УДК 004

Практическое применение парадигмы ООП при создании графического настольного приложения

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен практическое применение парадигмы ООП на примере создания графического приложения. Приложение создано с помощью языка программирования C#. Результатом исследование является графическое интерактивное приложение.
Ключевые слова: ООП, C#, графическое приложение

Practical application of the OOP paradigm in creating a graphical desktop application

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article considers the practical application of the OOP paradigm on the example of creating a graphical application. The application was created using the C# programming language. The result of the study is a graphical interactive application.
Keywords: OOP, C#, graphical application

1 Введение

1.1 Актуальность

Рассматриваемая проблема представляет из себя актуальную и значимую область исследования в современной науке о программировании. Исследуемая парадигма программирования позволяет структурировать код таким образом, чтобы он был более модульным, понятным и поддерживаемым, что особенно важно для сложных графических интерфейсов. В условиях быстрого развития технологий и увеличения требований к функциональности и производительности программного обеспечения, использование ООП становится неотъемлемой частью процесса разработки. Это связано с тем, что данный подход предоставляет мощные инструменты для инкапсуляции, наследования и полиморфизма, которые позволяют создавать гибкие и масштабируемые приложения. Кроме того, объектно-ориентированное программирование способствует улучшению взаимодействия между разработчиками, облегчая совместную работу и

ускоряя процесс разработки. Таким образом, изучение и применение исследуемой парадигмы в контексте настольных приложений имеет большое значение для повышения качества и эффективности программного обеспечения.

1.2 Обзор исследований

А.Н.Рыбалёв в своём труде предложил подход к программной реализации системы управления коммутационными аппаратами на основе технологии объектно-ориентированного программирования [1]. И.В.Палкина, Р.Т.Набиуллин и Н.И.Симакина посвятили статью парадигме объектно-ориентированного программирования в области разработке клиент-серверных приложений [2]. А.В.Денискин в своей статье рассмотрел такое свойство языка C# как многопоточность [3]. И.Д.Ноек и В.А.Демичев разработали компьютерную игру на языке C# с помощью методов объектно-ориентированного программирования и кроссплатформенной библиотеки Monogame [4]. Е.Яценко исследовала перспективы использования парадигмы ООП в разработке современных программ с помощью языка C# [5].

1.3 Цель исследования

Целью исследования является создание графического приложения с применением парадигмы ООП языка программирования C#.

2 Материалы и методы

Для создания программы используется объектно-ориентированный язык программирования C#, а также методы парадигмы ООП. В качестве IDE выступает Visual Studio 2019.

3 Результаты и обсуждения

В проекте создаются классы, которые разделяются на две папки – «Engine» и «Game». Это сделано для логического и структурного разделения кода. Данное действие позволяет четко отделить базовую функциональность движка игры от специфической логики и объектов разрабатываемой игры.

Папка «Engine» содержит классы, которые отвечают за базовую функциональность движка игры. Эти классы предоставляют общие инструменты и утилиты, которые могут быть использованы не только в рамках рассматриваемой игры, но также в различных других играх. Например, классы «FileSystem», «Render», «Resources», «Sound», «Time», «Transform», «Vector» и другие обеспечивают функции для работы с файлами, рендеринга графики, управления ресурсами, воспроизведения звуков, управления временем, трансформаций и векторов. Эти классы являются универсальными и могут быть переиспользованы в других проектах.

Папка «Game» содержит классы, которые специфичны для конкретной игры. Эти классы определяют игровую логику, объекты и сцены, которые характерны для данного приложения. Например, классы «Bomb», «Effect»,

«House», «Plane», «Scene» представляют собой конкретные игровые объекты и сцены, которые используются в этой игре. Этот функционал зависит от контекста игры и не может быть легко переиспользован в других проектах без изменений.

Данные решения в проекте основываются на парадигме объектно-ориентированного программирования (ООП) для структурирования и организации кода, что значительно улучшает его читаемость, модульность, расширяемость и поддерживаемость. Данный подход предоставляет механизмы, которые позволяют эффективно управлять сложностью проекта и обеспечивать его масштабируемость.

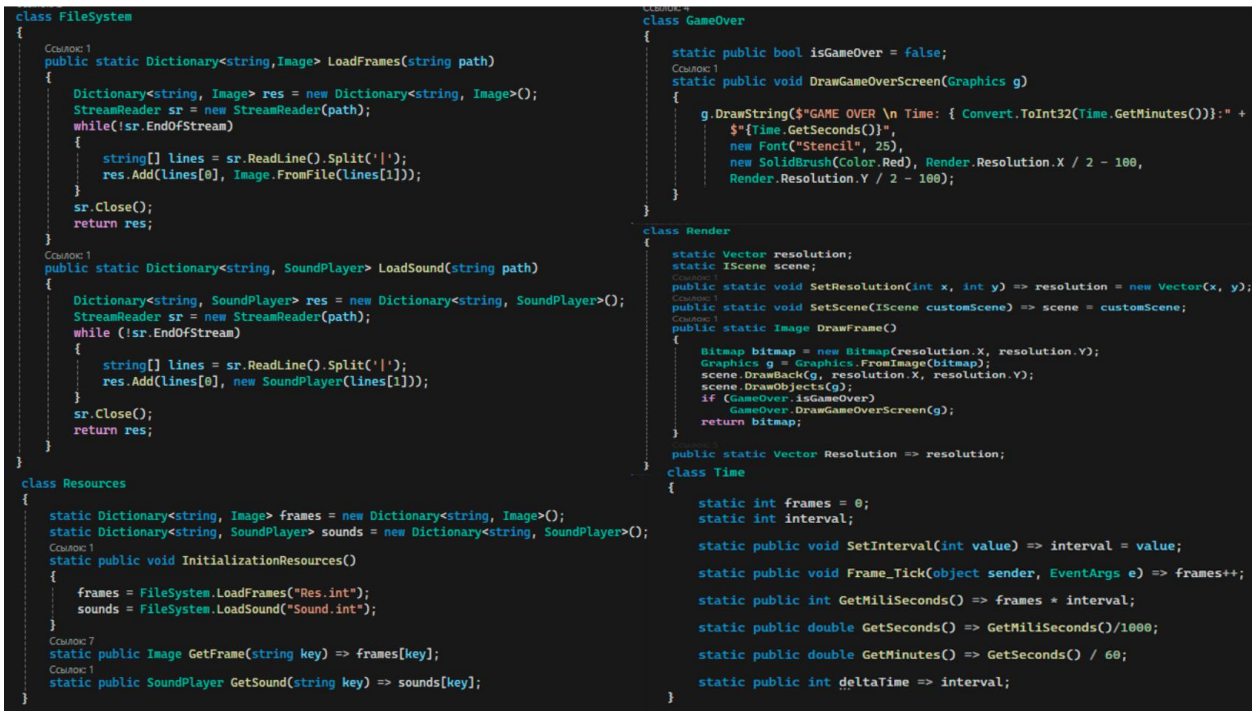
Модульность и инкапсуляция позволяют разделить программу на отдельные модули (классы), каждый из которых отвечает за определенную функциональность. Например, классы инкапсулируют соответствующие аспекты игры, такие как управление файлами, рендеринг, управление ресурсами, звук, время, трансформации и векторы. Это позволяет скрыть внутренние детали реализации и предоставить внешний интерфейс для взаимодействия, что уменьшает вероятность ошибок и упрощает изменение внутренней логики без влияния на другие части программы.

Наследование и полиморфизм позволяют создавать новые классы на основе существующих, что уменьшает дублирование кода. Например, классы «Bomb», «Effect», «House», «Plane» наследуются от класса «Transform» и реализуют интерфейс «IGameObject», что обеспечивает единообразие в работе с различными игровыми объектами. Полиморфизм позволяет использовать объекты разных классов через общий интерфейс, что упрощает код и делает его более гибким. Например, метод «DrawObjects» в классе «Scene» может работать с любыми объектами, реализующими интерфейс «IGameObject».

Абстракция позволяет скрыть сложные детали реализации и предоставить простой интерфейс для взаимодействия. Например, класс «Time» предоставляет методы для работы с временем, скрывая сложные вычисления и системные вызовы. Это упрощает использование временных функций в других частях программы

Таким образом, использование ООП в данном проекте позволяет эффективно организовать игровые объекты, управлять сценами, обрабатывать ресурсы, звук, время и другие аспекты игры. Это делает код более структурированным, удобным для расширения и поддержки, что в конечном итоге улучшает качество и производительность игры.

Классы из папки «Engine» в данном проекте играют ключевую роль в обеспечении базовой функциональности движка игры (рис. 1).



```
class FileSystem
{
    Ссылок: 1
    public static Dictionary<string,Image> LoadFrames(string path)
    {
        Dictionary<string, Image> res = new Dictionary<string, Image>();
        StreamReader sr = new StreamReader(path);
        while(!sr.EndOfStream)
        {
            string[] lines = sr.ReadLine().Split('|');
            res.Add(lines[0], Image.FromFile(lines[1]));
        }
        sr.Close();
        return res;
    }
    Ссылок: 1
    public static Dictionary<string, SoundPlayer> LoadSound(string path)
    {
        Dictionary<string, SoundPlayer> res = new Dictionary<string, SoundPlayer>();
        StreamReader sr = new StreamReader(path);
        while (!sr.EndOfStream)
        {
            string[] lines = sr.ReadLine().Split('|');
            res.Add(lines[0], new SoundPlayer(lines[1]));
        }
        sr.Close();
        return res;
    }
}
class Resources
{
    static Dictionary<string, Image> frames = new Dictionary<string, Image>();
    static Dictionary<string, SoundPlayer> sounds = new Dictionary<string, SoundPlayer>();
    Ссылок: 1
    static public void InitializationResources()
    {
        frames = FileSystem.LoadFrames("Res.int");
        sounds = FileSystem.LoadSound("Sound.int");
    }
    Ссылок: 7
    static public Image GetFrame(string key) => frames[key];
    Ссылок: 1
    static public SoundPlayer GetSound(string key) => sounds[key];
}
```

```
class GameOver
{
    static public bool isGameOver = false;
    Ссылок: 1
    static public void DrawGameOverScreen(Graphics g)
    {
        g.DrawString($"GAME OVER \n Time: { Convert.ToInt32(Time.GetMinutes())}:" +
            $"{Time.GetSeconds()}",
            new Font("Stencil", 25),
            new SolidBrush(Color.Red), Render.Resolution.X / 2 - 100,
            Render.Resolution.Y / 2 - 100);
    }
}
class Render
{
    static Vector resolution;
    static IScene scene;
    Ссылок: 1
    public static void SetResolution(int x, int y) => resolution = new Vector(x, y);
    Ссылок: 1
    public static void SetScene(IScene customScene) => scene = customScene;
    Ссылок: 1
    public static Image DrawFrame()
    {
        Bitmap bitmap = new Bitmap(resolution.X, resolution.Y);
        Graphics g = Graphics.FromImage(bitmap);
        scene.DrawBack(g, resolution.X, resolution.Y);
        scene.DrawObjects(g);
        if (GameOver.isGameOver)
            GameOver.DrawGameOverScreen(g);
        return bitmap;
    }
    public static Vector Resolution => resolution;
}
class Time
{
    static int frames = 0;
    static int interval;

    static public void SetInterval(int value) => interval = value;

    static public void Frame_Tick(object sender, EventArgs e) => frames++;

    static public int GetMiliSeconds() => frames * interval;

    static public double GetSeconds() => GetMiliSeconds()/1000;

    static public double GetMinutes() => GetSeconds() / 60;

    static public int deltaTime => interval;
}
```

Рисунок 1. Классы группы «Engine»

Класс «FileSystem» отвечает за загрузку ресурсов, таких как изображения и звуки, из файлов. Он взаимодействует с классом «Resources», предоставляя ему методы для загрузки кадров и звуков. Это позволяет централизованно управлять ресурсами игры, что упрощает их использование в других частях программы.

«Render» отвечает за рендеринг сцены и управление разрешением экрана и взаимодействует с классом «Scene», который реализует интерфейс «IScene», для рисования фона и объектов сцены. «Render» также проверяет состояние игры через класс «GameOver» и, если игра завершена, рисует экран «GAME OVER». Данный класс обеспечивает централизованное управление графическим выводом, что упрощает работу с графикой в игре.

«Resources» управляет ресурсами игры, такими как изображения и звуки. Он взаимодействует с классом «FileSystem» для загрузки ресурсов и предоставляет методы для получения изображений и звуков по ключу. Описываемый класс обеспечивает централизованное управление ресурсами, что упрощает их использование в других частях программы.

«Sound» отвечает за воспроизведение звуков в игре. Он работает с классом «Resources» для получения звуков и предоставляет метод для их воспроизведения. Этот класс обеспечивает удобный интерфейс для работы с звуками, скрывая детали реализации и упрощая их использование в других частях программы.

«Time» управляет временем в игре. Предоставляет методы для установки интервала времени между кадрами, обновления счетчика кадров и получения текущего времени в миллисекундах, секундах и минутах. Этот класс обеспечивает централизованное управление временем, что упрощает работу с временными интервалами в игре.

«Transform» хранит информацию о позиции и размере объектов в игре, а также методы для установки, обновления позиции и размера объектов и для проверки столкновений. Класс обеспечивает базовую функциональность для работы с трансформациями объектов, что упрощает их управление и взаимодействие.

«Vector» представляет собой вектор в двумерном пространстве. Он осуществляет методы для инициализации векторов, получения координат и выполнения арифметических операций, таких как сложение и вычитание векторов. Всё это обеспечивает базовую функциональность для работы с векторами, упрощает управление позициями и размерами объектов в игре.

Классы из группы "Game" в данном проекте ответственны за специфическую логику и объекты (рис. 2). Эти классы определяют поведение игровых объектов, их взаимодействие и управление сценами, что делает их ключевыми компонентами игрового процесса. Первые четыре класса реализуют интерфейс «IGameObject», что обеспечивает единообразие в работе с различными игровыми объектами.

```
class Bomb : Transform, IGameObject
{
    int speed;
    int speedOfset;
    int curSpeed;
    Scene scene;
    Random random = new Random();

    public Bomb(int sx, int sy, int speed, int offsetSpeed, Scene scene)
    {
        this.scene = scene;
        SetSize(new Vector(sx, sy));
        this.speed = speed;
        this.speedOfset = offsetSpeed;
        NextPlane();
    }

    public void NextPlane()
    {
        curSpeed = speed + random.Next(-speedOfset, speedOfset);
        SetPosition(scene.planes[random.Next(0, scene.planes.Count)].Position);
    }

    public void Draw(Graphics g)
    {
        AddPosition(new Vector(0, curSpeed));
        g.DrawImage(Resources.GetFrame("Bomb"),
        Position.X, Position.Y, Size.X, Size.Y);
        if (Position.Y > Render.Resolution.Y)
            NextPlane();
    }

    public void Break()
    {
        scene.UseEffect(Position);
        NextPlane();
    }
}
```

```
class Plane : Transform, IGameObject
{
    int speed;
    Ссылок 1
    public Plane(int x, int y, int sx, int sy, int speed, int offsetSpeed)
    {
        Random random = new Random();
        SetPosition(new Vector(x, y + random.Next(-sy,sy)));
        SetSize(new Vector(sx, sy));
        this.speed = speed + random.Next(-offsetSpeed,offsetSpeed);
    }
    Ссылок 2
    public void Draw(Graphics g)
    {
        AddPosition(new Vector(speed,0));
        if (Position.X > Render.Resolution.X - Size.X)
            speed *= -1;
        if (Position.X < 0)
            speed *= -1;
        g.DrawImage(speed > 0? Resources.GetFrame("PlaneRight"):
            Resources.GetFrame("PlaneLeft"),
        Position.X, Position.Y, Size.X, Size.Y);
    }
}
```

```
class Scene : IScene
{
    Image backGround;
    int houseOffset = 100;
    int planeAddTime = 10;
    int planeTimer;
    int maxPlanes = 7;
    public List<House> houses = new List<House>();
    public List<Plane> planes = new List<Plane>();
    public List<Bomb> bombs = new List<Bomb>();
    public List<Effect> effects = new List<Effect>();
    Ссылок 1
    public Scene()
    {

        Resources.InitializationResources();
        backGround = Resources.GetFrame("Back");
        for(int i = 0; i< Render.Resolution.X / houseOffset; i++)
            houses.Add(new House(25+i* houseOffset, 280,70,70));
        Sound.Play("Siren");
    }
    Ссылок 2
    public void DrawBack(Graphics g, int x, int y) => g.DrawImage(backGround, 0, 0, x, y);
    Ссылок 2
    public void DrawObjects(Graphics g)
    {
        if((planeTimer -= Time.deltaTime) <= 0 && planes.Count < maxPlanes)
        {
            planeTimer = planeAddTime*1000;
            planes.Add(new Plane(0, 70, 70, 40, 7, 2));
            bombs.Add(new Bomb(40, 40, 7, 2, this));
        }
        List<IGameObject> objects = new List<IGameObject>();
        objects.AddRange(planes);
        objects.AddRange(houses);
        objects.AddRange(bombs);
        objects.AddRange(effects);
        foreach (var i in objects)
            i.Draw(g);
        CheckBreak();
    }
```

```
class Effect : Transform, IGameObject
{
    int countTiles = 5;
    int curTile = 0;
    Scene scene;
    Ссылок 1
    public Effect(Vector position,int sx, int sy,Scene scene)
    {
        Sound.Play("Exp");
        SetPosition(position);
        SetSize(new Vector(sx,sy));
        this.scene = scene;
    }
    Ссылок 2
    public void Draw(Graphics g)
    {
        if (curTile == countTiles)
            scene.effects.Remove(this);
        g.DrawImage(Resources.GetFrame("Exp"+curTile.ToString()),
        Position.X, Position.Y, Size.X, Size.Y);
        curTile++;
    }
}
```

Рисунок 2. Классы группы «Game»

Класс «Bomb» представляет собой бомбу, которая сбрасывается самолетами и падает вниз. Описываемый класс управляет своей позицией и скоростью, а также взаимодействует с классом «Scene» для определения

позиции самолетов и добавления новых бомб. Когда бомба достигает нижней границы экрана или сталкивается с домом, она разрушается и создает эффект взрыва. «Effect» представляет собой эффект взрыва, который возникает при разрушении бомбы. Представленный класс управляет анимацией взрыва, рисуя последовательные кадры и удаляя себя из сцены после завершения анимации. Данный функционал взаимодействует с классом «Scene» для добавления и удаления эффектов.

«House» представляет собой дом, который может быть разрушен бомбами. Он управляет своим состоянием (разрушен или нет) и рисует соответствующее изображение в зависимости от состояния. Когда бомба сталкивается с домом, то он разрушается, и это состояние используется для проверки условий завершения игры.

«Plane» представляет собой самолет, который летает по экрану и сбрасывает бомбы. Данный класс осуществляет управление своей позицией и скоростью, а также взаимодействует с классом «Scene» для добавления новых самолетов и бомб. Самолет меняет направление движения, когда достигает края экрана, и рисует соответствующее изображение в зависимости от направления движения.

«Scene» реализует интерфейс «IScene» и управляет всей сценой игры, включая фоновый рисунок, дома, самолеты, бомбы и эффекты. Этот класс инициализирует ресурсы, устанавливает фоновое изображение и добавляет дома в сцену. Также управляет таймером для добавления новых самолетов, бомб и рисует все объекты на экране. Параллельно всему этому проверяет столкновения бомб с домами и управляет состоянием игры, устанавливая флаг завершения игры, если все дома разрушены.

Взаимодействие между этими классами обеспечивает интеграцию различных аспектов игрового процесса, таких как управление объектами, их поведение и взаимодействие, а также управление сценами и состоянием игры.

После программирования движка и классов игры осуществляется тестирование и проверка готовой программы (рис. 3).

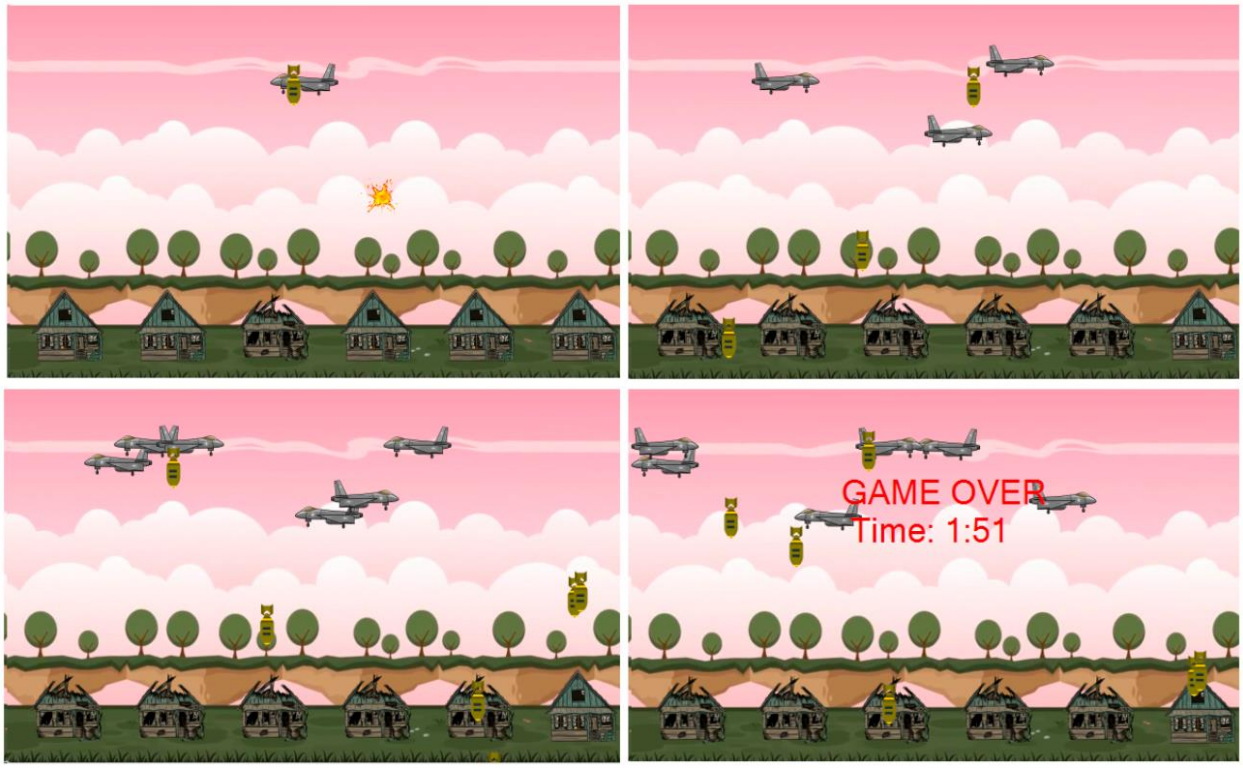


Рисунок 3. Результат работы программы

Выводы

Таким образом в ходе исследования было разработано графическое приложение, основанное на парадигме объектно-ориентированного программирования с помощью языка C#.

Библиографический список

1. Рыбалёв А.Н. Применение технологии ООП при разработке программ управления коммутационными аппаратами // Вестник Амурского государственного университета. Серия: Естественные и экономические науки. 2023. № 103. С. 49-54.
2. Палкина И.В., Набиуллин Р.Т., Симакина Н.И. Применение парадигм ООП в разработке клиент-серверного приложения // В сборнике: Наука и образование в обеспечении устойчивого развития человеческого потенциала в условиях перехода к цифровой экономике. Материалы X Российской с международным участием научно-практической конференции. Пермь, 2023. С. 332-338.
3. Денискин А. В. Многопоточность в языке программирования C# // Academy. 2017. № 2 (17). С. 21-24.
4. Ноек И. Д., Демичев В. А. Разработка компьютерной игры на языке программирования C# с применением основных принципов объектно-ориентированного программирования и свободного программного обеспечения Monogame // В сборнике: Материалы Ежегодной межвузовской студенческой научной конференции ОЧУ ВО "Еврейский университет". Москва, 2019. С. 251-259.

5. Яценко Е. Перспективы использования объектно-ориентированного языка программирования C# при разработке современных программных компонентов // Материалы XXXI Международной студенческой научно-практической конференции. Мозырь, 2024. С. 73.