# **УДК 004.4'418**

#

# **ТЕХНОЛОГИИ ИССЛЕДОВАНИЯ И ИЗМЕНЕНИЯ .NET ПРОГРАММ НА ПРИМЕРЕ СОЗДАНИЯ МОДИФИЦИРОВАННОЙ ВЕРСИИ ИГРЫ TERRARIA**

# **Носарев П.Ю.1, Скороходов М.А.1**

# **1** Нижнетагильский государственный социально-педагогический университет (ф) РГППУ в г. Нижнем Тагиле, email: nosare@ya.ru; mishaskorohodov@yandex.ru

# **Аннотация**

В данной статье рассматривается создание программного обеспечения для увеличения функционала игры Terraria. С помощью него можно будет добавлять новые вещи, убирать недоработки игры а также закрывать уязвимости и для этого не нужно быть хорошим программистом. В связи с тем что обычно для добавления нового контента и выполнения множества доработок внутри игры требуются хорошие навыки в программировании и поэтому важно создать такое ПО, которое будет помогать простым людям без навыков улучшать игру и свой игровой опыт. Также в статье рассматриваются преимущества данного ПО, его положительные стороны.

**Ключевые слова:** C#, .NET, CIL, модификация программ, реверс-инженеринг

**TECHNOLOGIES FOR RESEARCHING AND CHANGING .NET PROGRAMS ON THE EXAMPLE OF CREATING A MODIFIED VERSION OF THE GAME TERRARIA**

# **Nosarev P.U.1, Skorokhodov M.A. 1**

# **1** Nizhny Tagil State Social and Pedagogical University (bo.) Russian State Vocational Pedagogic University, e-mail: nosare@ya.ru; mishaskorohodov@yandex.ru

# **Abstract**

# This article discusses the creation of software to increase the functionality of the Terraria game. With it, you can add new things, remove game flaws and close vulnerabilities, and you don't need to be a good programmer for this. Due to the fact that usually good programming skills are required to add new content and perform many improvements inside the game, and therefore it is important to create such software that will help ordinary people without skills to improve the game and their gaming experience. The article also discusses the advantages of this software, its positive aspects.

# **Keywords:** C#, .NET, CIL, program modification, reverse engineering

**Введение**

“Частное или общее?” – один из многих вопросов, на какие не может быть дан ответ в той системе понятий, что обнаруживает ограниченность человеческого мышления, завязанного на категориях, операции над которыми зачастую приводят к недопустимым упрощениям, какие в последствии исключают множество вариантов решения конкретного вопроса.
Насколько программы должны быть самодостаточными или зависимыми от исполняющей среды?

Стремительное развитие информационных технологий практически гарантирует то, что многие из нас так или иначе были вынуждены работать с компьютерными программами. Но всегда ли это был положительный опыт?

Программы достаточно сложно спроектировать так, чтобы их можно было легко исправить под свои нужды в достаточной на той мере, так как всегда будет некоторый неизменный фундамент, - выделение всех абстракций и иерархий в приложении может обеспечить теоретическую возможность вносить даже самые безумные изменения минимальными правками, но практическую - навряд ли, так как возрастание числа сущностей неизменно ведет к возрастанию требований к желающему внести правки - так что некоторые программы необходимо иметь возможность изменить.

Разные подходы к разработке программ обнаруживают у этой ситуации несколько возможных путей.

Некоторые программы компилируются непосредственно в машинный код, низкоуровневый язык, изобилующий необходимыми для достижения максимальной производительности и эффективности по памяти с учетом архитектурных особенностей командами, использование которого для разработки прикладных программ едва ли оправдано. Возрастающее в экспоненциальной прогрессии разнообразие цифровых устройств и соответствующих им программных платформ заставило разработчиков по всему миру задуматься о необходимости абстрагироваться от аппаратных особенностей и переложить заботы о них на плечи виртуальной машины, программы, транслирующей некий промежуточный, доступный для трансляции большинством платформ, код в машинный. Логично предположить, что раз программы для виртуальных машин написаны на некотором более абстрактном языке, лишенном всяких умопомрачительных сложностей, связанных с внутренним устройством даже самого простого вычислительного устройства, то и исследовать, и изменять такие программы будет легче. Но все ли однозначно относительно того, должна ли программа быть доступной для исследования и изменения?

# **Обзор предметной области**

# **NET Framework**

# .NET Framework - совокупность программных средств, разработанная Microsoft с целью реализации экосистемы зависимых приложений, т.е. выполняющихся в новом режиме исполнения посредством дополнительного программного слоя, позволяющего абстрагироваться от проблем кроссплатформенности, подключения большинства библиотек и использования разных технологий для разных языков.

# Платформа позволяет работать на разных языках в рамках одного проекта. Это реализуется за счет компиляции всех файлов с кодом в промежуточный язык Common Intermediate Language (CIL), предоставляющего единый интерфейс взаимодействия компонентов, написанных на разных языках. CIL код впоследствии компонуется с метаданными, формируя сборку, программу, которую можно запустить с помощью CLR (Common Language Runtime, виртуальной машины), где разные его компоненты будут скомпилированы лишь тогда, когда понадобятся компилятором JIT (Just In Time) или преобразовать в машинный (с некоторыми ограничениями) для конкретной процессорной архитектуры с помощью NGen.exe

# **C#**

# Используемый в проекте язык программирования C# является объектно-ориентированным языком с C-подобным синтаксисом, наиболее похожим на таковой у С++ и Java, языков, входящих в пятерку наиболее популярных по данным индекса TIOBE (<https://tiobe.com/tiobe-index/>), что делает его изучение легче для уже постигавших на эти языки людей и перспективнее для тех, кто собирается. Имеет статическую типизацию, что обеспечивает лучшую по сравнению с языками с динамической типизацией производительность, а также и худшие возможности совершить ошибку из-за неявных преобразований типов, когда разрешены неявно только расширяющиеся преобразования между однородными типами, что важно для бизнеса, на который изначально была нацелена разработка языка. Поддерживает полиморфизм, [перегрузку многих операторов](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2), в том числе и явного и неявного приведений типов, [делегаты](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%B3%D0%B0%D1%82_%28%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%29), атрибуты, [события](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%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), [переменные](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%28%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%29), [свойства](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_%28%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%29), [обобщённые](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%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) типы и методы, [итераторы](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80), [анонимные функции](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%BE%D0%BD%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) с поддержкой [замыканий](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_%28%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%29), [LINQ](https://ru.wikipedia.org/wiki/Language_Integrated_Query), [исключения](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), [комментарии](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%B8%D0%B8_%28%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%29) в формате [XML](https://ru.wikipedia.org/wiki/XML).

# Система пользовательских типов в C# представлена следующими компонентами:

* Классы – шаблоны для объектов общего назначения, по умолчанию копируемые по ссылке, или хранилища независимых от конкретных экземпляров методов и данных (статические классы)
* Структуры – шаблоны для объектов общего назначения, по умолчанию копируемые по значению, то есть каждый раз с заемом участка памяти для уже существующих на месте оригинального объекта байт
* Интерфейсы – контракты, предъявляющие всем реализующим его типам требования в наличии определенных членов.
* Перечисления – списки именованных констант, обычно используемых для представления именованных пар имя – число.
* Делегаты - дескрипторы методов с определенной сигнатурой, т.е. спецификацией, определяющей принимаемые типы и возвращаемый тип

Все типы, кроме делегатов, содержат члены. Так, перечисления содержат пары значений имя элемента – его номер (опционально), интерфейсы содержат сигнатуры методов (в том числе и конструкторов, свойств), классы и структуры содержат поля помимо методов.

# **Ildasm.exe**

# Ildasm.exe является дизассемблером CIL, программой с графическим интерфейсом, позволяющей получить CIL код для уже скомпилированных программ. Позволяет сохранять код в компилируемые в последующем с помощью Ilasm.exe (ассемблера CIL) проекты (Файл -> Дамп или комбинация клавиш Ctrl + D)

# **DnSpy**

Представляет из себя отладчик и редактор .NET сборок. Предоставляет неопытному пользователю изменять программы без углубленных познаний о CIL. Не подходит, если распространение измененных версий Terraria недопустимо, какое может быть в силу многих причин – проверить небольшой проект, изменяющий террарию, гораздо быстрее, чем истерзанную в разных местах 17-мегабайтовую свалку кода разного качества.

## **Terraria**

Конкретный объект исследований – Terraria – [компьютерная игра](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%B8%D0%B3%D1%80%D0%B0) в жанре [приключенческой](https://ru.wikipedia.org/wiki/Action-adventure) [песочницы](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%81%D0%BE%D1%87%D0%BD%D0%B8%D1%86%D0%B0_%28%D0%B6%D0%B0%D0%BD%D1%80%29), разработанная американской студией [Re-Logic](https://ru.wikipedia.org/wiki/Re-Logic)
Рис. 5. Геймплей Terraria

# **ГЛАВА 2. Практическая часть - фундамент**

## Концепция вносимых изменений

Взаимодействие с программой через текст не зря выбирают все начинающие программисты за его простоту. Игра имеет чат, открываемый по нажатии клавиши Enter, куда можно набрать сообщение и последующим нажатием этой же клавиши отправить его на сервер или в чат локального мира. Первый этап разработки заключается в поиске метода, обрабатывающего отправку сообщения на сервер, чтобы его можно было перехватить и внедрить в него логику, добавляющую террарии возможность выполнения команд на своей стороне.

## C:\Users\Paul\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Untitled Diagram (6).pngРис 2. Диаграмма последовательности проекта

## **Обнаружение метода**

В силу не самой качественной кодовой базы проекта, этот этап может занять куда больше времени, чем покажется на первый взгляд. Беглый анализ структуры проекта с сопоставлением реализованного через отдельные классы с выделенными иерархиями в отдельных пространствах имен позволяет установить, что все, что было разработано и с тех пор благополучно забыто до 2013 года, так или иначе находится в одном монолитном классе Terraria.Main. В случае, если вместо монолита было бы нескончаемое множество маленьких непонятно оправдывающих ли свое существование классов, для поиска можно было бы использовать Mono.Cecil, учитывая, что событие так или иначе связано с нажатием клавиши Enter, какое обнаруживается скорее всего с помощью метода KeyboardState.IsKeyPressed с соответствующим значением Key = Keys.Enter. Наиболее простым методом в данной ситуации является поиск по идентификаторам сборки. Идентификаторы используются для различения отвечающих за разные объекты вещи, чаще всего даются осмысленные, позволяющие интуитивно отнести идентифицируемый объект с отвечающей за конкретную функцию подсистемой. В данном случае, поиск по слову chat позволяет обнаружить поле, в котором хранится набранный чат текст.


Рис. 3. Результаты поиска по слову chat

Последующий анализ использования поля приводит нас к методу DoUpdate\_HandleChat, какой очень вероятно выполняет нужное нам действие исходя из своего названия.



Рис. 4. Окно анализатора

## **Анализ функциональных частей**

private static void DoUpdate\_HandleChat()
{
    if (Main.CurrentInputTextTakerOverride != null) // начало части, отвечающей за предотвращение отрисовки чата при некоторых условиях, например, в случае существования объекта, перехватывающего текстовый поток, в случае, если открыт интерфейс изменения текста таблички, или если игрок использует геймпад
    {
        Main.drawingPlayerChat = false;
        return;
    }
    if (Main.editSign)
    {
        Main.drawingPlayerChat = false;
    }
    if (PlayerInput.UsingGamepad)
    {
        Main.drawingPlayerChat = false;
    } // конец части
    if (!Main.drawingPlayerChat)
    {
        Main.chatMonitor.ResetOffset(); // обращение ранее установленного сдвига в отображении сообщений, какое можно задать с помощью стрелок вверх вниз
        return;
    }
    int linesOffset = 0;
    if (Main.keyState.IsKeyDown(Keys.Up)) // обновление сдвига
    {
        linesOffset = 1;
    }
    else if (Main.keyState.IsKeyDown(Keys.Down))
    {
        linesOffset = -1;
    }
    Main.chatMonitor.Offset(linesOffset); // изменение данных класса, отвечающего за отрисовку чата, в соответствии с сдвигом
    if (Main.keyState.IsKeyDown(Keys.Escape)) // выход из чата по нажатии Esc
    {
        Main.drawingPlayerChat = false;
    }
    string a = Main.chatText;
    Main.chatText = Main.GetInputText(Main.chatText, false);
    int num = (int)((float)Main.screenWidth \* (1f / Main.UIScale)) - 330;
    if (a != Main.chatText) // код, отвечающий за предотвращение выхода текста из чата за пределы экрана
    {
        for (float x = ChatManager.GetStringSize(FontAssets.MouseText.Value, Main.chatText, Vector2.One, -1f).X; x > (float)num; x = ChatManager.GetStringSize(FontAssets.MouseText.Value, Main.chatText, Vector2.One, -1f).X)
        {
            int num2 = Math.Max(0, (int)(x - (float)num) / 100);
            Main.chatText = Main.chatText.Substring(0, Main.chatText.Length - 1 - num2);
        }
    }
    if (a != Main.chatText)
    {
        SoundEngine.PlaySound(12, -1, -1, 1, 1f, 0f);
    }
    if (Main.inputTextEnter && Main.chatRelease) // код, отвечающий за событие отправки сообщения. InputTextEnter, судя по всему, равен true тогда, когда в текущем обновлении нажат Enter при открытом чате, chatRelease, отвечает за похожее состояние, называемом “высвобождение чата», включающее в себя также закрытие чата без отправки сообщения по Esc
    {
        if (Main.chatText != "") // нет смысла отправлять пустое сообщение
        {
            ChatMessage message = ChatManager.Commands.CreateOutgoingMessage(Main.chatText);
            if (Main.netMode == 1) // netMode отвечает за сетевой режим. Равен 0, если это одиночная игра, 1, если многопользовательская и террария в ней клиент
            {
                ChatHelper.SendChatMessageFromClient(message);
            }
            else if (Main.netMode == 0)
            {
                ChatManager.Commands.ProcessIncomingMessage(message, Main.myPlayer);
            }
        }
        Main.chatText = "";
        Main.ClosePlayerChat();
        Main.chatRelease = false;
        SoundEngine.PlaySound(11, -1, -1, 1, 1f, 0f);
    }
}

Листинг 1. Код метода Terraria.Main.DoUpdate\_HandleChat

Исходя из листинга 1, встраиваемый метод, вызывающий обработчик команд, должен располагаться до вызовов ChatHelper.SendChatMessageFromClient или ChatManager.Commands.ProcessIncomingMessage и после проверки сообщения на предмет пустоты. Это реализуемо несколькими способами. Один из них заключается в обертывании части от создания экземпляра объекта ChatMessage до блока else if в тело условия, выражением которого будет проверка возвращаемого обработчиком команд логического значения. Для простоты реализации легче предусмотреть логику, из которой метод будет возвращать true, если сообщение необходимо отправить, то есть в случае, когда набранный текст не являлся командой, и false в противоположной ситуации. Таким образом, код в теле условия с выражением Main.chatText != “” приобретет вид

if (Main.chatText != "") // there is no point in sending an empty message

{

 if (MyTerrariaFunctions.ExecuteCommand(Main.chatText))

 {

 ChatMessage message = ChatManager.Commands.CreateOutgoingMessage(Main.chatText);

 if (Main.netMode == 1)

 {

ChatHelper.SendChatMessageFromClient(message);

 }

 else if (Main.netMode == 0)

 {

 ChatManager.Commands.ProcessIncomingMessage(message, Main.myPlayer);

 }

 }

}

Листинг 2. Измененный вариант метода DoUpdate\_HandleChat

Можно заметить, что к данному варианту применима оптимизация, суть которой заключается в образовании сложного условия с дизъюнкцией из двух условий, где одно полностью вложено в другое. Окончательный вариант измененного метода отличается от оригинального лишь выражением условия Main.chatText != “”, - в измененном методе оно принимает вид Main.chatText != “” && MyTerrariaFunctions.ExecuteCommand(Main.chatText)

if (Main.chatText != "" && MyTerrariaFunctions.ExecuteCommand(Main.chatText)))

{

ChatMessage message = ChatManager.Commands.CreateOutgoingMessage(Main.chatText);

…

}

Листинг 3. Окончательная высокоуровневая версия изменения

## **Анализ некоторых необходимых команд CIL**

Следующий этап разработки невозможен до тех пор, пока неизвестны базовые принципы работы CLR.

Виртуальная машина CLR упрощенно представляет из себя стековую машину с регистрами – ячейками памяти текущего фрейма, где хранится состояние метода (переменные).

Часть команд предназначена для работы с стеком – с помещением на него объектов:

* Констант (целого типа – ldc.i4, ldc.i8, вещественного – ldc.r4, строкового – ldstr, токена метаданных - ldftn)
* Значений полей – (ldfld, ldsfld)
* Возвращаемых значений методов (call, callvirt, calli)

Часть для работы с «регистрами» и регистровой памятью:

* Загрузка значения из регистра – ldloc, ldarg
* Сохранение значения регистра – stloc, starg
* Выделение неуправляемой памяти - localloc

Часть команд нужна для преобразования типов (conv, call), часть для работы с массивами (ldelem, stelem), ссылочными значениями (ldind.ref, stind.ref), объектами (box, castclass, initobj), арифметикой (add, sub, mul, div).

Наибольший интерес для нас сейчас представляют команды управления потоком программы, сущности, задающей ветви кода и условия их выполнения.

Основной массив команд для управления потоком начинается с буквы B (branch - ветвь). Команды деляется по арности, по количество передаваемых им аргументов. Нульарные ничего не принимают, унарные принимают 1 аргумент, а бинарные 2.

 /\* 0x00329436 7E430B0004 \*/ IL\_016E: ldsfld string Terraria.Main::chatText

 /\* 0x0032943B 721F0D0070 \*/ IL\_0173: ldstr ""

 /\* 0x00329440 281002000A \*/ IL\_0178: call bool [mscorlib]System.String::op\_Inequality(string, string) // Main.chatText != “”

 /\* 0x00329445 2C3A \*/ IL\_017D: brfalse.s IL\_01B9 // if the method returns false, go to the IL\_01B9 label

 /\* 0x00329447 7E500F0004 \*/ IL\_017F: ldsfld class Terraria.Chat.ChatCommandProcessor Terraria.UI.Chat.ChatManager::Commands // start of code executed if both subconditions are true. So we need to add code between brfalse and ldsfld

 /\* 0x0032944C 7E430B0004 \*/ IL\_0184: ldsfld string Terraria.Main::chatText

 /\* 0x00329451 6F9F280006 \*/ IL\_0189: callvirt instance class Terraria.Chat.ChatMessage Terraria.Chat.ChatCommandProcessor::CreateOutgoingMessage(string)

 /\* 0x00329456 1305 \*/ IL\_018E: stloc.s V\_5

 // ... part of the code is omitted ...

 /\* 0x00329481 721F0D0070 \*/ IL\_01B9: ldstr ""

 /\* 0x00329486 80430B0004 \*/ IL\_01BE: stsfld string Terraria.Main::chatText // Main.chatText = “”

Main.chatText = “”

Листинг 4. CIL код обработчика события отправки сообщения в чат

Первое, что необходимо сделать – вызвать метод обработчика команд, передав его в аргументы Main.chatText. Для этого надо сначала загрузить значение поля на стек

ldsfld string Terraria.Main::chatText

Затем вызвать метод

call bool DVG.MyTerrariaFunctions::ExecuteCommand(string)

И в соответствии с возвращаемым значением - если метод вернул false, то прервать отправку - произвести переход

brfalse.s IL\_01B9

## **Формализация задания для реализации автоматизированного патча.**

Следующий этап характеризуется выработкой стратегии для автоматизации процесса модификации. Отправной точкой будет служить выбор инструкции, располагающейся как можно ближе к месту, в которое будет производиться инъекция. Можно подсчитать, сколько раз вызывается метод String.op\_Inequality и посчитать от него, можно взять загрузку на стек значения поя  Terraria.UI.Chat.ChatManager::Commands. Рассмотрим 1 вариант. Для начала, следует дойти до получения списка инструкций метода, для которого надо сначала получить тип, для которого надо получить модуль, для которого нужно загрузить сборку.



Рис. 5. Программа для автоматической модификации Terraria

**Практическая часть – прикладная часть**

## **Разработка библиотеки**

Back-end модифицированной террарии будет представлять из себя:

* Статический класс MyTerrariaFunctions, существующий только для того, чтобы вызывать его методы из террарии, какие просто делегируют полномочия другим классам
* Статический класс Client, реализующий большинство методов, такие как, например Initialize и OnChat
* Статический класс Commands, выступающий хранилищем для методов, инкапсулирующих действия, связанные с набором определенных команд
* Структура CommandArgs, хранящая в себе данные о введенной команде и ее параметрах
	+ SmartSplitAutomata – вспомогательный класс, реализующий особый метод определения границ аргументов с помощью конечного автомата, когда обычно разделителем считается пробел, а необычно – неэкранированные кавычки, ограниченные с 2 сторон пробелами или отсутствием символов, причем захват осуществляется без преобразования экранированных символов
* Структура ConversionIntermediate, реализующая возможность преобразования string в различные типы с помощью синтаксического сахара вокруг операторов явного приведения типа
* Класс-исключение RHTCException, переопределяющий метод ToString так, чтобы возвращалась только сообщение исключения
* Абстрактный класс Command, нечто вызываемое по CommandArgs, идентифицируемого по множеству имен, имеющее поле с описанием
* Класс ActionCommand, вызывающий конкретный метод
* Интерфейс ICommandProcessor, определяющий требования к агентам, реагирующим на событие попытки отправки сообщения в чат
* Класс CommandProcessor, реализующий набор команд, начинающихся с определенного символа
* Интерфейс IPlugin, определяющий метод void Initialize(), какой должны будут реализовывать всякие загружаемые через клиент модули



Рис. 6. Класс MyTerrariaFunctions



Рис. 7. Класс CommandProcessor



Fig. 8. Command base class

**Команда load**

Механизм загрузки плагинов реализуется через команду, первым аргументом которой указывается путь (относительный и ли абсолютный)к динамически подключаемой библиотеке с классом, реализующим интерфейс IPlugin



Рис. 10. Код метода LoadPlugin

**Практическая часть – плагины**

****

Рис. 15. Расширение, предоставляющее возможность копировать и вставлять части мира с помощью выделения в режиме камеры

# **Заключение**

В результате работы были достигнуты следующие задачи

* Произвести обзор предметной области и используемых в проекте средств и технологий
* Разработать программу, осуществляющую изменение Terraria таким образом, чтобы она стала поддерживать загружаемые извне программные модули.
* Разработать расширяющие функциональность Terraria модули, запускаемые модифицированной программой из предыдущей задачи Terraria

Компилируемые в промежуточный код программы имеют большой потенциал для использования в обучении программированию, так как представляют обучающемся стимулы для работы, превышающие таковые при разработке программ, умеющих выводить числа в консоль. Кроме этого, безопасность через неясность, хоть и неявно используемая некоторыми разработчиками, преследующими цель сделать исследование своих программ невозможным, является деструктивной практикой, доказавшей свою неэффективность ( [Кивино гнездо: О "взломе" Skype](http://www.computerra.ru/own/kiwi/546763/) (недоступная ссылка). КОМПЬЮТЕРРА-ONLINE (июль 2010). Дата обращения: 12 декабря 2012. [Архивировано](https://web.archive.org/web/20120312112507/http%3A/www.computerra.ru/own/kiwi/546763/) 12 марта 2012 года.). Пусть некоторые части программы и есть смысл защищать, таких частей немного, да и защищать их можно иными способами.

Полученный от исследования опыт может быть использован в будущих проектах, связанных с перспективными направлениями «динамическая кодогенерация», «полиморфные приложения», «реверс-инженеринг компьютерных программ»

# **Список литературы**

1. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов = Professional C# 5.0 and .NET 4.5. — М.: [«Диалектика»](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B8%D0%B0%D0%BB%D0%B5%D0%BA%D1%82%D0%B8%D0%BA%D0%B0_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2013. — 1440 с. — [ISBN 978-5-8459-1850-5](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F%3A%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785845918505).
2. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание = Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2013. — 1312 с. — [ISBN 978-5-8459-1814-7](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F%3A%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785845918147).
3. [Джеф Просиз](https://ru.wikipedia.org/wiki/%D0%94%D0%B6%D0%B5%D1%84_%D0%9F%D1%80%D0%BE%D1%81%D0%B8%D0%B7). Программирование для Microsoft .NET = Programming Microsoft .NET. — М.: [Русская редакция](https://ru.wikipedia.org/w/index.php?title=%D0%A0%D1%83%D1%81%D1%81%D0%BA%D0%B0%D1%8F_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2003. — С. 704. — [ISBN 5-7502-0217-8](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F%3A%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5750202178).
4. ["Standard ECMA-335: Common Language Infrastructure (CLI)"](http://www.ecma-international.org/publications/standards/Ecma-335.htm)*. ecma-international.org(6 ed.).*[ECMA](https://en.wikipedia.org/wiki/Ecma_International)*. June 2012.*[Archived](https://www.webcitation.org/6HfXxFXH8?url=http://www.ecma-international.org/publications/standards/Ecma-335.htm)*from the original on June 26, 2013*. Retrieved August 31, 2005*.*
5. ["Standard ECMA-334: C# Language Specification"](http://www.ecma-international.org/publications/standards/Ecma-334.htm). ecma-international.org (4 ed.). [ECMA](https://en.wikipedia.org/wiki/Ecma_International). June 2006. [Archived](https://web.archive.org/web/20101031042906/http%3A/www.ecma-international.org/publications/standards/Ecma-334.htm) from the original on October 31, 2010. Retrieved August 31,2005.
6. Абрамян М.Э. Visual C# на примерах (+ CD-ROM); БХВ-Петербург - М., 2008. - **685** c.
7. Нейгел Кристиан , Ивьен Билл , Глинн Джей , Уотсон Карли , Скиннер Морган C# 2008 и платформа .NET 3.5 для профессионалов; Вильямс - М., 2008. - **772** c.
8. “Drill Into .NET Framework Internals to See How the CLR Creates Runtime Objects”. Hanu Kommalapati and Tom Christian, 2018. Режим доступа: электронный ресурс https://blog.csdn.net/phiger/article/details/5566988
9. “Промежуточный язык CIL” Режим доступа: электронный ресурс <https://professorweb.ru/my/csharp/charp_theory/level1/1_8.php>
10. Cifuentes, Cristina (July 1994). ["6: Control Flow Analysis"](https://web.archive.org/web/20161122154356/http%3A/www.labri.fr/perso/fleury/download/papers/binary_analysis/cifuentes-thesis.pdf) (PDF). Reverse Compilation Techniques (Thesis). [Queensland University of Technology](https://en.wikipedia.org/wiki/Queensland_University_of_Technology). Archived from [the original](http://www.labri.fr/perso/fleury/download/papers/binary_analysis/cifuentes-thesis.pdf)(PDF) on 22 November 2016.
11. ["Standard ECMA-372 C++/CLI Language Specification"](https://web.archive.org/web/20080810165757/http%3A/www.ecma-international.org/publications/standards/Ecma-372.htm). ECMA. December 1, 2005. Archived from [the original](http://www.ecma-international.org/publications/standards/Ecma-372.htm) on August 10, 2008. Retrieved January 16, 2008.
12. ["Understanding .NET Just-In-Time Compilation"](http://blogs.telerik.com/justteam/posts/13-05-28/understanding-net-just-in-time-compilation)*. telerik.com. May 28, 2013.*[Archived](https://web.archive.org/web/20130611052214/http%3A/blogs.telerik.com/justteam/posts/13-05-28/understanding-net-just-in-time-compilation)*from the original on June 11, 2013*. Retrieved May 21, 2015*.*