класса **SpaceGame** рисует все элементы игры (корабль игрока, звезды, корабли врагов, выстрелы, количество жизней у игрока и количество набранных очков).

Метод **Update**() класса **SpaceGame** обновляет состояния всех объектов, находящихся на игровом поле.

Метод **GameOver**() класса **SpaceGame** отслеживает конец игры. Если игрок проиграл, данный метод удаляет с игрового все выстрелы и обнуляет счет.

Метод **AddToTxt**() класса **SpaceGame** записывает в файл лучший счет игрока.

Метод **AddCurrentScore()** класса **SpaceGame** записывает в файл счет игрока.

* 1. **Взаимодействие объектов**

Одним из ключевых моментов в игре является столкновения игровых объектов. Например, корабль игрока сталкивается с кораблем противника, выстрел попадает во врага.

К счастью, проверка столкновений двухмерных объектов в MonoGame довольно проста. Каждый спрайт представляет некоторое прямоугольное пространство. Проверка на пересечение его с другим прямоугольником фактически будет означать проверку на столкновение. [11]

Прямоугольник в MonoGame представлен классом **Rectangle**. Его метод **Intersects**() позволяет узнать, пересекается ли он с другим прямоугольником.

Формируем два прямоугольника и применяем метод **Intersects**(). Если он возвращает *true*, значит оба прямоугольника пересекаются.

В игре проверяются столкновения трёх типов: столкновение противников с выстрелом игрока, столкновение игрока с выстрелом противника и столкновение игрока с противником.

Для отслеживания столкновения спрайтов были разработаны следующие методы:

* Метод **CollideShipAndEnemy(Enemy enemy)** проверяет, столкнулся ли корабль игрока с кораблем противника

private static bool CollideShipAndEnemy(Enemy enemy)

{

int width = 1, height = 1;

Rectangle ship = new Rectangle((int)Spaceship.GetPosition.X,

(int)Spaceship.GetPosition.Y, 76, 88);

switch (enemy.GetEnemy())

{

case Enemies.Enemy1:

width = 76;

height = 65;

break;

case Enemies.Enemy2:

width = 64;

height = 86;

break;

case Enemies.Enemy3:

width = 86;

height = 75;

break;

}

Rectangle enemies = new Rectangle((int)enemy.GetPosition.X,

(int)enemy.GetPosition.Y + 15, width, height);

return ship.Intersects(enemies);

}

* Метод **CollideEnemyAndShipShot(Enemy enemy, Shot shots)** проверяет, попал ли выстрел игрока в корабль противника

private static bool CollideEnemyAndShipShot(Enemy enemy, Shot shots)

{

int width = 1, height = 1;

Rectangle shot = new Rectangle((int)shots.GetPosition.X,

(int)shots.GetPosition.Y, 60, 35);

switch (enemy.GetEnemy())

{

case Enemies.Enemy1:

width = 76;

height = 65;

break;

case Enemies.Enemy2:

width = 64;

height = 86;

break;

case Enemies.Enemy3:

width = 86;

height = 75;

break;

}

Rectangle enemies = new Rectangle((int)enemy.GetPosition.X,

(int)enemy.GetPosition.Y, width, height);

return enemies.Intersects(shot);

}

* Метод **CollideEnemyShotAndShip(ShotEnemy shotEnemy)** проверяет, попал ли выстрел противника в корабль игрока

private static bool CollideEnemyShotAndShip(ShotEnemy shotEnemy)

{

Rectangle ship = new Rectangle((int)Spaceship.GetPosition.X,

(int)Spaceship.GetPosition.Y, 72, 88);

Rectangle shot = new Rectangle((int)shotEnemy.GetPosition.X,

(int)shotEnemy.GetPosition.Y, 20, 20);

return ship.Intersects(shot);

}

К сожалению, такая схема не позволяет с абсолютной точностью установить, столкнулся один объект с другим или нет, так как текстуры объектов не являются идеальными прямоугольниками. Но для данной игры этот факт не так уж и важен, и никаких проблем из-за этого замечено не было.

* 1. **Описание экранов игры**

Всего в игре есть четыре экрана: экран главного меню, игровой экран, экран паузы и экран завершения игры.

В главного меню игрок может ознакомиться с управлением, узнать лучший счет и информацию о разработчике игры (Рисунок 3).

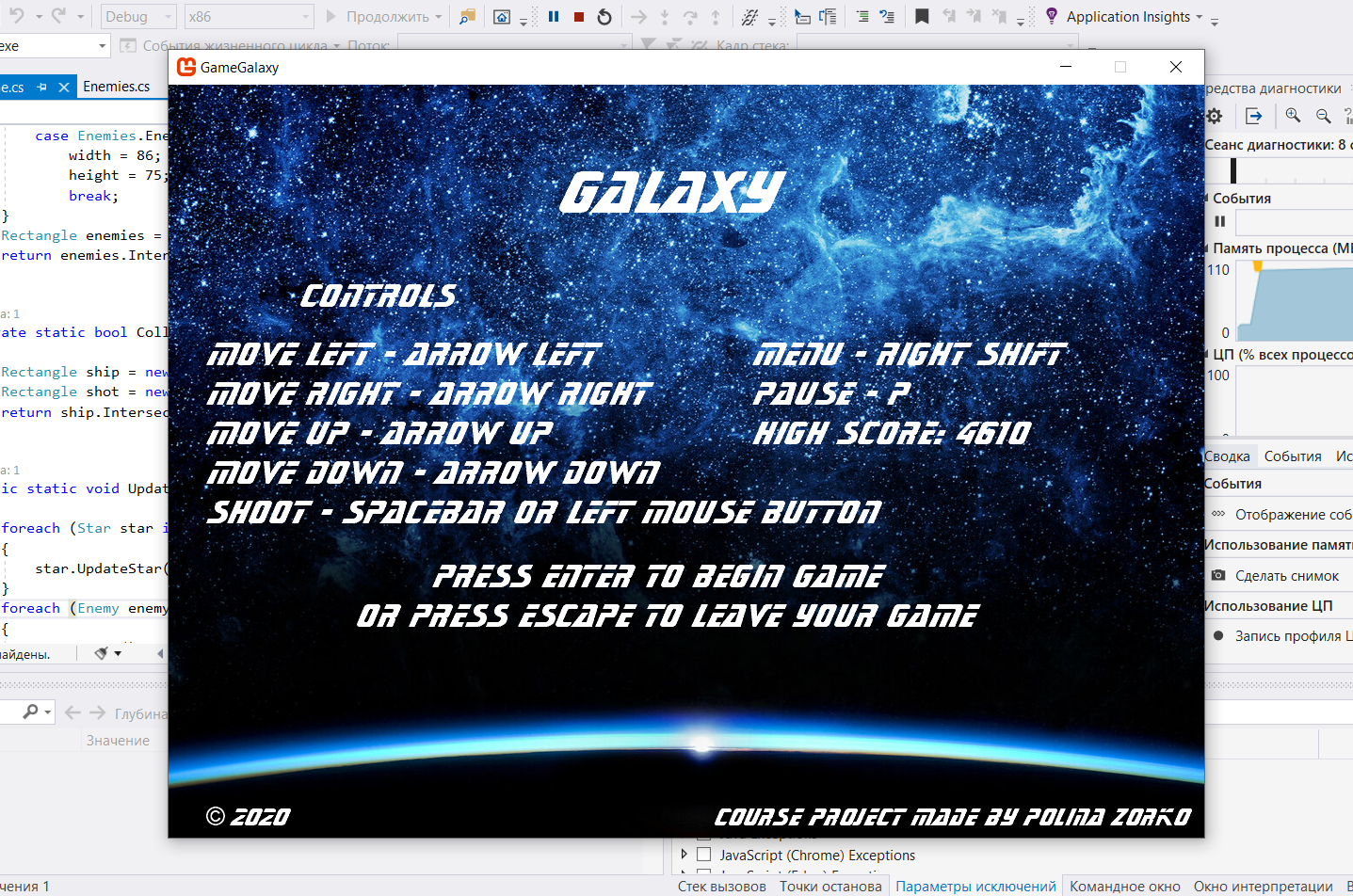


Рисунок 3 – Экран главного меню

Второй экран – это непосредственно сам игровой экран (Рисунок 4).

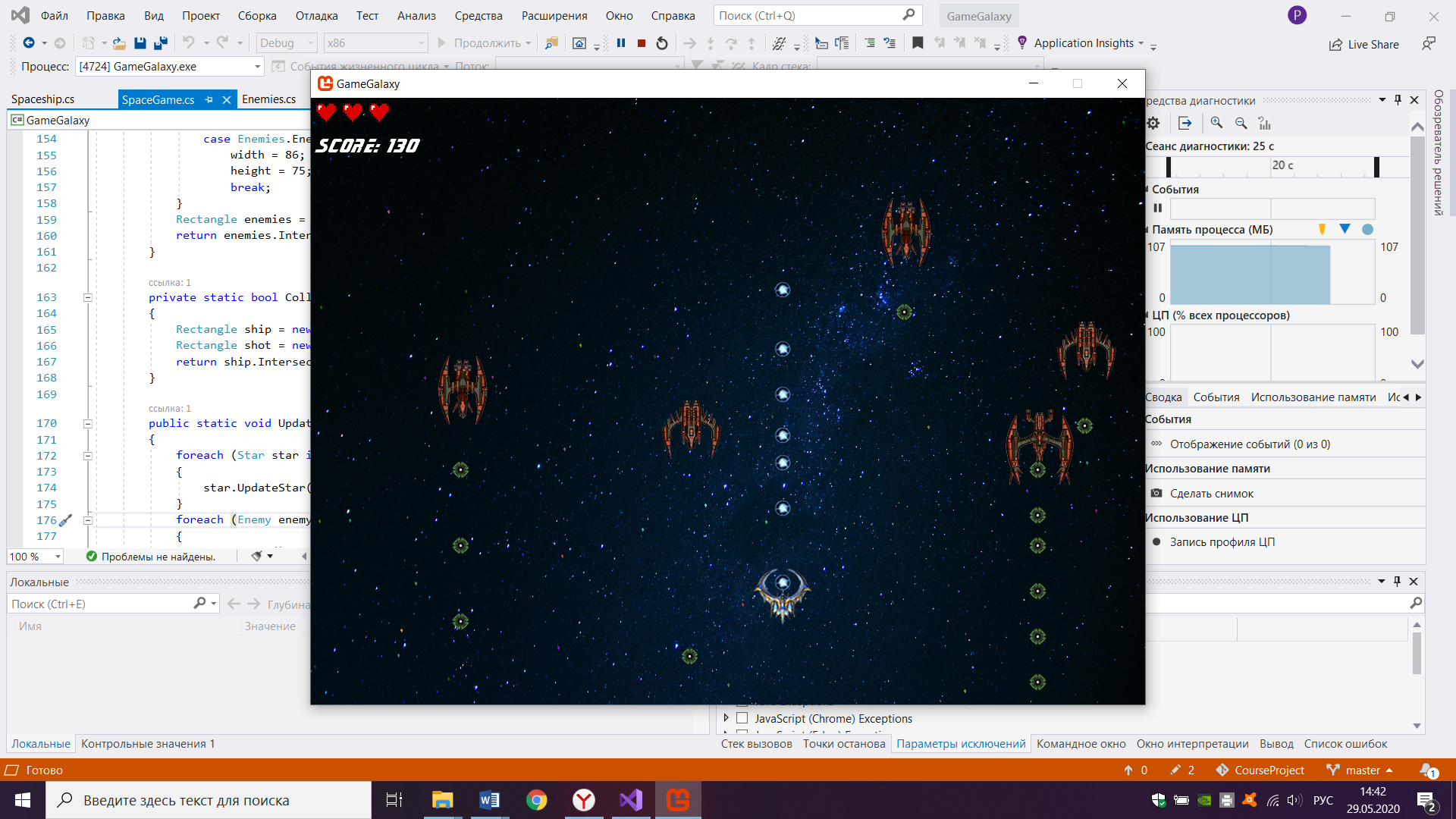


Рисунок 4 – Игровой экран

Во время игры игрок может набирать очки. Количество начисленных очков зависит от типа корабля врага, так за первый вражеский корабль (Рисунок 5) можно получить 10 очков, за второй (Рисунок 6) – 20, а за третий (Рисунок 7) – 30.



Рисунок 5 – Первый вражеский корабль



Рисунок 6 – Второй вражеский корабль



Рисунок 7 – Третий вражеский корабль

Если во время игрового процесса нажать клавишу P, игрок перейдет на экран паузы (Рисунок 8).

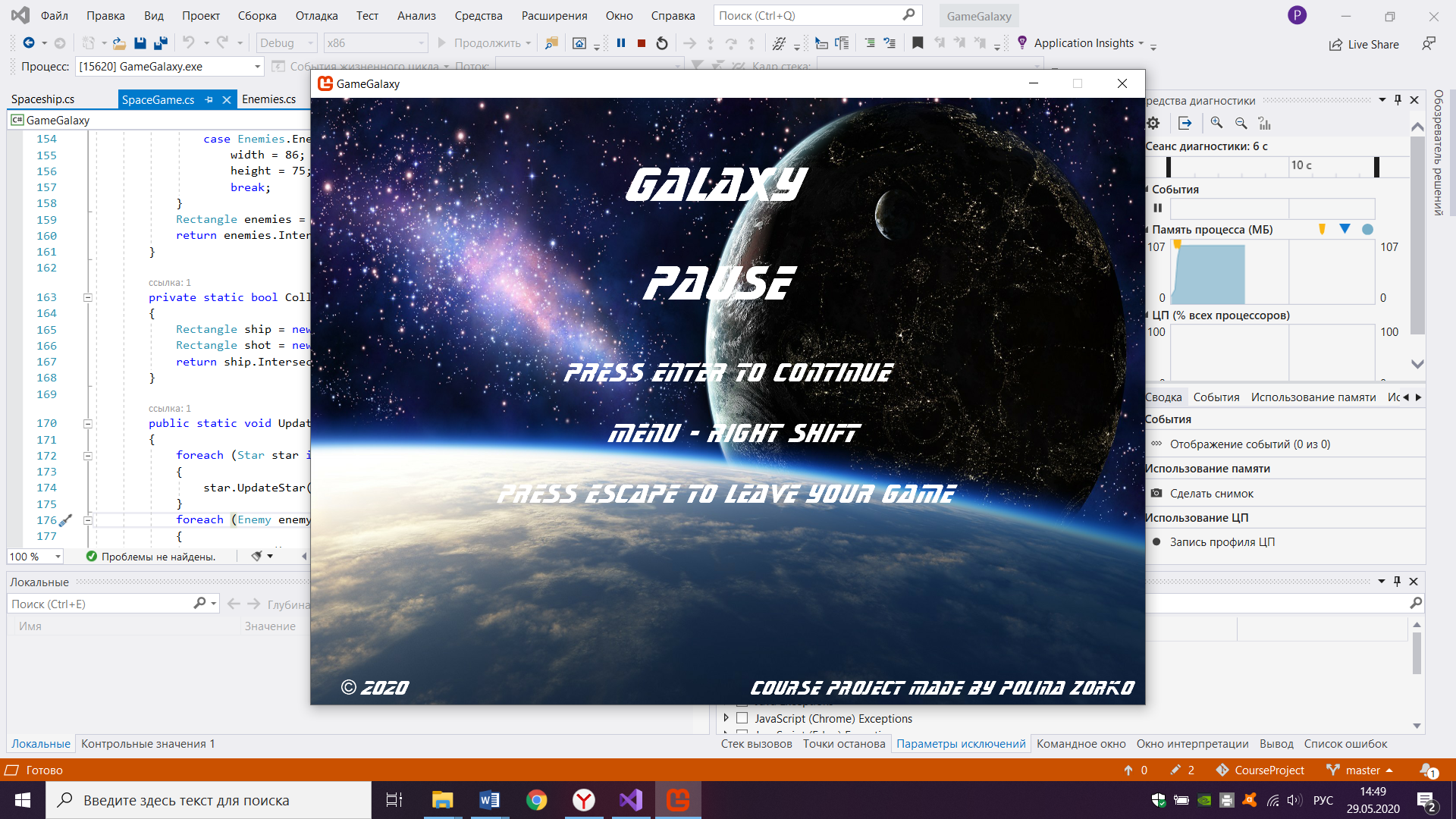


Рисунок 8 – Экран паузы

Затем игрок может выбрать продолжить ли ему игру или выйти из нее.

Если же у игрока закончились все игровые жизни, он переходит на экран завершения игры (Рисунок 9).

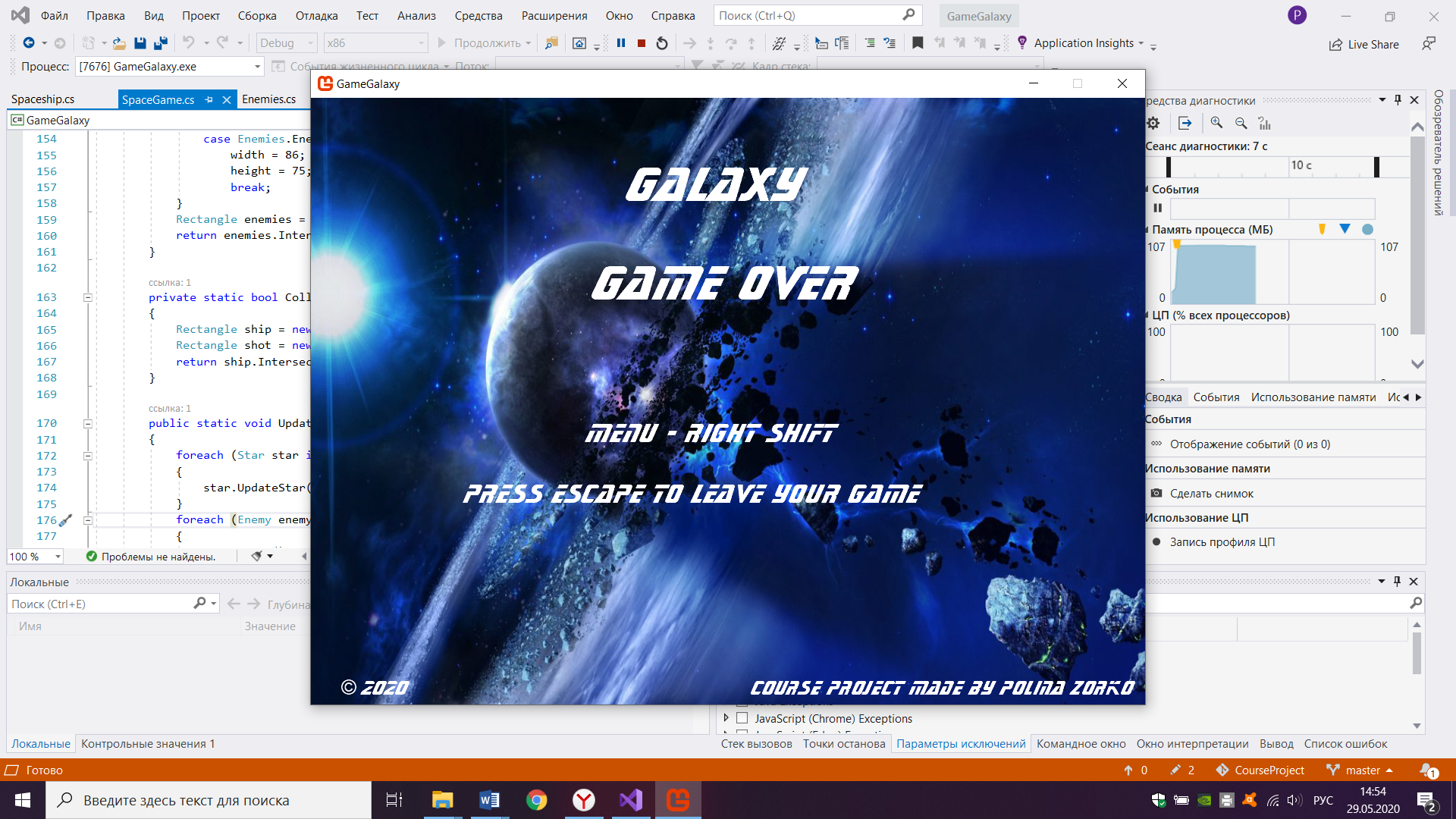


Рисунок 9 – Экран завершения игры

Весь игровой процесс сопровождается различными музыкальными эффектами.

Значения лучших результатов хранятся в отдельном файле **HighScore.txt**.

Если игрок нажмет клавишу **Escape**, то игра сразу же закроется, без сохранения каких-либо результатов.

Для удобного переключения между режимами игры было создано перечисление **State**. Для этого создаем поле **state** типа **State**. В зависимости от режима игры **state** может принимать значение **MainScreen**, **Game**, **GameOver** и **Pausе**. Так поле **state** дает игре понять, когда нужно совершить переключение. Так, например, если после запуска игры нажать клавишу **Enter**, значение поля **state** с **MainScreen** изменится на **Game**.

Заключение

Данная курсовая работа была выполнена в соответствии c поставленной задачей в среде **Microsoft Visual Studio 2019**. В процессе разработки приложения было проведено исследование компонентов программной среды **Microsoft Visual Studio 2019**, которые использовались при создании программы. Был изучен игровой движок **MonoGame**.

Разработка первой игры позволила получить более глубокие знания по языку **C#**. Также получены базовые знания по работе в программе **Adobe Photoshop CС**, которая использовалась для создания спрайтов.

Итогом курсовой работы стал игровой продукт «Галактика». Программа «Галактика» носит развлекательный характер, а также развивает внимательность и скорость реакции. Она подойдет, как взрослому человеку, так и ребенку дошкольного возраста.

Как и любое приложение, данный проект можно улучшить. В качестве ближайших улучшений можно рассматривать:

1. Создание различных уровней игры;
2. Увеличение разнообразия кораблей противника;
3. Добавление кораблям противника более сложных моделей движения;
4. Добавление возможности ловить бонусы (например, бонусная жизнь, двойной выстрел и т.п.).

Список используемых источников

1. Албахари, Джозеф, Албахари, Бен. С# 7.0 Справочник. Полное описание языка. : Пер. с англ. – СпБ. : ООО “Альфа-книга”, 2018. – 1024 с. : ил. – Парал. тит. англ.
2. Шилдт, Герберт. C# 4.0: полное руководство.: Пер. с англ. − М.: ООО «И.Д. Вильямс», 2011. − 1056 с.
3. Microsoft Visual Studio // Википедия: свободная энциклопедия [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio>. – Дата доступа: 23.04.2020.
4. Shoot ’em up // Википедия: свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Shoot\_’em\_up. – Дата доступа: 23.04.2020.
5. XNA is Dead; Long Live the New XNA, MonoGame // Сайт Code magazine [Электронный ресурс]. – Режим доступа: https://www.codemag.com/article/1411081. – Дата доступа: 23.04.2020.
6. Игровой движок // Википедия: свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Игровой\_движок. – Дата доступа: 23.04.2020.
7. Интегрированная среда разработки // Википедия: свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Интегрированная\_среда\_разработки. – Дата доступа: 23.04.2020.
8. Краткий обзор языка C# // Сайт Microsoft [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp. – Дата доступа: 23.04.2020.
9. Создание игры UWP на двухмерной платформе MonoGame // Сайт Microsoft [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/windows/uwp/get-started/get-started-tutorial-game-mg2d. – Дата доступа: 23.04.2020.
10. Спрайт (компьютерная графика) // Википедия: свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Спрайт\_(компьютерная\_графика). – Дата доступа: 23.04.2020.
11. Столкновение спрайтов // Сайт о программировании [METANIT.COM](https://metanit.com/). [Электронный ресурс] – 2012. – Режим доступа: https://metanit.com/sharp/monogame/2.8.php. – Дата доступа: 23.04.2020.