МИНИСТРЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ

РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИИ

Отчет

О выполнении лабораторной работы № 3, 4

«Разработка программного обеспечения с использованием паттернов проектирования. Создание приложений с визуальным интерфейсом.»

По дисциплине «Программирование на языке С#»

Выполнил ст. гр. ІТШІ-19-2: Принял:

Меденицкий Алексей Бибичков И. Е.

2020

**Цель работы**

Изучение особенностей разработки программного обеспечения с использованием паттернов проектирования различных типов. Изучение особенностей создания приложений с графикой и анимацией в среде Visual Studio на языке программирования C#.

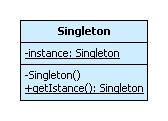
**Задание «Graph File Manager»**

Реализовать файловый менеджер отображающий файлы в виде графа. Для проектирования использовать паттерны Singleton, Composite, Command. В качестве графической библиотеки использовать monogame.

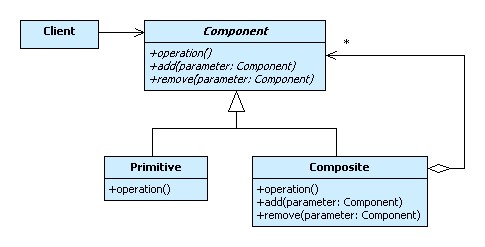
В файловом менеджере должна быть предусмотрена возможность вращения, перемещения и изменения размера отображаемого графа. Так же необходимо наличие подписей с названиями файлов и различными иконками исходя из типа обьекта (файл/папка) и в случае с файлом - его расширения.

**UML диаграммы использумых паттренов**

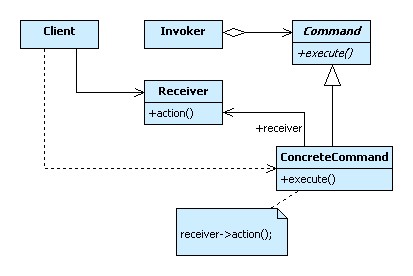
1. Singleton



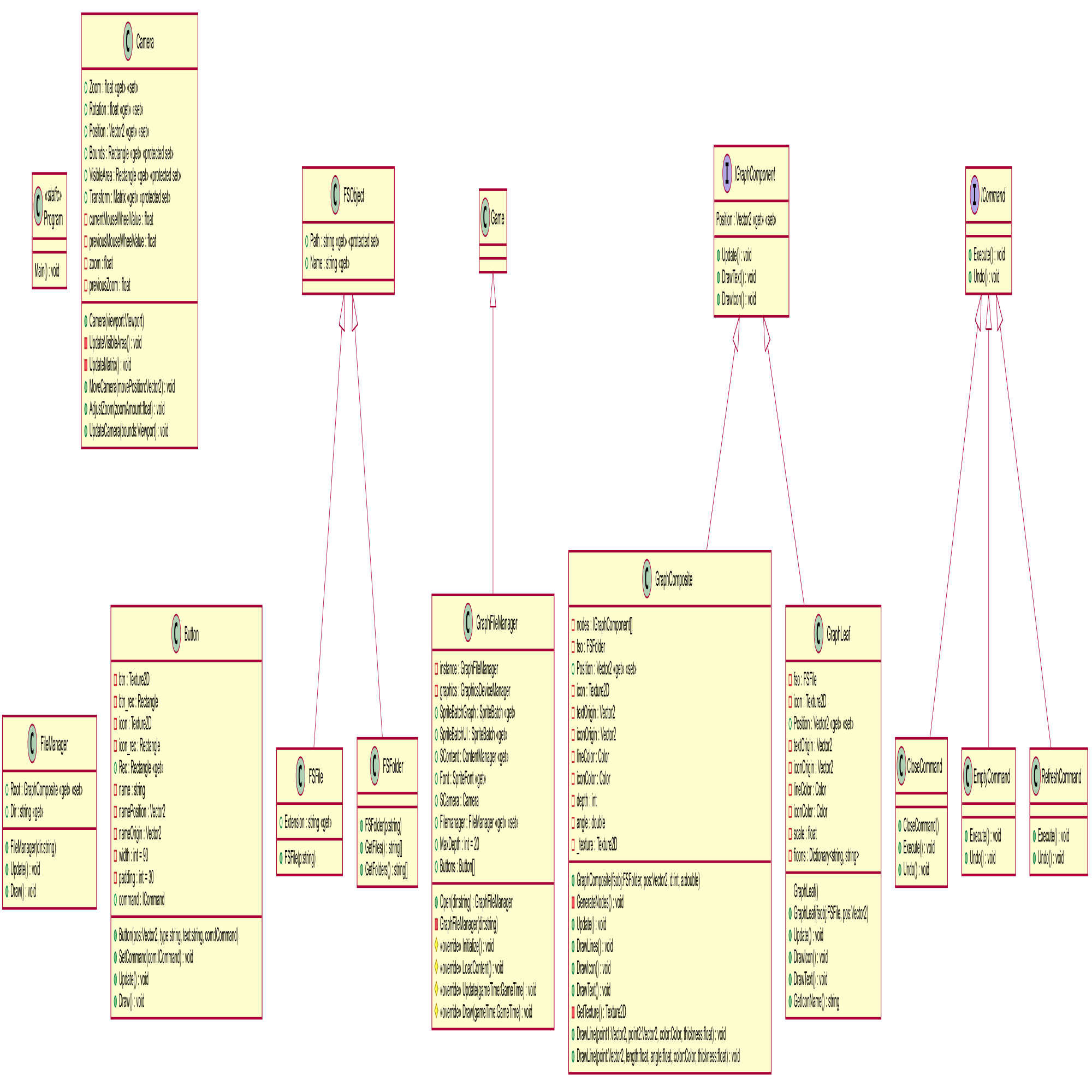
1. Composite

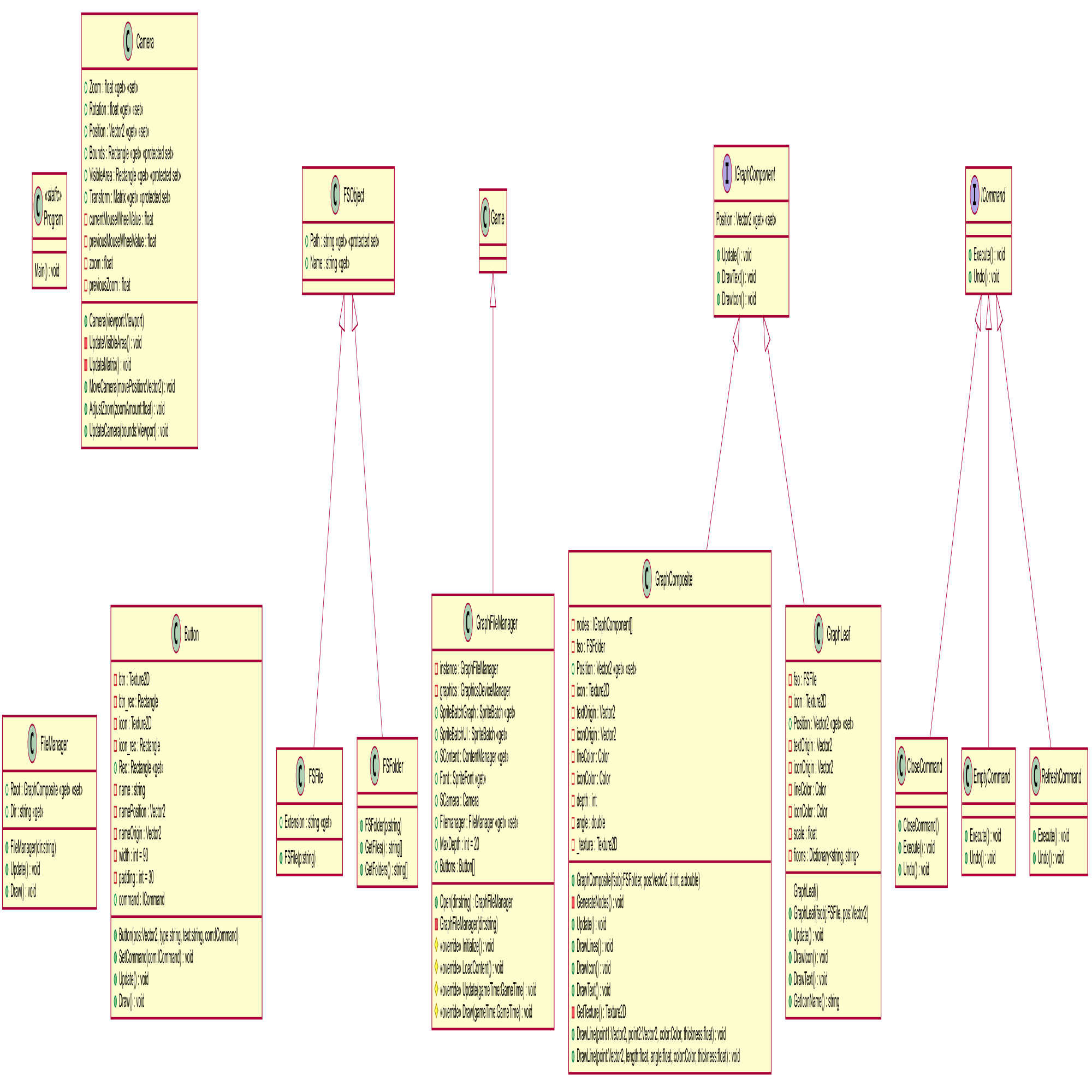


1. Command



**UML диаграмма разработанного приложения**





**Словесное описание особенностей отношений между реализованными в данной работе классами и интерфейсами.**

В основе программы лежит класс-Singleton GraphFileManager, наследуемый от класса Game (стандартного класса MonoGame). GraphFileManager прежде всего содержит экземпляр самого себя, экземпляр класса Camera, экземпляр класса FileManager и массив кнопок (класс Button). Данный класс выполняет необходимые первоначальные настройки Monogame не углубляясь в реаоизацию самого приложения. Для этого предназначен класс FileManager. Он определяет корневой каталог из которого будет расти граф и содержит ссылку типа IGraphComponent гарантировано хранящую экземпляр класса GraphComposite. Классы GraphComposite и GraphLeaf реализуют интерфейс IGraphComponent и реализуют паттерн Composite. GraphLeaf реализует лист дерева (графа) и отвечает лишь за самого себя. Содержит экземпляр класса FSFile, содержащий логику для работы с файлами. Класс GraphComposite отвечает за “разветвление” дерева и помимо экземпляра класса FSFolder для работы с файлами содержит также массив ссылок типа IGraphComponent, так как папка может хранить в себе как файлы так и другие папки. FSFile и FSFolder наследуются от класса FSObject, который содержит общие методы и поля для работы с обьектом файловой системы.

Паттерн Command реализует класс Button, который содержит ссылку типа ICommand, которая может хранить экземпляр класса какой либо команды. При нажатии на кнопку она вызывает метод Execute у присвоенной команды. Для примера было реализовано три простых класса команд - CloseCommand, EmptyCommand, RefreshCommand. Все эти классы реализуют интерфейс ICommand.

**Интерфейсы классов**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using Microsoft.Xna.Framework.Content;

using System;

using System.Collections.Generic;

namespace GraphFileManager

{

public static class Program

{

[STAThread]

static void Main();

}

public class GraphFileManager : Game

{

private static GraphFileManager instance;

private GraphicsDeviceManager graphics;

public static SpriteBatch SpriteBatchGraph { get; private set; }

public static SpriteBatch SpriteBatchUI { get; private set; }

public static ContentManager SContent {get; private set;}

public static SpriteFont Font {get; private set;}

public static Camera SCamera;

public FileManager Filemanager {get; set;}

public static int MaxDepth = 20;

public static Button[] Buttons;

public static GraphFileManager Open(string dir=".");

private GraphFileManager(string dir);

protected override void Initialize();

protected override void LoadContent();

protected override void Update(GameTime gameTime);

protected override void Draw(GameTime gameTime);

}

public class Camera

{

public float Zoom { get; set; }

public float Rotation { get; set; }

public Vector2 Position { get; set; }

public Rectangle Bounds { get; protected set; }

public Rectangle VisibleArea { get; protected set; }

public Matrix Transform { get; protected set; }

private float currentMouseWheelValue, previousMouseWheelValue, zoom, previousZoom;

public Camera(Viewport viewport);

private void UpdateVisibleArea();

private void UpdateMatrix();

public void MoveCamera(Vector2 movePosition);

public void AdjustZoom(float zoomAmount);

public void UpdateCamera(Viewport bounds);

}

public class Button

{

private Texture2D btn;

private Rectangle btn\_rec;

private Texture2D icon;

private Rectangle icon\_rec;

public Rectangle Rec { get{ return icon\_rec; } }

private string name;

private Vector2 namePosition;

private Vector2 nameOrigin;

private int width = 90;

private int padding = 30;

public ICommand command;

public Button(Vector2 pos, string type, string text, ICommand com=null);

public void SetCommand(ICommand com);

public void Update();

public void Draw();

}

public interface ICommand

{

public void Execute();

public void Undo();

}

public class CloseCommand : ICommand

{

public CloseCommand() {}

public void Execute();

public void Undo() { }

}

public class EmptyCommand : ICommand

{

public void Execute() { }

public void Undo() { }

}

public class RefreshCommand : ICommand

{

public void Execute();

public void Undo() { }

}

public class FileManager {

public GraphComposite Root {get; set;}

public string Dir {get; private set;}

public FileManager(string dir);

public void Update();

public void Draw()

}

public interface IGraphComponent

{

Vector2 Position {get; set;}

public void Update();

public void DrawText();

public void DrawIcon();

}

public class GraphComposite : IGraphComponent {

private IGraphComponent[] nodes;

private FSFolder fso;

public Vector2 Position {get; set;}

private Texture2D icon;

private Vector2 textOrigin;

private Vector2 iconOrigin;

private Color lineColor;

private Color iconColor;

private int depth;

private double angle;

public GraphComposite(FSFolder fsobj, Vector2 pos, int d, double a);

private void GenerateNodes();

public void Update();

public void DrawLines();

public void DrawIcon();

public void DrawText();

// Legacy code

private static Texture2D \_texture;

private static Texture2D GetTexture();

public static void DrawLine(Vector2 point1, Vector2 point2, Color color, float thickness = 1f);

public static void DrawLine(Vector2 point, float length, float angle, Color color, float thickness = 1f);

}

public class GraphLeaf : IGraphComponent {

private FSFile fso;

private Texture2D icon;

public Vector2 Position {get; set;}

private Vector2 textOrigin;

private Vector2 iconOrigin;

private Color lineColor;

private Color iconColor;

private float scale;

private static Dictionary<string, string> ficons;

static GraphLeaf();

public GraphLeaf(FSFile fsobj, Vector2 pos);

public void Update();

public void DrawIcon();

public void DrawText();

public string GetIconName()

}

public class FSObject

{

public string Path { get; protected set; }

public string Name {

get;

}

}

public class FSFolder : FSObject

{

public FSFolder(string p);

public string[] GetFiles();

public string[] GetFolders();

}

public class FSFile : FSObject

{

public string Extension {

get;

}

public FSFile(string p);

}

**Реализация классов**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using Microsoft.Xna.Framework.Content;

using System;

using System.Collections.Generic;

namespace GraphFileManager

{

public static class Program

{

[STAThread]

static void Main()

{

using (var fm = GraphFileManager.Open())

fm.Run();

}

}

public class GraphFileManager : Game

{

private static GraphFileManager instance;

private GraphicsDeviceManager graphics;

public static SpriteBatch SpriteBatchGraph { get; private set; }

public static SpriteBatch SpriteBatchUI { get; private set; }

public static ContentManager SContent {get; private set;}

public static SpriteFont Font {get; private set;}

public static Camera SCamera;

public FileManager Filemanager {get; set;}

public static int MaxDepth = 20;

public static Button[] Buttons;

public static GraphFileManager Open(string dir=".")

{

if (instance == null)

instance = new GraphFileManager(dir);

return instance;

}

private GraphFileManager(string dir)

{

SContent = Content;

SContent.RootDirectory = "Content";

graphics = new GraphicsDeviceManager(this);

graphics.IsFullScreen = true;

graphics.ApplyChanges();

IsMouseVisible = true;

Font = SContent.Load<SpriteFont>("MyFont");

Filemanager = new FileManager(dir);

}

protected override void Initialize()

{

Buttons = new Button[2];

Buttons[0] = new Button(new Vector2(0, 0), "close", "Close", new CloseCommand());

Buttons[1] = new Button(new Vector2(Buttons[0].Rec.X + Buttons[0].Rec.Width + 20, 0), "refresh", "Refresh", new RefreshCommand());

base.Initialize();

}

protected override void LoadContent()

{

SpriteBatchGraph = new SpriteBatch(GraphicsDevice);

SpriteBatchUI = new SpriteBatch(GraphicsDevice);

SCamera = new Camera(GraphicsDevice.Viewport);

// TODO: use this.Content to load your game content here

}

protected override void Update(GameTime gameTime)

{

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))

Exit();

SCamera.UpdateCamera(GraphicsDevice.Viewport);

Filemanager.Update();

foreach (var btn in Buttons)

btn.Update();

base.Update(gameTime);

}

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.Black);

SpriteBatchGraph.Begin(

SpriteSortMode.Deferred,

BlendState.AlphaBlend,

null, null, null, null,

SCamera.Transform);

Filemanager.Draw();

SpriteBatchGraph.End();

SpriteBatchUI.Begin();

foreach (var btn in Buttons)

btn.Draw();

SpriteBatchUI.End();

base.Draw(gameTime);

}

}

public class Camera

{

public float Zoom { get; set; }

public float Rotation { get; set; }

public Vector2 Position { get; set; }

public Rectangle Bounds { get; protected set; }

public Rectangle VisibleArea { get; protected set; }

public Matrix Transform { get; protected set; }

private float currentMouseWheelValue, previousMouseWheelValue, zoom, previousZoom;

public Camera(Viewport viewport)

{

Bounds = viewport.Bounds;

Zoom = 1f;

Rotation = 0f;

Position = Vector2.Zero;

}

private void UpdateVisibleArea()

{

var inverseViewMatrix = Matrix.Invert(Transform);

var tl = Vector2.Transform(Vector2.Zero, inverseViewMatrix);

var tr = Vector2.Transform(new Vector2(Bounds.X, 0), inverseViewMatrix);

var bl = Vector2.Transform(new Vector2(0, Bounds.Y), inverseViewMatrix);

var br = Vector2.Transform(new Vector2(Bounds.Width, Bounds.Height), inverseViewMatrix);

var min = new Vector2(

MathHelper.Min(tl.X, MathHelper.Min(tr.X, MathHelper.Min(bl.X, br.X))),

MathHelper.Min(tl.Y, MathHelper.Min(tr.Y, MathHelper.Min(bl.Y, br.Y))));

var max = new Vector2(

MathHelper.Max(tl.X, MathHelper.Max(tr.X, MathHelper.Max(bl.X, br.X))),

MathHelper.Max(tl.Y, MathHelper.Max(tr.Y, MathHelper.Max(bl.Y, br.Y))));

VisibleArea = new Rectangle((int)min.X, (int)min.Y, (int)(max.X - min.X), (int)(max.Y - min.Y));

}

private void UpdateMatrix()

{

Transform =

Matrix.CreateTranslation(new Vector3(-Position.X, -Position.Y, 0)) \*

Matrix.CreateRotationZ(Rotation) \*

Matrix.CreateScale(new Vector3(Zoom, Zoom, 0)) \*

Matrix.CreateTranslation(new Vector3(Bounds.Width \* 0.5f, Bounds.Height \* 0.5f, 0));

UpdateVisibleArea();

}

public void MoveCamera(Vector2 movePosition)

{

Vector2 newPosition = new Vector2(Position.X + (float)(movePosition.Y \* Math.Sin(Rotation)) + (float)(movePosition.X \* Math.Cos(Rotation)),

Position.Y + (float)(movePosition.Y \* Math.Cos(Rotation)) - (float)(movePosition.X \* Math.Sin(Rotation)));

Position = newPosition;

}

public void AdjustZoom(float zoomAmount)

{

Zoom += zoomAmount;

if (Zoom < 0.001f)

Zoom = 0.001f;

}

public void UpdateCamera(Viewport bounds)

{

Bounds = bounds.Bounds;

UpdateMatrix();

Vector2 cameraMovement = Vector2.Zero;

int moveSpeed;

if (Zoom > .8f) {

moveSpeed = 15;

} else if (Zoom < .8f && Zoom >= .6f) {

moveSpeed = 25;

} else if (Zoom < .6f && Zoom > .35f) {

moveSpeed = 25;

} else if (Zoom < .35f && Zoom > .20f) {

moveSpeed = 30;

} else if (Zoom <= .20f) {

moveSpeed = 50;

} else {

moveSpeed = 10;

}

if (Keyboard.GetState().IsKeyDown(Keys.W)) {

cameraMovement.Y = -moveSpeed;

}

if (Keyboard.GetState().IsKeyDown(Keys.S)) {

cameraMovement.Y = moveSpeed;

}

if (Keyboard.GetState().IsKeyDown(Keys.A)) {

cameraMovement.X = -moveSpeed;

}

if (Keyboard.GetState().IsKeyDown(Keys.D)) {

cameraMovement.X = moveSpeed;

}

if (Keyboard.GetState().IsKeyDown(Keys.E)) {

Rotation += 0.05f;

}

if (Keyboard.GetState().IsKeyDown(Keys.Q)) {

Rotation -= 0.05f;

}

previousMouseWheelValue = currentMouseWheelValue;

currentMouseWheelValue = Mouse.GetState().ScrollWheelValue;

if (currentMouseWheelValue > previousMouseWheelValue) {

AdjustZoom(.1f);

}

if (currentMouseWheelValue < previousMouseWheelValue) {

AdjustZoom(-.1f);

}

previousZoom = zoom;

zoom = Zoom;

MoveCamera(cameraMovement);

}

}

public class Button

{

private Texture2D btn;

private Rectangle btn\_rec;

private Texture2D icon;

private Rectangle icon\_rec;

public Rectangle Rec { get{ return icon\_rec; } }

private string name;

private Vector2 namePosition;

private Vector2 nameOrigin;

private int width = 90;

private int padding = 30;

public ICommand command;

public Button(Vector2 pos, string type, string text, ICommand com=null)

{

name = text;

btn = GraphFileManager.SContent.Load<Texture2D>("Textures/ButtonBG");

btn\_rec = new Rectangle((int)pos.X, (int)pos.Y, width, (int)((float)btn.Height/(float)btn.Width\*(float)width));

icon = GraphFileManager.SContent.Load<Texture2D>("Textures/"+type);

icon\_rec = new Rectangle((int)pos.X+padding/2, (int)pos.Y+padding/2, width-padding, (int)((float)icon.Height/(float)icon.Width\*(float)(width-padding)));

nameOrigin = new Vector2(GraphFileManager.Font.MeasureString(name).X / 2f, 0);

namePosition = new Vector2(pos.X+icon\_rec.Width/2f+padding/2, pos.Y+btn\_rec.Height-padding/4-GraphFileManager.Font.MeasureString(name).Y);

command = com != null ? com : new EmptyCommand();

}

public void SetCommand(ICommand com) {

command = com;

}

public void Update()

{

var mouseState = Mouse.GetState();

var mousePosition = new Point(mouseState.X, mouseState.Y);

if (btn\_rec.Contains(mousePosition)) {

if (mouseState.LeftButton == ButtonState.Pressed) {

command.Execute();

}

}

}

public void Draw()

{

GraphFileManager.SpriteBatchUI.Draw(btn, btn\_rec, Color.White);

GraphFileManager.SpriteBatchUI.Draw(icon, icon\_rec, Color.White);

GraphFileManager.SpriteBatchUI.DrawString(GraphFileManager.Font, name, namePosition, Color.Black,

0, nameOrigin, Vector2.One, SpriteEffects.None, 0);

}

}

public interface ICommand

{

public void Execute();

public void Undo();

}

public class CloseCommand : ICommand

{

public CloseCommand() {}

public void Execute()

{

GraphFileManager.Open().Exit();

}

public void Undo() { }

}

public class EmptyCommand : ICommand

{

public void Execute() { }

public void Undo() { }

}

public class RefreshCommand : ICommand

{

public void Execute() {

GraphFileManager.Open().Filemanager = new FileManager(GraphFileManager.Open().Filemanager.Dir);

}

public void Undo() { }

}

public class FileManager {

public GraphComposite Root {get; set;}

public string Dir {get; private set;}

public FileManager(string dir)

{

Dir = dir;

Root = new GraphComposite(new FSFolder(dir), new Vector2(0, 0), 0, 0);

}

public void Update()

{

Root.Update();

}

public void Draw()

{

Root.DrawLines();

Root.DrawIcon();

Root.DrawText();

}

}

public interface IGraphComponent

{

Vector2 Position {get; set;}

public void Update();

public void DrawText();

public void DrawIcon();

}

public class GraphComposite : IGraphComponent {

private IGraphComponent[] nodes;

private FSFolder fso;

public Vector2 Position {get; set;}

private Texture2D icon;

private Vector2 textOrigin;

private Vector2 iconOrigin;

private Color lineColor;

private Color iconColor;

private int depth;

private double angle;

public GraphComposite(FSFolder fsobj, Vector2 pos, int d, double a)

{

fso = fsobj;

Position = pos;

depth = d;

angle = a;

textOrigin = new Vector2(GraphFileManager.Font.MeasureString(fso.Name).X / 2f, 0);

lineColor = Color.Gray;

iconColor = depth == 0 ? Color.Blue : Color.White;

icon = GraphFileManager.SContent.Load<Texture2D>("Textures/folder");

textOrigin.Y = -icon.Height/2f\*0.08f;

iconOrigin = new Vector2(icon.Width \* 0.5f, icon.Height \* 0.5f);

if (depth < GraphFileManager.MaxDepth && fso != null)

GenerateNodes();

}

private void GenerateNodes()

{

string[] files = ((FSFolder)fso).GetFiles();

string[] folders = ((FSFolder)fso).GetFolders();

int N = files.Length + folders.Length;

int L = (GraphFileManager.MaxDepth - depth) \* 100;

double step = 2 \* Math.PI / (depth == 0 ? N : N + 1);

if (N == 0)

return;

nodes = new IGraphComponent[N];

int index = 0;

for (int i = 0; i < files.Length; ++i, ++index) {

Vector2 pos = (new Vector2((float)Math.Cos(step\*(index+1) + angle), (float)Math.Sin(step\*(index+1) + angle))) \* (files.Length \* 15 + 50) + Position;

nodes[index] = new GraphLeaf(new FSFile(files[i]), pos);

}

for (int i = 0; i < folders.Length; ++i, ++index) {

Vector2 pos = (new Vector2((float)Math.Cos(step\*(index+1) + angle), (float)Math.Sin(step\*(index+1) + angle))) \* L + Position;

nodes[index] = new GraphComposite(new FSFolder(folders[i]), pos,

depth+1, Math.Atan2(pos.Y-Position.Y, pos.X-Position.X) + Math.PI);

}

}

public void Update()

{

if (nodes != null)

foreach (var node in nodes)

node.Update();

}

public void DrawLines()

{

if (nodes != null)

foreach (var node in nodes) {

DrawLine(Position, node.Position, lineColor);

if (node is GraphComposite) {

((GraphComposite)node).DrawLines();

}

}

}

public void DrawIcon()

{

GraphFileManager.SpriteBatchGraph.Draw(icon, Position, null, iconColor, -GraphFileManager.SCamera.Rotation,

new Vector2(icon.Width \* 0.5f, icon.Height \* 0.5f), new Vector2(0.08f, 0.08f), SpriteEffects.None, 0f);

if (nodes != null)

foreach (var node in nodes)

node.DrawIcon();

}

public void DrawText()

{

GraphFileManager.SpriteBatchGraph.DrawString(GraphFileManager.Font, fso.Name, Position, Color.Red,

-GraphFileManager.SCamera.Rotation, textOrigin, Vector2.One, SpriteEffects.None, 0);

if (nodes != null)

foreach (var node in nodes)

node.DrawText();

}

// Legacy code

private static Texture2D \_texture;

private static Texture2D GetTexture()

{

SpriteBatch spriteBatch = GraphFileManager.SpriteBatchGraph;

if (\_texture == null)

{

\_texture = new Texture2D(spriteBatch.GraphicsDevice, 1, 1, false, SurfaceFormat.Color);

\_texture.SetData(new[] {Color.White});

}

return \_texture;

}

public static void DrawLine(Vector2 point1, Vector2 point2, Color color, float thickness = 1f)

{

SpriteBatch spriteBatch = GraphFileManager.SpriteBatchGraph;

var distance = Vector2.Distance(point1, point2);

var angle = (float)Math.Atan2(point2.Y - point1.Y, point2.X - point1.X);

DrawLine(point1, distance, angle, color, thickness);

}

public static void DrawLine(Vector2 point, float length, float angle, Color color, float thickness = 1f)

{

SpriteBatch spriteBatch = GraphFileManager.SpriteBatchGraph;

var origin = new Vector2(0f, 0.5f);

var scale = new Vector2(length, thickness);

spriteBatch.Draw(GetTexture(), point, null, color, angle, origin, scale, SpriteEffects.None, 0);

}

}

public class GraphLeaf : IGraphComponent {

private FSFile fso;

private Texture2D icon;

public Vector2 Position {get; set;}

private Vector2 textOrigin;

private Vector2 iconOrigin;

private Color lineColor;

private Color iconColor;

private float scale;

private static Dictionary<string, string> ficons;

static GraphLeaf()

{

ficons = new Dictionary<string, string>();

ficons.Add("word", "doc docx");

ficons.Add("csharp", "cs");

ficons.Add("picture", "bmp ico img jpg png");

}

public GraphLeaf(FSFile fsobj, Vector2 pos)

{

fso = fsobj;

Position = pos;

textOrigin = new Vector2(GraphFileManager.Font.MeasureString(fso.Name).X / 2f, 0);

lineColor = Color.Gray;

iconColor = Color.White;

icon = GraphFileManager.SContent.Load<Texture2D>("Textures/" + this.GetIconName());

scale = 26.44f / icon.Height;

textOrigin.Y = -icon.Height/2f\*scale;

iconOrigin = new Vector2(icon.Width \* 0.5f, icon.Height \* 0.5f);

}

public void Update()

{

}

public void DrawIcon()

{

GraphFileManager.SpriteBatchGraph.Draw(icon, Position, null, iconColor, -GraphFileManager.SCamera.Rotation,

new Vector2(icon.Width \* 0.5f, icon.Height \* 0.5f), new Vector2(scale, scale), SpriteEffects.None, 0f);

}

public void DrawText()

{

GraphFileManager.SpriteBatchGraph.DrawString(GraphFileManager.Font, fso.Name, Position, Color.Red,

-GraphFileManager.SCamera.Rotation, textOrigin, Vector2.One, SpriteEffects.None, 0);

}

public string GetIconName()

{

string ext = fso.Extension;

if (ext == "") return "file";

foreach (var kvp in ficons)

if (kvp.Value.Contains(ext))

return kvp.Key;

return "file";

}

}

public class FSObject

{

public string Path { get; protected set; }

public string Name {

get {

return System.IO.Path.GetFileName(Path);

}

}

}

public class FSFolder : FSObject

{

public FSFolder(string p)

{

Path = p;

}

public string[] GetFiles()

{

return Directory.GetFiles(Path);

}

public string[] GetFolders()

{

return Directory.GetDirectories(Path);

}

}

public class FSFile : FSObject

{

public string Extension {

get {

try {

return System.IO.Path.GetExtension(this.Path).Substring(1);

} catch(System.ArgumentOutOfRangeException) {

return System.IO.Path.GetExtension(this.Path);

}

}

}

public FSFile(string p)

{

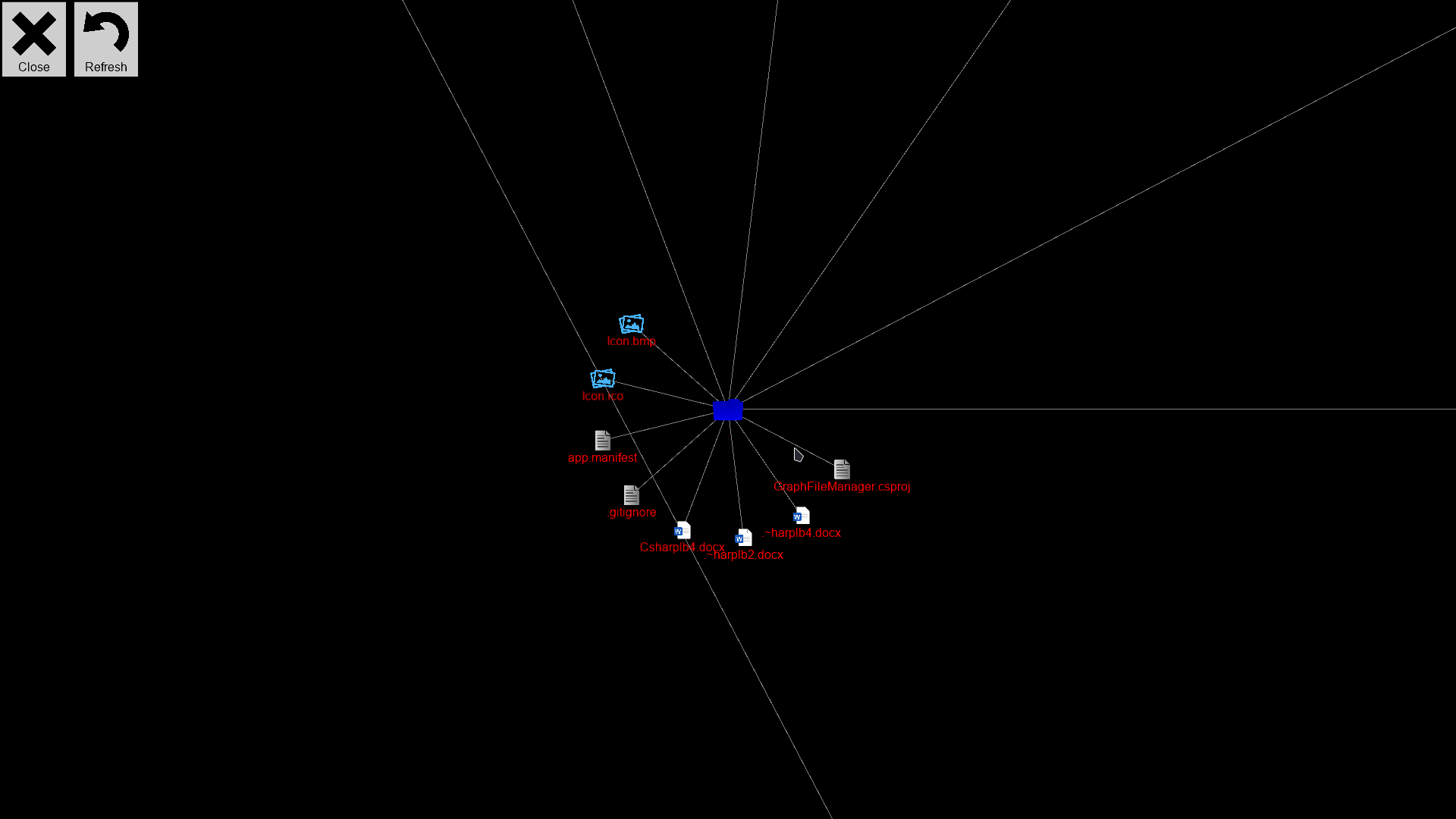
Path = p;

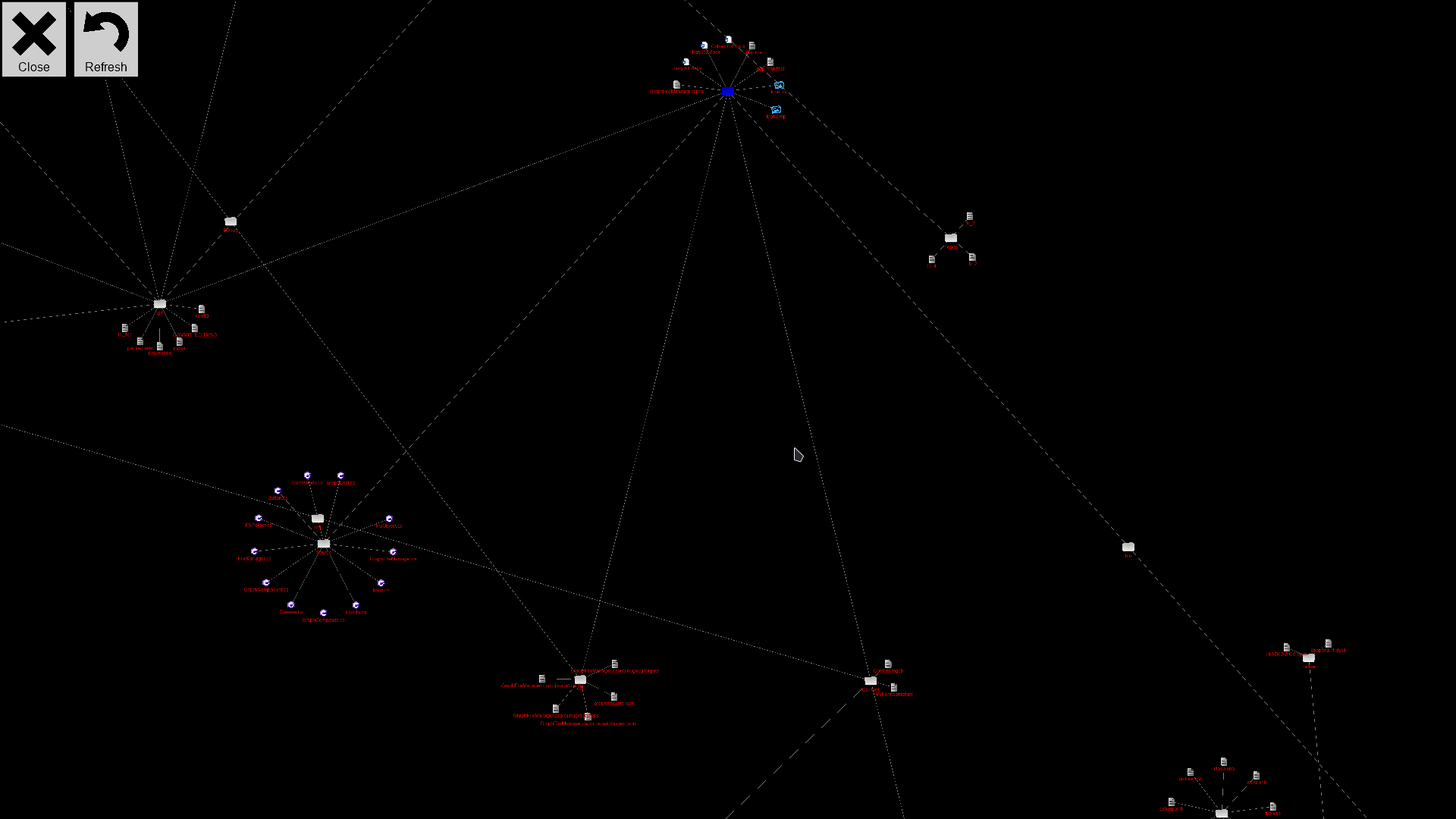
}

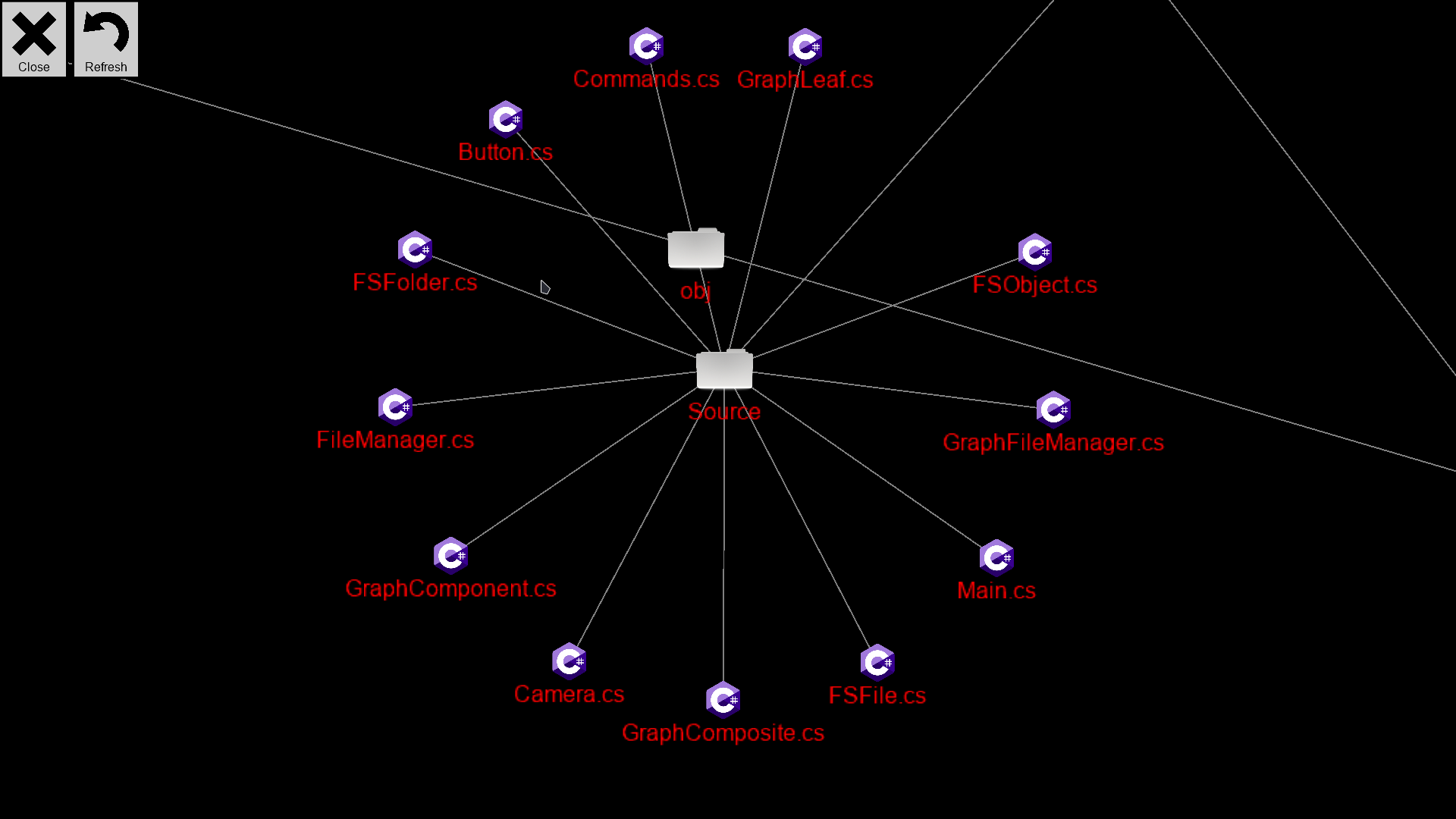
}

}

**Скрины результатов работы программы**

****

****

****

**Выводы по работе**

В результате выполнения лабораторной работы были изучены различные паттерны проектирования, в частности порождающий паттерн Singleton, структурный паттерн Composite и паттерн поведения Command. Также были изучены особенности создания графических приложений с использованием библиотеки monogame. Для закрепления знаний на практике, была написана программа, реализующая файловый менеджер с отображением элементов файловой системы в виде графа.