Střední škola informatiky, poštovnictví a finančnictví Brno, příspěvková organizace

Ročníková práce

Brno 2017

David Knieradl

Střední škola informatiky, poštovnictví a finančnictví Brno, příspěvková organizace

Brno 2017

David Knieradl

Střední škola informatiky, poštovnictví a finančnictví Brno, příspěvková organizace

Mario

Ročníková práce

Vedoucí práce:
Mgr. Skalka František

Vypracoval: David Knieradl
Studijní obor: IT
Číslo oboru: 18-20-M/01
Třída: IT3

Brno 2017

Střední škola informatiky, poštovnictví a finančnictví Brno, příspěvková organizace

Brno 2017

Prohlašuji, že jsem ročníkovou práci vypracoval samostatně s přispěním vedoucího práce a použil jsem jen literaturu a informační zdroje uvedené v kapitole literatura.

Okomentoval(a): [m1]: Pozor na mužský vs. ženský rod!!!

Děkuji Mgr. Františku Skalkovi za odborné vedení a cenné rady, které mi poskytl při zpracování této ročníkové práce.

Souhlasím s půjčováním a zpřístupněním ročníkové práce.

V Brně 14. května 2017 ....

Obsah

1 Úvod

Tato ročníková práce se zabývá tématem „Tabulky“. Jelikož je téma „Tabulky“, tak tento program samozřejmě využívá různé druhy polí. Nejčastěji se v něm setkáte s poli jednorozměrnými. Jedná se pouze o program ve velice rané fázi vývoje. Napsán byl formou OOP, neboli objektově orientovaným programováním. Ve stručnosti se jedná o takový druh programování, kdy vytváříme několik různých objektů, se kterými následně pracujeme. Podle názvu je poznat, že jsem přetvořil jednu z celosvětově známých her a to hru firmy Nintendo „Mario“. Obsahuje však pouze jeden svět, hlavní menu a stránku o programu. Největší část programu však tvoří jádro, které jsem během roku 2016/2017 stvořil. Lze jej nalézt na mém GitHubovém profilu (meowside) pod názvem DKEngine (www.github.com/meowside/DKEngine). Zde naleznete hodiny velice záživného čtení. Celá ročníková práce včetně jádra DKEngine čítá několik tisíc řádku. Při spuštění programu je potřeba jádro nejdříve inicializovat, až poté lze s ním pracovat podle potřeby. Pohyb postavy probíhá pomocí kláves „WASD“ a „Space“. Klávesa „W“ slouží pro skákání postavy. V momentě, kdy je nad postavou jiný objekt s komponentou „Collider“, tak se o tento objekt postava zastaví. Takto funguje kolize do strany i levé, pravé a spodní. V momentě střetu s nepřítelem se rozhoduje, do které strany byl hráč zasažen. Pokud se jednalo o stranu spodní, nepřítel byl poražen a přičte se skóre. V opačném případě přechází hráč do stavu o jedno nižší. Struktura takovýchto stavů je následovná.

Invincible -> Fire -> Super -> Small -> Dead

V tento moment je maximální funkčí stav „Super“. Do tohoto stavu se lze dostat pomocí objektu „PowerUp“, který vytvoří na mapě pohybující se houbu. Po získání této houby se přehraje animace zvětšení postavy a příčtou se body za získání houby.

Systém skóre je v tuto chvíli velice jednoduchý. Za přemožení nepřítele „Goomba“ lze získat bodů 100, za získání houby 200, za získání penízku 100. Momentálně ve hře nfunguje kombo systém, který by násobil skóre pomocí v sérii zabitých nepřátel.

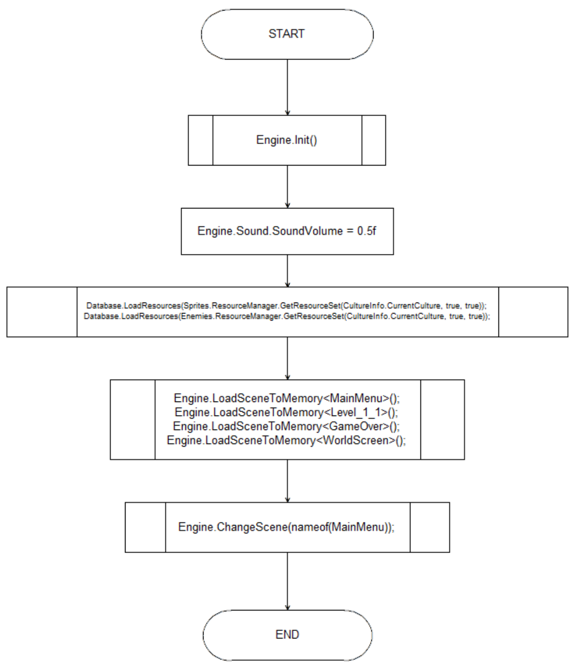
Program dále využívá knihovny NAudio. Jedná se o knihovnu zaměřenou na přehrávání audio záznamů. Pomocí této knihovny jsem vytvočil zvukový systém shopný přehrávat několik zvukových efektů zároveň v reálném čase.

Animace jsou následně prováděny pomocí visuálních prvků, kterými jsou obrázky ve formátu „PNG“ nebo „GIF“. K animování slouží objekt „Animator“ dostupný v jádře DKEngine.

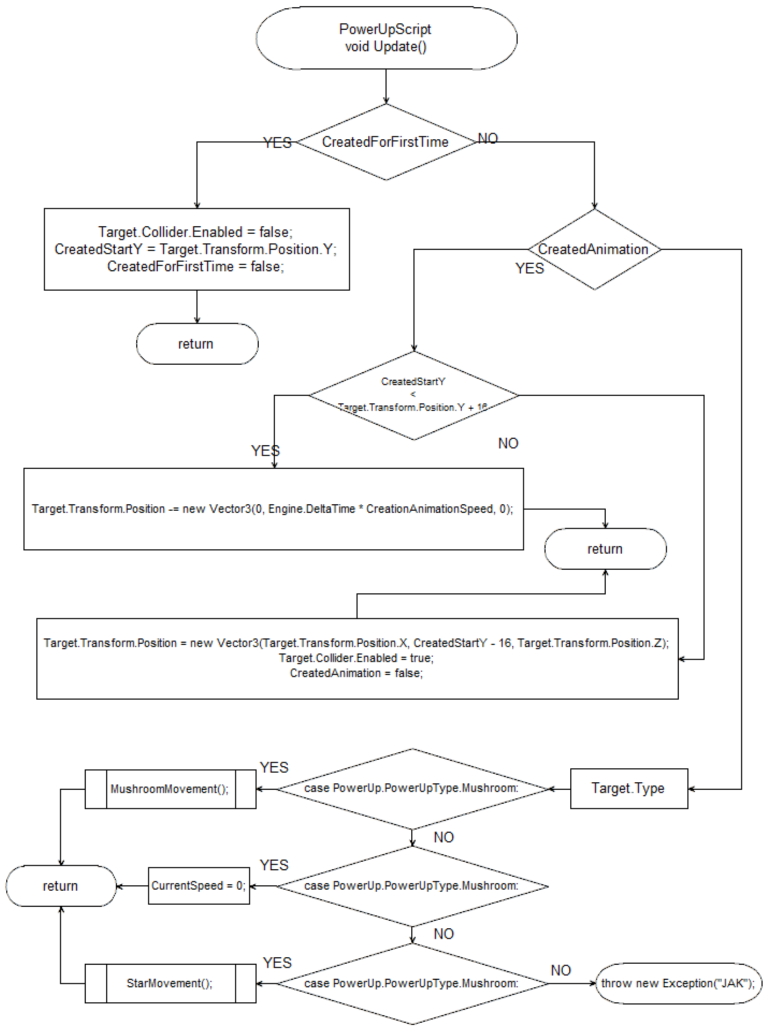
2 Stať

2.1 Vývojový diagram

2.1.1 Metoda „Main“


START
Engine.Init()
Engine.Sound.SoundVolume = 0.5f
Database.LoadResources(Sprites.ResourceManager.GetResourceSet(CultureInfo.CurrentCulture, true, true));
Database.LoadResources(Enemies.ResourceManager.GetResourceSet(CultureInfo.CurrentCulture, true, true));
Engine.LoadSceneToMemory<MainMenu>();
Engine.LoadSceneToMemory<Level_1_1>();
Engine.LoadSceneToMemory<GameOver>();
Engine.LoadSceneToMemory<WorldScreen>();
Engine.ChangeScene(nameof(MainMenu));
END

2.1.2 PowerUpScript – metoda Update


PowerUpScript
void Update()
CreatedForFirstTime
YES
NO
Target.Collider.Enabled = false;
CreatedStartY = Target.Transform.Position.Y;
CreatedForFirstTime = false;
return
CreatedAnimation
YES
CreatedStartY
<
Target.Transform.Position.Y + 16
YES
NO
Target.Transform.Position -= new Vector3(0, Engine.DeltaTime * CreationAnimationSpeed, 0);
return
Target.Transform.Position = new Vector3(Target.Transform.Position.X, CreatedStartY - 16, Target.Transform.Position.Z);
Target.Collider.Enabled = true;
CreatedAnimation = false;
Target.Type
case PowerUp.PowerUpType.Mushroom:
YES
MushroomMovement();
NO
case PowerUp.PowerUpType.Mushroom:
YES
CurrentSpeed = 0;
return
NO
case PowerUp.PowerUpType.Mushroom:
YES
StarMovement();
NO
throw new Exception("JAK");

2.2 Zdrojový kód

2.2.1 DKEngine

2.2.1.1 Engine.cs

```
/**
* (C) 2017 David Knieradl
*
* For the brave souls who get this far: You are the chosen ones,
* the valiant knights of programming who toil away, without rest,
* fixing our most awful code. To you, true saviors, kings of men,
* I say this: never gonna give you up, never gonna let you down,
* never gonna run around and desert you. Never gonna make you cry,
* never gonna say goodbye. Never gonna tell a lie and hurt you.
*/

using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.Ext;
using DKEngine.Core.UI;
using DKEngine.Data;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Threading;
using System.Threading.Tasks;

namespace DKEngine
{
    /// <summary>
    /// Engine class
    /// </summary>
    public static class Engine
    {
        /// <summary>
        /// Sound subclass
        /// </summary>
        public static class Sound
        {
            /// <summary>
            /// Enables sound
            /// </summary>
            public static bool IsSoundEnabled = true;

            /// <summary>
            /// Sets volume on sound inicialization
            /// </summary>
            public static float SoundVolume = 1f;

            internal readonly static SoundPlayer Instance = new SoundPlayer();
        }

        /// <summary>
        /// Render subclass
        /// </summary>
        public static class Render
        {
            /// <summary>
            /// Sets resolution scale in %
            /// </summary>
            public const int ResolutionScale = 50;

            /// <summary>
            /// The resolution ratio
            /// </summary>
            public const float ResolutionRatio = ResolutionScale / 100f;

            /// <summary>
            /// The rendered image width
            /// </summary>
            public const int RenderWidth = (int)(640 * ResolutionRatio);

            /// <summary>
            /// The rendered image height
            /// </summary>
            public const int RenderHeight = (int)(480 * ResolutionRatio);

            internal const int ImageBufferSize = 3 * RenderWidth * RenderHeight;
            internal const int ImageKeyBufferSize = RenderWidth * RenderHeight;

            internal static byte[] imageBuffer;
            internal static byte[] imageBufferKey;
            internal static byte[] ImageOutData;

            internal static bool AbortRender = false;

            internal const int Limiter = 1000;
        }

        /// <summary>
        /// Input subclass
        /// </summary>
        public static class Input
        {
            [DllImport("user32.dll")]
            private static extern ushort GetKeyState(short nVirtKey);

            private const ushort keyDownBit = 0x80;

            internal static bool[] KeysPressed;
            internal static bool[] KeysDown;
            internal static bool[] KeysReleased;
            internal static bool[] KeysUp;

            internal static short NumberOfKeys;

            /// <summary>
            /// Determines whether [is key pressed] [the specified key].
            /// </summary>
            /// <param name="key">The key</param>
            /// <returns>
            ///   <c>true</c> if [is key pressed] [the specified key]; otherwise, <c>false</c>.
            /// </returns>
            public static bool IsKeyPressed(ConsoleKey key)
            {
                return KeysPressed[(short)key];
            }

            /// <summary>
            /// Determines whether [is key down] [the specified key].
            /// </summary>
            /// <param name="key">The key</param>
            /// <returns>
            ///   <c>true</c> if [is key down] [the specified key]; otherwise, <c>false</c>.
            /// </returns>
            public static bool IsKeyDown(ConsoleKey key)
            {
                return KeysDown[(short)key];
            }

            /// <summary>
            /// Determines whether [is key up] [the specified key].
            /// </summary>
            /// <param name="key">The key</param>
            /// <returns>
            ///   <c>true</c> if [is key up] [the specified key]; otherwise, <c>false</c>.
            /// </returns>
            public static bool IsKeyUp(ConsoleKey key)
            {
                return KeysUp[(short)key];
            }

            /// <summary>
            /// Determines whether [is key released] [the specified key].
            /// </summary>
            /// <param name="key">The key</param>
            /// <returns>
            ///   <c>true</c> if [is key released] [the specified key]; otherwise, <c>false</c>.
            /// </returns>
            public static bool IsKeyReleased(ConsoleKey key)
            {
                return KeysReleased[(short)key];
            }

            internal static void CheckForKeys()
            {
                for (int key = 0; key < NumberOfKeys; key++)
                {
                    bool IsDown = ((GetKeyState((short)key) & keyDownBit) == keyDownBit);

                    if (IsDown)
                    {
                        if (!KeysDown[key])
                        {
                            KeysUp[key] = false;
                            KeysReleased[key] = false;
                            KeysPressed[key] = true;
                            KeysDown[key] = true;
                        }
                        else if (KeysPressed[key])
                        {
                            KeysPressed[key] = false;
                        }
                    }
                    else
                    {
                        if (KeysDown[key])
                        {
                            KeysPressed[key] = false;
                            KeysDown[key] = false;
                            KeysReleased[key] = true;
                            KeysUp[key] = true;
                        }
                        else if (KeysReleased[key])
                        {
                            KeysReleased[key] = false;
                        }
                    }
                }
            }
        }

        private static bool _IsInitialised = false;

        private static Thread BackgroundWorks;
        private static TextBlock FpsMeter;
        private static Stopwatch DeltaT;
        internal static Camera BaseCam;

        internal static Scene CurrentScene { get; set; }
        internal static Scene LoadingScene { get; set; }

        internal static Type LoadingSceneType { get; set; }

        internal static List<GameObject> RenderObjects;

        private static float deltaT = 0;
        public static float DeltaTime { get { return deltaT; } }

        private static TimeSpan lastUpdated = new TimeSpan();
        public static TimeSpan LastUpdated { get { return lastUpdated; } }

        public static string SceneName { get { return Engine.LoadingScene != null ? Engine.LoadingScene.Name : ""; } }

        private static readonly TimeSpan _firstTimeLoadDelay = new TimeSpan(0, 0, 1);
        private static TimeSpan FirstTimeLoadDelay = new TimeSpan();
        private static bool FirstTimeLoaded = true;

        internal static event EngineHandler UpdateEvent;

        internal delegate void EngineHandler();

        /// <summary>
        /// Sets engine to work.
        /// </summary>
```

```
        /// <exception cref="System.Exception">
        /// Engine initialisation failed\n" + e
        /// or
        /// Engine is being initialised second time
        /// </exception>
        public static void Init()
        {
            if (!_IsInitialised)
            {
                try
                {
                    WindowControl.WindowInit();
                    Database.InitDatabase();

                    Render.imageBuffer = new byte[Render.ImageBufferSize];
                    Render.imageBufferKey = new byte[Render.ImageKeyBufferSize];
                    Render.ImageOutData = new byte[Render.ImageBufferSize];

                    Input.NumberOfKeys = (short)Enum.GetNames(typeof(ConsoleKey)).Length;
                    Input.KeysPressed = new bool[Input.NumberOfKeys];
                    Input.KeysDown = new bool[Input.NumberOfKeys];
                    Input.KeysUp = new bool[Input.NumberOfKeys];
                    Input.KeysReleased = new bool[Input.NumberOfKeys];

                    DeltaT = Stopwatch.StartNew();

                    RenderObjects = new List<GameObject>(0xFFFF);

                    //Sound.OutputDevice = new WaveOut();

                    FpsMeter = new TextBlock();
                    FpsMeter.Transform.Position = new Vector3(4, -4, 128);
                    FpsMeter.Transform.Dimensions = new Vector3(50, 5, 1);
                    FpsMeter.VAlignment = Text.VerticalAlignment.Bottom;
                    FpsMeter.HAlignment = Text.HorizontalAlignment.Left;
                    FpsMeter.Text = "0";
                    FpsMeter.IsGUI = true;
                    FpsMeter.TextShadow = true;
                    FpsMeter.Foreground = Color.FromArgb(0xFF, 0x00, 0xFF, 0xFF);

                    FpsMeter.InitInternal();

                    UpdateEvent += FpsMeter.Scripts[0].UpdateHandle;

                    BackgroundWorks = new Thread(Update);
                    //RenderWorker    = new Thread(RenderImage);
                    BackgroundWorks.Start();
                    //RenderWorker.Start();

#if !DEBUG
                    SplashScreen();
#endif

                    _IsInitialised = true;
                }
                catch (Exception e)
                {
                    throw new Exception("Engine initialisation failed\n" + e);
                }
            }
            else
                throw new Exception("Engine is being initialised second time");
        }

        public static void LoadSceneToMemory<T>(object[] argsPreLoad = null, object[] argsPostLoad = null)
            where T : Scene
        {
            Engine.LoadingScene = (T)Activator.CreateInstance(typeof(T));
            Engine.LoadingScene.argsPreLoad = argsPreLoad;
            Engine.LoadingScene.argsPostLoad = argsPostLoad;
            Engine.LoadingScene.Set(argsPreLoad);
            Engine.LoadingScene.Init();

            Database.AddScene(Engine.LoadingScene);
        }

        /// <summary>
        /// Loads the scene to memory.
        /// </summary>
        /// <typeparam name="T">Scene</typeparam>
        /// <param name="argsPreLoad">The arguments pre load.</param>
        /// <param name="argsPostLoad">The arguments post load.</param>
        public static void LoadScene<T>(object[] argsPreLoad = null, object[] argsPostLoad = null) where T : Scene
        {
            LoadingSceneType = typeof(T);
            LoadScene(LoadingSceneType, argsPreLoad, argsPostLoad);
        }

        public static void LoadScene(Type scene, object[] argsPreLoad = null, object[] argsPostLoad = null)
        {
            if (!scene.IsSubclassOf(typeof(Scene)))
                throw new Exception($"Provided type {scene} is not subclass of Scene");

            Engine.LoadingScene = (Scene)Activator.CreateInstance(LoadingSceneType);

            Engine.LoadingScene.argsPreLoad = argsPreLoad;
            Engine.LoadingScene.argsPostLoad = argsPostLoad;

            Engine.LoadingScene.Set(argsPreLoad);
            Engine.LoadingScene.Init();

            UnregisterScene();
            RegisterScene(Engine.LoadingScene, argsPostLoad);
        }

        /// <summary>
        /// Reloads the scene.
        /// </summary>
        /// <param name="Name">The name of scene</param>
        public static void ReloadScene(string Name, object[] argsPreLoad = null)
        {
            Database.RewriteWorld(Name, argsPreLoad);
        }

        /// <summary>
        /// Changes the scene.
        /// </summary>
        /// <param name="Name">The name</param>
        /// <param name="Reload">if set to <c>true</c> [reload]</param>
        /// <param name="args">The arguments</param>
        public static void ChangeScene(string Name, bool Reload = false, object[] argsPreLoad = null, object[] argsPostLoad = null)
        {
            UnregisterScene();
            if (Reload)
            {
                ReloadScene(Name, argsPreLoad);

                if (argsPostLoad != null)
                    Database.GetScene(Name).argsPostLoad = argsPostLoad;
            }

            RegisterScene(Database.GetScene(Name), argsPostLoad);
        }

        private static void UnregisterScene()
        {
            try
            {
                Engine.CurrentScene.Unload();

                foreach (var item in CurrentScene.AllBehaviors)
                {
                    try
                    {
                        UpdateEvent -= item.UpdateHandle;
                    }
                    catch { }
                }

                while (CurrentScene.GameObjectsAddedToRender.Count > 0)
                {
                    GameObject tmp = CurrentScene.GameObjectsAddedToRender.Pop();
                    if (Engine.RenderObjects.Contains(tmp))
                        Engine.RenderObjects.Remove(tmp);

                    CurrentScene.GameObjectsToAddToRender.Push(tmp);
                }
            }
            catch { }
        }

        private static void RegisterScene(Scene source, object[] args)
        {
            Engine.LoadingScene = source;

            if (args != null)
            {
                source.argsPostLoad = args;
                source.Set(args);
            }
            else if (source.argsPostLoad != null)
                source.Set(source.argsPostLoad);

            foreach (var item in source.AllBehaviors)
            {
                try
                {
                    UpdateEvent += item.UpdateHandle;
                }
                catch { }
            }

            Engine.BaseCam = Engine.LoadingScene.BaseCamera;
            Engine.CurrentScene = source;
        }

        public static void ReloadCurrentScene()
        {
            UnregisterScene();
            LoadScene(LoadingSceneType);
        }

        private static void SplashScreen()
        {
            if (!_IsInitialised)
            {
                Engine.LoadScene<SplashScreenScene>();

                SpinWait.SpinUntil(() => ((SplashScreenScene)Engine.CurrentScene).Splash.Animator.NumberOfPlays >= 1);
            }
        }

        private static void Update()
        {
            Task imageRender = Task.Factory.StartNew(RenderImage);

            int NumberOfFrames = 0;
            TimeSpan timeOut = new TimeSpan(0, 0, 0, 0, 500);
            Stopwatch time = Stopwatch.StartNew();
            Stopwatch fpsLimiter = Stopwatch.StartNew();

            while (true)
            {
                Input.CheckForKeys();

                lastUpdated += DeltaT.Elapsed;
                deltaT = (float)DeltaT.Elapsed.TotalSeconds;
                DeltaT?.Restart();

                if (!FirstTimeLoaded)
                {
                    UpdateEvent?.Invoke();

                    while (Engine.CurrentScene?.NewlyGeneratedComponents.Count > 0)
                    {
                        Engine.CurrentScene.NewlyGeneratedComponents.Pop().InitInternal();
                    }

                    while (Engine.CurrentScene?.NewlyGeneratedBehaviors.Count > 0)
                    {
```

```
                        Behavior tmp = Engine.CurrentScene.NewlyGeneratedBehaviors.Pop();
                        UpdateEvent += tmp.UpdateHandle;
                        tmp.Start();
                    }

                    while (Engine.CurrentScene?.DestroyObjectAwaitList.Count > 0)
                    {
                        GameObject tmp = Engine.CurrentScene.DestroyObjectAwaitList[0];
                        Engine.CurrentScene.DestroyObjectAwaitList.RemoveAt(0);
                        tmp.Destroy();
                    }

                    while (Engine.CurrentScene?.GameObjectsToAddToRender.Count > 0)
                    {
                        GameObject tmp = Engine.CurrentScene.GameObjectsToAddToRender.Pop();
                        Engine.RenderObjects.Add(tmp);
                        Engine.CurrentScene.GameObjectsAddedToRender.Push(tmp);
                    }

                    List<GameObject> reference = Engine.RenderObjects.GetGameObjectsInView();

                    if (Engine.CurrentScene != null)
                    {
                        List<Collider> VisibleTriggers = Engine.CurrentScene?.AllGameObjectsColliders.Where(obj => obj.IsTrigger).ToList();
                        List<Collider> VisibleColliders = Engine.CurrentScene?.AllGameObjectsColliders.Where(obj => !obj.IsTrigger).ToList();
                        int ColliderCount = VisibleTriggers.Count;
                        for (int i = 0; i < ColliderCount; i++)
                            VisibleTriggers[i]?.TriggerCheck(VisibleColliders);
                    }

                    Engine.CurrentScene?.BaseCamera?.BufferImage(reference);

                    Buffer.BlockCopy(Render.imageBuffer, 0, Render.ImageOutData, 0, Render.ImageBufferSize);
                }
                else
                {
                    FirstTimeLoadDelay += new TimeSpan(0, 0, 0, 0, (int)(DeltaTime * 1000));

                    if (FirstTimeLoadDelay > _firstTimeLoadDelay)
                    {
                        FirstTimeLoaded = false;
                        FirstTimeLoadDelay = new TimeSpan();
                    }
                }

                NumberOfFrames++;

                Vsync(Render.Limiter, (int)fpsLimiter.ElapsedMilliseconds);
                fpsLimiter.Restart();

                if (time.ElapsedMilliseconds > timeOut.TotalMilliseconds)
                {
                    long t = NumberOfFrames * 1000 / time.ElapsedMilliseconds;
                    FpsMeter.Text = t.ToString();
                    /*#if DEBUG
                                Debug.WriteLine(t);
                    #endif*/
                    time.Restart();
                    NumberOfFrames = 0;
                }
            }
        }

        private static async void RenderImage()
        {
            IntPtr ConsoleWindow = GetConsoleWindow();

            using (Graphics g = Graphics.FromHwnd(ConsoleWindow))
            {
                g.CompositingQuality = System.Drawing.Drawing2D.CompositingQuality.HighSpeed;
                g.PixelOffsetMode = System.Drawing.Drawing2D.PixelOffsetMode.HighSpeed;
                g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.None;
                g.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.NearestNeighbor;

                Rectangle Screen = System.Windows.Forms.Screen.FromHandle(ConsoleWindow).Bounds;
                int Width = Screen.Width;
                int Height = Screen.Height;

                float ScaleRatio = Height / Engine.Render.RenderHeight;

                int RasteredHeight = (int)(Engine.Render.RenderHeight * ScaleRatio);
                int RasteredWidth = (int)(Engine.Render.RenderWidth * ScaleRatio);

                int XOffset = (int)(Width - RasteredWidth) / 2;
                int YOffset = (int)(Height - RasteredHeight) / 2;

                while (!Render.AbortRender)
                {
                    Rectangle ScreenResCheck = System.Windows.Forms.Screen.FromHandle(ConsoleWindow).Bounds;

                    if (ScreenResCheck != Screen)
                    {
                        Width = ScreenResCheck.Width;
                        Height = ScreenResCheck.Height;

                        XOffset = (int)(Width - (Engine.Render.RenderWidth * ScaleRatio)) / 2;
                        YOffset = (int)(Height - (Engine.Render.RenderHeight * ScaleRatio)) / 2;
                    }

                    unsafe
                    {
                        fixed (byte* ptr = Render.ImageOutData)
                        {
                            using (Bitmap outFrame = new Bitmap(Render.RenderWidth,
                                                                Render.RenderHeight,
                                                                3 * Render.RenderWidth,
                                                                System.Drawing.Imaging.PixelFormat.Format24bppRgb,
                                                                new IntPtr(ptr)))
                            {
                                Rectangle imageRect = new Rectangle(XOffset,
                                                                    YOffset,
                                                                    RasteredWidth,
                                                                    RasteredHeight);

                                g.DrawImage(outFrame, imageRect);
                            }
                        }
                    }

                    await Task.Delay(1);
                }
            }
        }

        private static void Vsync(int TargetFrameRate, int ImageRenderDelay)
        {
            int targetDelay = 1000 / TargetFrameRate;

            if (ImageRenderDelay < targetDelay)
            {
                Thread.Sleep(targetDelay - ImageRenderDelay);
            }
        }

        [DllImport("kernel32.dll", SetLastError = true)]
        private static extern IntPtr GetConsoleWindow();
    }
}
```

2.2.1.2 Core/Components/AnimationNode.cs

```
namespace DKEngine.Core.Components
{
    /// <summary>
    /// Node used in Animator Component
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Component" />
    public sealed class AnimationNode : Component
    {
        public Material Animation = null;
        public bool IsLoop = false;

        private AnimationNode()
            : base(null)
        { }

        public AnimationNode(string Name, Material Source)
            : base(null)
        {
            this.Name = Name;
            this.Animation = Source;
        }

        public override void Destroy()
        { }
    }
}
```

2.2.1.3 Core/Components/Animator.cs

```
/*
* (C) 2017 David Knieradl
*/

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Used for GameObject material animation
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Behavior" />
    /// <seealso cref="DKEngine.IAnimated" />
    public class Animator : Behavior, IAnimated
    {
        public TimeSpan CurrentAnimationTime;
        internal Dictionary<string, AnimationNode> Animations;
        private AnimationNode _current;

        public int NumberOfPlays { get; private set; } = 0;

        public AnimationNode Current
        {
            get { return _current; }
            set
            {
                if (value != _current)
                {
                    _current = value;
                    Parent.Model = _current.Animation;
                    NumberOfPlays = 0;
                    CurrentAnimationTime = new TimeSpan(0);
                }
            }
        }

        public int AnimationState
        {
            get
            {
                return (int)(CurrentAnimationTime.TotalMilliseconds / Parent.Model.DurationPerFrame % Parent.Model.Fra-
mes);
            }
        }

        public Animator(GameObject Parent)
            : base(Parent)
        {
            this.CurrentAnimationTime = new TimeSpan(0);
            this.Animations = new Dictionary<string, AnimationNode>();

            this.Name = string.Format("{0}_{1}", Parent.Name, nameof(Animator));
        }

        /// <summary>
        /// Adds the animation.
        /// </summary>
        /// <param name="Name">The animation node name.</param>
        /// <param name="Source">The source material for animation node.</param>
        public void AddAnimation(string Name, Material Source)
        {
            Animations.Add(Name, new AnimationNode(Name, Source));
            if (Animations.Count == 1)
            {
                Play(Animations.ElementAt(0).Key);
            }
        }

        /// <summary>
        /// Adds the animation.
        /// </summary>
        /// <param name="Name">The animation node name.</param>
        /// <param name="MaterialKey">The material key to search for material.</param>
        public void AddAnimation(string Name, string MaterialKey)
        {
            Animations.Add(Name, new AnimationNode(Name, Database.GetGameObjectMaterial(MaterialKey)));
            if (Animations.Count == 1)
            {
                Play(Animations.ElementAt(0).Key);
            }
        }

        /// <summary>
        /// Plays the specified animation name.
        /// </summary>
        /// <param name="AnimationName">Name of the animation.</param>
        public void Play(string AnimationName)
        {
            if (AnimationName != Current?.Name)
            {
                AnimationNode Result;

                try
                {
                    Result = Animations[AnimationName];
                }
                catch (Exception e)
                {
                    Debug.WriteLine("Animation \"{0}\"not found\n{1}", AnimationName, e);
                    return;
                }

                Current = Result;
            }
        }

        protected internal override void Update()
        {
            if (Parent?.Model?.Frames > 1)
            {
                CurrentAnimationTime = CurrentAnimationTime.Add(new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime * 1000)));

                if (CurrentAnimationTime.TotalMilliseconds > Parent.Model.Duration)
                {
                    CurrentAnimationTime = CurrentAnimationTime.Subtract(new TimeSpan(0, 0, 0, 0, Parent.Model.Duration));
                    NumberOfPlays++;
                }
            }
        }
```

1

```
        protected internal override void Start()
        { }

        public override void Destroy()
        {
            Engine.UpdateEvent -= UpdateHandle;

            Parent = null;
            UpdateHandle = null;
        }
    }
}
```

2.2.1.4 Core/Components/Behavior.cs

```
using System;
using System.Diagnostics;

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Base class for updated components
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Component" />
    public abstract class Behavior : Component
    {
        internal Engine.EngineHandler UpdateHandle;

        public Behavior(GameObject Parent)
            : base(Parent)
        {
            UpdateHandle = new Engine.EngineHandler(Update);
        }

        internal sealed override void Init()
        {
            try
            {
                Engine.LoadingScene.AllBehaviors.Add(this);
                Engine.LoadingScene.NewlyGeneratedBehaviors.Push(this);
            }
            catch (Exception e)
            {
                Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
            }
        }

        /// <summary>
        /// Called before parent object is rendered for first time
        /// </summary>
        protected internal abstract void Start();

        /// <summary>
        /// Updates each frame once
        /// </summary>
        protected internal abstract void Update();
    }
}
```

2.2.1.5 Core/Components/Camera.cs

```
/*
* (C) 2017 David Knieradl
*/

using DKEngine.Core.Ext;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Camera used for rendering
```

```
        /// </summary>
        /// <seealso cref="DKEngine.Core.Components.Component" />
        public sealed class Camera : Component
        {
            /// <summary>
            /// Sets the canvas background color
            /// </summary>
            public Color BackGround = Color.Black;

            /// <summary>
            /// The offset position of camera
            /// </summary>
            public Vector3 Position;

            internal float X { get { return RenderingGUI ? 0 : Parent != null ? Parent.Transform.Position.X + Position.X : Position.X; } }
            internal float Y { get { return RenderingGUI ? 0 : Parent != null ? Parent.Transform.Position.Y + Position.Y : Position.Y; } }

            private bool RenderingGUI = false;

            public Camera()
                : base(null)
            {
                this.Position = new Vector3(0, 0, 0);
                Engine.LoadingScene.BaseCamera = this;

                this.Name = string.Format("{0}", nameof(Camera));
            }

            public Camera(GameObject Parent)
                : base(Parent)
            {
                Engine.LoadingScene.BaseCamera = this;

                this.Name = string.Format("{0}_{1}", Parent.Name, nameof(Camera));
            }

            internal void BufferImage(List<GameObject> GameObjectsInView)
            {
                BackGroundInit();
                RenderingGUI = true;
                List<GameObject> GUI = GameObjectsInView.Where(item => item.IsGUI).ToList();
                int GUICount = GUI.Count;
                for (int i = 0; i < GUICount; i++)
                    GameObjectsInView.Remove(GUI[i]);

                while (GUICount > 0)
                {
                    float tempHeight = GUI.FindMaxZ();
                    GameObject[] toRender = GUI.Where(item => item.Transform.Position.Z == tempHeight).ToArray();

                    int toRenderCount = toRender.Length;
                    for (int i = toRenderCount - 1; i >= 0; i--)
                    {
                        toRender[i].Render();
                        GUI.Remove(toRender[i]);
                        GUICount--;
                    }
                }

                RenderingGUI = false;
                int TempCount = GameObjectsInView.Count;

                while (TempCount > 0)
                {
                    float tempHeight = GameObjectsInView.FindMaxZ();
                    GameObject[] toRender = GameObjectsInView.Where(item => item.Transform.Position.Z == tempHeight).ToArray();

                    int toRenderCount = toRender.Length;
                    for (int i = toRenderCount - 1; i >= 0; i--)
                    {
                        toRender[i].Render();
                        GameObjectsInView.Remove(toRender[i]);
                        TempCount--;
                    }
            }
        }

        private void BackGroundInit()
        {
            byte R = BackGround.R;
            byte G = BackGround.G;
            byte B = BackGround.B;

            int imageBufferLenght = Engine.Render.imageBuffer.Length;
            for (int i = 0; i < imageBufferLenght; i += 3)
            {
                Engine.Render.imageBuffer[i + 2] = R;
                Engine.Render.imageBuffer[i + 1] = G;
                Engine.Render.imageBuffer[i] = B;
            }

            Array.Clear(Engine.Render.imageBufferKey, 0, Engine.Render.imageBufferKey.Length);
        }

        public sealed override void Destroy()
        {
            if (Engine.BaseCam == this)
                Engine.BaseCam = null;

            try
            {
                Engine.LoadingScene.AllComponents.Remove(this.Name);
            }
            catch
            { }

            Parent = null;
        }
    }
}
```

2.2.1.6 Core/Components/Collider.cs

```
/*
* (C) 2017 David Knieradl
*/

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using static DKEngine.Core.Components.Transform;

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Collider used for GameObjects
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Component" />
    public class Collider : Component
    {
        internal event CollisionEnterHandler CollisionEvent;

        internal delegate void CollisionEnterHandler(Collider m);

        /// <summary>
        /// Determines size and position of collider
        /// </summary>
        public RectangleF Area = new RectangleF();

        /// <summary>
        /// If is TRUE => Triggers OnColliderEnter once another GameObject enter this collider
        /// </summary>
        public bool IsTrigger = false;

        /// <summary>
        /// Collider is enabled or disabled
        /// </summary>
        public bool Enabled = true;

        private float X { get { return Parent.Transform.Position.X + Area.X; } }
        private float Y { get { return Parent.Transform.Position.Y + Area.Y; } }
        private float Width { get { return Parent.Transform.Scale.X * Area.Width; } }
        private float Height { get { return Parent.Transform.Scale.Y * Area.Height; } }

        private bool _Right;
        private bool _Left;
        private bool _Top;
        private bool _Bottom;

        /// <summary>
        /// Creates new Instance of Collider class
        /// </summary>
        /// <param name="Parent">Parent of collider (determines size of collider)</param>
        internal Collider(GameObject Parent)
            : base(Parent)
        {
            this.Area = new RectangleF(0, 0, Parent.Transform.Dimensions.X, Parent.Transform.Dimensions.Y);

            this.Name = string.Format("{0}_{1}", Parent.Name, nameof(Collider));
        }

#if DEBUG

        /// <summary>
        /// Returns string containing <b>bool</b> value for each of the directions of this object.
        /// </summary>
        /// <returns></returns>
        public string DebugTestCollision()
        {
            return string.Format("Left {0}\nRight {1}\nTop {2}\nDown {3}", Collision(Direction.Left), Collision(Direction.Right),
Collision(Direction.Up), Collision(Direction.Down));
        }

#endif

        /// <summary>
        /// Collision check in specified direction.
        /// </summary>
        /// <param name="direction"></param>
        /// <returns></returns>
        public bool Collision(Direction direction)
        {
            if (this.IsTrigger || !this.Enabled)
                return false;

            if (LastUpdated != Engine.LastUpdated)
            {
                _Right = false;
                _Left = false;
                _Bottom = false;
                _Top = false;

                int count = Engine.CurrentScene.AllGameObjectsColliders.Count;
                for (int i = 0; i < count; i++)
                {
                    Collider tmp = Engine.CurrentScene.AllGameObjectsColliders[i];

                    bool _L = false;
                    bool _R = false;
                    bool _T = false;
                    bool _B = false;

                    float _LeftSpan = float.MaxValue;
                    float _RightSpan = float.MaxValue;
                    float _BottomSpan = float.MaxValue;
                    float _TopSpan = float.MaxValue;

                    if (_L = Left(tmp))
                    {
                        _LeftSpan = LeftSpan(tmp);
                    }

                    if (_R = Right(tmp))
                    {
                        _RightSpan = RightSpan(tmp);
                    }

            if (_T = Up(tmp))
            {
                _TopSpan = TopSpan(tmp);
            }

            if (_B = Down(tmp))
            {
                _BottomSpan = BottomSpan(tmp);
            }

            if (_T && _TopSpan <= _LeftSpan && _TopSpan <= _RightSpan && _TopSpan <= _BottomSpan)
            {
                _Top = true;
                this.Parent.Transform.Position += new Vector3(0, _TopSpan, 0);
                continue;
            }

            if (_B && _BottomSpan <= _LeftSpan && _BottomSpan <= _RightSpan && _BottomSpan <= _TopSpan)
            {
                _Bottom = true;
                this.Parent.Transform.Position += new Vector3(0, -_BottomSpan, 0);
                continue;
            }

            if (_L && _LeftSpan <= _BottomSpan && _LeftSpan <= _TopSpan && _LeftSpan <= _RightSpan)
            {
                _Left = true;
                this.Parent.Transform.Position += new Vector3(_LeftSpan, 0, 0);
                continue;
            }

            if (_R && _RightSpan <= _BottomSpan && _RightSpan <= _TopSpan && _RightSpan <= _LeftSpan)
            {
                _Right = true;
                this.Parent.Transform.Position += new Vector3(-_RightSpan, 0, 0);
                continue;
            }
        }
    }

    switch (direction)
    {
        case Direction.Up:
            return _Top;

        case Direction.Left:
            return _Left;

        case Direction.Down:
            return _Bottom;

        case Direction.Right:
            return _Right;

        default:
            return false;
    }
}

internal void TriggerCheck(List<Collider> VisibleObjects)
{
    if (!this.Enabled)
        return;

    int VisibleObjectsCount = VisibleObjects.Count;
    for (int i = 0; i < VisibleObjectsCount; i++)
    {
        Collider tmp = VisibleObjects[i];

        if (!tmp.Enabled)
            continue;

        if (Collided(tmp))
        {
            CollisionEvent?.Invoke(VisibleObjects[i]);
            continue;
        }
```

```
            }
        }

        private float LeftSpan(Collider obj)
        {
            return obj.X + obj.Width - this.X;
        }

        private float TopSpan(Collider obj)
        {
            return obj.Y + obj.Height - this.Y;
        }

        private float RightSpan(Collider obj)
        {
            return this.X + this.Width - obj.X;
        }

        private float BottomSpan(Collider obj)
        {
            return this.Y + this.Height - obj.Y;
        }

        private bool Left(Collider obj)
        {
            try
            {
                if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
                    return (this.Y < obj.Y + obj.Height && this.Y + this.Height > obj.Y && this.X >= obj.X + obj.Width / 2 && this.X <= obj.X + obj.Width); //(this.Y < obj.Y + obj.Width && this.Y + this.Width > obj.Y && this.X <= obj.X + obj.Width && this.X > obj.X);
            }
            catch { }

            return false;
        }

        private bool Right(Collider obj)
        {
            try
            {
                if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
                    return (this.Y < obj.Y + obj.Height && this.Y + this.Height > obj.Y && this.X + this.Width >= obj.X && this.X + this.Width <= obj.X + obj.Width / 2);//(this.Y < obj.Y + obj.Width && this.Y + this.Width > obj.Y && this.X + this.Width >= obj.X && this.X < X);
            }
            catch { }

            return false;
        }

        private bool Up(Collider obj)
        {
            try
            {
                if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
                    return (this.X < obj.X + obj.Width && this.X + this.Width > obj.X && this.Y <= obj.Y + obj.Height && this.Y >= obj.Y + obj.Height / 2);//(this.X < obj.X + obj.Width && this.X + this.Width > obj.X && this.Y <= obj.Y + obj.Width && this.Y > obj.Y);
            }
            catch { }

            return false;
        }

        private bool Down(Collider obj)
        {
            try
            {
                if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
                    return (this.X < obj.X + obj.Width && this.X + this.Width > obj.X && this.Y + this.Height >= obj.Y && this.Y + this.Height <= obj.Y + obj.Height / 2);//(this.X < obj.X + obj.Width && this.X + this.Width > obj.X && this.Y + this.Width >= obj.Y && this.Y < obj.Y);
            }
            catch { }

            return false;
        }

        private bool Collided(Collider obj)
        {
            try
            {
                if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
                    return (this.X < obj.X + obj.Width && this.X + this.Width > obj.X && this.Y < obj.Y + obj.Height && this.Y +
this.Height > obj.Y);
            }
            catch { }

            return false;
        }

        public override void Destroy()
        {
            try
            {
                Engine.CurrentScene.AllGameObjectsColliders.Remove(this);
            }
            catch { }

            try
            {
                Engine.CurrentScene.AllComponents.Remove(this.Name);
            }
            catch
            { }

            if (Parent.Collider == this)
                Parent.Collider = null;
        }

        public void SetCollisionManually(Direction direction)
        {
            switch (direction)
            {
                case Direction.Up:
                    _Top = true;
                    break;

                case Direction.Left:
                    _Left = true;
                    break;

                case Direction.Down:
                    _Bottom = true;
                    break;

                case Direction.Right:
                    _Right = true;
                    break;

                default:
                    break;
            }
        }

        internal sealed override void Init()
        {
            try
            {
                Engine.LoadingScene.AllGameObjectsColliders.Add(this);
            }
            catch (Exception e)
            {
                Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
            }
        }
    }
}
```

2.2.1.7 Core/Components/Component.cs

```
using DKEngine.Core.Ext;
using DKEngine.Core.UI;
using System;
using System.Diagnostics;

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Base class for all objects using DKEngine library
    /// </summary>
    public abstract class Component
    {
        private TimeSpan _lastUpdated;

        internal TimeSpan LastUpdated
        {
            get
            {
                TimeSpan tmp = _lastUpdated;
                _lastUpdated = Engine.LastUpdated;
                return tmp;
            }
        }

        /// <summary>
        /// The parent object of this instance
        /// </summary>
        public GameObject Parent = null;

        /// <summary>
        /// The name of this instance
        /// </summary>
        public string Name = "";

        internal Component(GameObject Parent)
        {
            this.Parent = Parent;
            _lastUpdated = Engine.LastUpdated;

            try
            {
                Engine.LoadingScene.NewlyGeneratedComponents.Push(this);
            }
            catch (Exception e)
            {
                Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
            }
        }

        internal void InitInternal()
        {
            Init();

            try
            {
                if (this.GetType() != typeof(Letter))
                {
                    Engine.LoadingScene.AllComponents.AddSafe(this);
                }
            }
            catch (Exception e)
            {
                Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
            }
        }

        internal virtual void Init()
        { }

        public abstract void Destroy();

        /// <summary>
        /// Finds the specified component of specified name.
        /// </summary>
        /// <typeparam name="T">Determines type of desired component</typeparam>
        /// <param name="Name">The name of desired component.</param>
        /// <returns></returns>
        public static T Find<T>(string Name) where T : Component
        {
            T retValue = null;

            try
            {
                retValue = (T)Engine.LoadingScene.AllComponents[Name];
            }
            catch (Exception ex)
            {
                Debug.WriteLine("Object not found\n" + ex);
            }

            return retValue;
        }
    }
}
```

2.2.1.8 Core/Components/Material.cs

```
/*
* (C) 2017 David Knieradl
*/

using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Low-Memory Material
    /// </summary>
    public sealed class Material
    {
        /// <summary>
        /// Source image used as Texture
        /// </summary>
        public readonly Bitmap Texture = null;

        /// <summary>
        /// Represents scaled length of image in pixels
        /// </summary>
        public readonly int Width = 0;

        /// <summary>
        /// Represents scaled height of image in pixels
        /// </summary>
        public readonly int Height = 0;

        /// <summary>
        /// Number of frames
        /// </summary>
        public readonly int Frames = 1;

        /// <summary>
        /// Total duration of animated image
        /// </summary>
        public readonly int Duration = 1;

        /// <summary>
        /// Duration between two frames of animation
        /// </summary>
        public readonly int DurationPerFrame = 1;

        /// <summary>
        /// Returns true if image is animated
        /// </summary>
        public readonly bool IsAnimated = false;

        /// <summary>
        /// Returns true if image is looped
        /// </summary>
        public readonly bool IsLooped = false;

        private int _SelectedLayer = -1;
        private FrameDimension _FrameDim = null;
        private BitmapData _BitmapData = null;
private byte[] _Data = null;
private byte _BytesPerPixel = 0;

/// <summary>
/// Loads image and creates new material
/// </summary>
/// <param name="source">Source image</param>
public Material(Image source)
{
    if (source != null)
    {
        _FrameDim = new FrameDimension(source.FrameDimensionsList[0]);
        Frames = source.GetFrameCount(_FrameDim);

        Texture = (Bitmap)source;

        Width = source.Width;
        Height = source.Height;

        if (ImageAnimator.CanAnimate(source))
        {
            int delay = 0;
            int this_delay = 0;
            int index = 0;

            for (int i = 0; i < Frames; i++)
            {
                this_delay = BitConverter.ToInt32(source.GetPropertyItem(20736).Value, index) * 10;
                delay += (this_delay < 1 ? 33 : this_delay);
                index += 4;
            }

            Duration = delay;
            DurationPerFrame = Duration / Frames;
            IsAnimated = true;
            IsLooped = BitConverter.ToInt16(source.GetPropertyItem(20737).Value, 0) != 1;
        }

        switch (source.PixelFormat)
        {
            case PixelFormat.Format32bppArgb:
                _BytesPerPixel = 4;
                break;

            case PixelFormat.Format24bppRgb:
                _BytesPerPixel = 3;
                break;

            default:
                throw new Exception("Unsupported");
        }

        _Data = new byte[Width * Height * _BytesPerPixel];
    }
}

/// <summary>
/// Creates new material with given color and scales it by parent's given scales
/// </summary>
/// <param name="clr">Source color</param>
/// <param name="Size">Vector3 used for material size</param>
public Material(Color clr, Vector3 Size)
{
    this.Width = (int)(Size.X < 1 ? 1 : Size.X);
    this.Height = (int)(Size.Y < 1 ? 1 : Size.Y);

    Frames = 1;

    _BytesPerPixel = 4;

    int size = Width * Height * _BytesPerPixel;
    _Data = new byte[size];

    for (int index = 0; index < size; index += _BytesPerPixel)
    {
        _Data[index + 3] = clr.A;
        _Data[index + 2] = clr.R;
```

```
                _Data[index + 1] = clr.G;
                _Data[index] = clr.B;
            }

            unsafe
            {
                fixed (byte* data = _Data)
                {
                    using (Bitmap tmp = new Bitmap(Width,
                                                   Height,
                                                   Width * _BytesPerPixel,
                                                   PixelFormat.Format32bppArgb,
                                                   new IntPtr(data)))
                    {
                        Texture = new Bitmap(tmp);
                    }
                }
            }

            _FrameDim = new FrameDimension(Texture.FrameDimensionsList[0]);
        }

        /// <summary>
        /// Creates new material with given color and scales it by parent's given scales
        /// </summary>
        /// <param name="clr">Source color</param>
        /// <param name="Parent">GameObject used for material size</param>
        public Material(Color clr, GameObject Parent)
            : this(clr, Parent.Transform.Dimensions)
        { }

        /// <summary>
        /// Render material into engine image buffer
        /// </summary>
        /// <param name="Parent">I3Dimensional for coordiantions</param>
        public void Render(GameObject Parent, Color? ReColor = null)
        {
            int AnimationState = Parent.Animator != null ? Parent.Animator.AnimationState : 0;
            bool HasShadow = Parent.HasShadow;

            if (_SelectedLayer != AnimationState)
            {
                if (_BitmapData != null)
                    Texture.UnlockBits(_BitmapData);
                Texture.SelectActiveFrame(_FrameDim, AnimationState);
                _BitmapData = Texture.LockBits(new Rectangle(0, 0, Width, Height), ImageLockMode.ReadOnly, Texture.PixelFormat);
                Marshal.Copy(_BitmapData.Scan0, _Data, 0, _Data.Length);
                _SelectedLayer = AnimationState;
            }

            float CamX = Engine.BaseCam != null ? Engine.BaseCam.X : 0;
            float CamY = Engine.BaseCam != null ? Engine.BaseCam.Y : 0;

            int x = (int)(Parent.Transform.Position.X - CamX);
            int y = (int)(Parent.Transform.Position.Y - CamY);

            float RasteredHeight = this.Height * Parent.Transform.Scale.Y;
            float RasteredWidth = this.Width * Parent.Transform.Scale.X;

            float NonRasteredWidthRatio = 1 / Parent.Transform.Scale.X;
            float NonRasteredHeightRatio = 1 / Parent.Transform.Scale.Y;

            float NonRasteredHeight = 0;
            float NonRasteredWidth = 0;
            if (ReColor == null)
            {
                for (int row = 0; row < RasteredHeight; row++)
                {
                    NonRasteredWidth = 0;

                    if (y + row >= Engine.Render.RenderHeight)
                        break;

                    for (int column = 0; column < RasteredWidth; column++)
                    {
                        if (x + column >= Engine.Render.RenderWidth)
```

```
                            break;

                        if (IsOnScreen(x + column, y + row))
                        {
                            int offset = (int)(3 * ((y + row) * Engine.Render.RenderWidth + (x + column)));
                            int keyOffset = (int)((y + row) * Engine.Render.RenderWidth + (x + column));

                            int tempColumn = (int)NonRasteredWidth;
                            int tempRow = (int)NonRasteredHeight;

                            int index = _BytesPerPixel * (tempRow * Width + tempColumn);

                            if (Engine.Render.imageBufferKey[keyOffset] != 255 && _BytesPerPixel == 4 ? _Data[index + 3] != 0 : true)
                            {
                                Color temp = MixPixel(Engine.Render.imageBufferKey[keyOffset], Engine.Render.imageBuffer[offset + 2], Engine.Render.imageBuffer[offset + 1], Engine.Render.imageBuffer[offset],
                                                        _BytesPerPixel == 4 ? _Data[index + 3] : (byte)255, _Data[index + 2], _Data[index + 1], _Data[index]);

                                Engine.Render.imageBufferKey[keyOffset] = temp.A;

                                Engine.Render.imageBuffer[offset] = temp.B;
                                Engine.Render.imageBuffer[offset + 1] = temp.G;
                                Engine.Render.imageBuffer[offset + 2] = temp.R;
                            }
                        }

                        NonRasteredWidth += NonRasteredWidthRatio;
                    }

                    NonRasteredHeight += NonRasteredHeightRatio;
                }
            }
            else
            {
                Color tempColor = (Color)ReColor;

                for (int row = 0; row < RasteredHeight; row++)
                {
                    NonRasteredWidth = 0;

                    if (y + row >= Engine.Render.RenderHeight)
                        break;

                    for (int column = 0; column < RasteredWidth; column++)
                    {
                        if (x + column >= Engine.Render.RenderWidth)
                            break;

                        if (IsOnScreen(x + column, y + row))
                        {
                            int offset = (int)(3 * ((y + row) * Engine.Render.RenderWidth + (x + column)));
                            int keyOffset = (int)((y + row) * Engine.Render.RenderWidth + (x + column));

                            int tempColumn = (int)NonRasteredWidth;
                            int tempRow = (int)NonRasteredHeight;

                            int index = _BytesPerPixel * (tempRow * Width + tempColumn);

                            if (Engine.Render.imageBufferKey[keyOffset] != 255 && _BytesPerPixel == 4 ? _Data[index + 3] != 0 : true)
                            {
                                Color temp = MixPixel(Color.FromArgb(Engine.Render.imageBufferKey[keyOffset], Engine.Render.imageBuffer[offset + 2], Engine.Render.imageBuffer[offset + 1], Engine.Render.imageBuffer[offset]),
                                                        tempColor);

                                Engine.Render.imageBufferKey[keyOffset] = temp.A;

                                Engine.Render.imageBuffer[offset] = temp.B;
                                Engine.Render.imageBuffer[offset + 1] = temp.G;
                                Engine.Render.imageBuffer[offset + 2] = temp.R;
                            }
                        }

                        NonRasteredWidth += NonRasteredWidthRatio;
                    }

                    NonRasteredHeight += NonRasteredHeightRatio;
                }
            }

            if (HasShadow)
            {
                NonRasteredHeight = 0;
                NonRasteredWidth = 0;

                x++;
                y++;

                for (int row = 0; row < RasteredHeight; row++)
                {
                    NonRasteredWidth = 0;

                    if (y + row >= Engine.Render.RenderHeight)
                        break;

                    for (int column = 0; column < RasteredWidth; column++)
                    {
                        if (x + column >= Engine.Render.RenderWidth)
                            break;

                        if (IsOnScreen(x + column, y + row))
                        {
                            int offset = (int)(3 * ((y + row) * Engine.Render.RenderWidth + (x + column)));
                            int keyOffset = (int)((y + row) * Engine.Render.RenderWidth + (x + column));

                            int tempColumn = (int)NonRasteredWidth;
                            int tempRow = (int)NonRasteredHeight;

                            int index = _BytesPerPixel * (tempRow * Width + tempColumn);

                            if (Engine.Render.imageBufferKey[keyOffset] != 255 && _BytesPerPixel == 4 ? _Data[index + 3] != 0 : true)
                            {
                                Color temp = MixPixel(Engine.Render.imageBufferKey[keyOffset], Engine.Render.imageBuffer[offset + 2], Engine.Render.imageBuffer[offset + 1], Engine.Render.imageBuffer[offset],
                                                      (byte)192, (byte)0, (byte)0, (byte)0);

                                Engine.Render.imageBufferKey[keyOffset] = temp.A;

                                Engine.Render.imageBuffer[offset] = temp.B;
                                Engine.Render.imageBuffer[offset + 1] = temp.G;
                                Engine.Render.imageBuffer[offset + 2] = temp.R;
                            }
                        }

                        NonRasteredWidth += NonRasteredWidthRatio;
                    }

                    NonRasteredHeight += NonRasteredHeightRatio;
                }
            }
        }

        public Color MixPixel(byte topA, byte topR, byte topG, byte topB, byte bottomA, byte bottomR, byte bottomG, byte bottomB)
        {
            if (topA == 0)
                return Color.FromArgb(bottomA, bottomR, bottomG, bottomB);

            if (bottomA == 0)
                return Color.FromArgb(topA, topR, topG, topB);

            float opacityTop = (float)topA / 255;

            byte newA = (byte)(topA + bottomA >= 255 ? 255 : topA + bottomA);
            byte A = (byte)(newA - topA);

            float opacityBottom = (float)A / 255;

            byte R = (byte)(topR * opacityTop + bottomR * opacityBottom);
            byte G = (byte)(topG * opacityTop + bottomG * opacityBottom);
            byte B = (byte)(topB * opacityTop + bottomB * opacityBottom);

            return Color.FromArgb(newA, R, G, B);
        }

        public Color MixPixel(Color top, Color bottom)
        {
            if (top.A == 0)
                return bottom;

            if (bottom.A == 0)
                return top;

            float opacityTop = (float)top.A / 255;

            byte newA = (byte)(top.A + bottom.A >= 255 ? 255 : top.A + bottom.A);
            byte A = (byte)(newA - top.A);

            float opacityBottom = (float)A / 255;

            byte R = (byte)(top.R * opacityTop + bottom.R * opacityBottom);
            byte G = (byte)(top.G * opacityTop + bottom.G * opacityBottom);
            byte B = (byte)(top.B * opacityTop + bottom.B * opacityBottom);

            return Color.FromArgb(newA, R, G, B);
        }

        private bool IsOnScreen(float x, float y)
        {
            return x >= 0 && x < Engine.Render.RenderWidth && y >= 0 && y < Engine.Render.RenderHeight;
        }
    }
}
```

2.2.1.9 Core/Components/Parabola.cs

```
using System;

namespace DKEngine.Core.Components
{
    /*
     * ------------------
     * DOES NOT WORK YET
     * ------------------
    */

    [Obsolete]
    public class Parabola : Behavior
    {
        public TimeSpan Time;
        public float Y;

        private float _accumulated = 0f;
        public float Accumulated { get { return _accumulated; } }

        public bool Enabled = false;

        private float Elapsed = 0f;

        private float[] ValuesInTime;
        private int NumberOfSamples;
        private const float SamplesInSecodnd = 1000;
        private const float X1 = 0f;
        public float X2 { get; private set; }

        public Parabola(GameObject Parent)
            : base(Parent)
        {
            Name = string.Format("{0}_{1}", Parent.Name, nameof(Parabola));
        }

        public override void Destroy()
        { }

        protected internal override void Start()
        {
            NumberOfSamples = (int)Time.TotalMilliseconds;
            ValuesInTime = new float[NumberOfSamples];
            float Duration = (float)Time.TotalSeconds;

            float lastResult = 0f;

            for (float i = 0; i < NumberOfSamples; i += 0.1f)
            {
                float constant = i / 1000f;
                float result = ((float)Math.Pow(-constant, 2) - (Duration * constant)) * Y;
                ValuesInTime[(int)i] = result - lastResult;
                lastResult = result;
            }
        }

        protected internal override void Update()
        {
            if (Enabled)
            {
                _accumulated = 0;
                float MaxTime = 0;
                float LeftoverTime = (float)((Elapsed + Engine.DeltaTime) * 1000 - Time.TotalMilliseconds);

                if (LeftoverTime > 0)
                {
                    MaxTime = (float)Time.TotalSeconds;
                    Enabled = false;
                }
                else
                {
                    MaxTime = Elapsed + Engine.DeltaTime;
                }

                int start = (int)(Elapsed * 1000);
                int end = (int)(MaxTime * 1000);
                for (int i = start; i < end; i++)
                {
                    _accumulated += ValuesInTime[i];
                }

                Elapsed += Engine.DeltaTime;
            }
        }
    }
}
```

2.2.1.10 Core/Components/SoundSource.cs

```
using NAudio.Wave;
using NAudio.Wave.SampleProviders;
using System;
using System.Collections.Generic;
using System.Linq;

namespace DKEngine.Core.Components
{
    internal class SoundPlayer
    {
        private readonly DirectSoundOut outputDevice;
        private readonly MixingSampleProvider mixer;
        private bool IsAvailable = true;

        internal SoundPlayer(int sampleRate = 44100, int channelCount = 2)
        {
            outputDevice = new DirectSoundOut(40);
            mixer = new MixingSampleProvider(WaveFormat.CreateIeeeFloatWaveFormat(sampleRate, channelCount))
            {
                ReadFully = true
            };
            outputDevice.Init(mixer);
            outputDevice.Play();
        }

        private ISampleProvider ConvertToRightChannelCount(CachedSoundSampleProvider input)
        {
            if (input.WaveFormat.Channels == mixer.WaveFormat.Channels)
            {
                input.cachedSound._SampleProvider = input;
                return input.cachedSound._SampleProvider;
            }
            if (input.WaveFormat.Channels == 1 && mixer.WaveFormat.Channels == 2)
            {
                input.cachedSound._SampleProvider = new MonoToStereoSampleProvider(input);
                return input.cachedSound._SampleProvider;
            }
            throw new NotImplementedException("Not yet implemented this channel count conversion");
        }

        public void PlaySound(Sound sound)
        {
            if (IsAvailable)
            {
                try
                {
                    sound._SampleProvider = ConvertToRightChannelCount(new CachedSoundSampleProvider(sound));
                    AddMixerInput(sound._SampleProvider);
                }
                catch
                {
                    IsAvailable = false;
                }
            }
        }

        public void StopSound(Sound sound)
        {
            if (IsAvailable)
            {
                try
                {
                    RemoveMixerInput(sound._SampleProvider);
                }
                catch
                { }
            }
        }

        private void AddMixerInput(ISampleProvider input)
        {
            mixer.AddMixerInput(input);
        }

        private void RemoveMixerInput(ISampleProvider input)
        {
            mixer.RemoveMixerInput(input);
        }

        public void Dispose()
        {
            outputDevice.Dispose();
        }
    }

    /// <summary>
    /// SoundSource component used for sound effects
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Component" />
    public class SoundSource : Component
    {
        private bool IsAvailable = true;

        public SoundSource(GameObject Parent)
            : base(Parent)
        {
            this.Name = string.Format("{0}_{1}", Parent.Name, nameof(SoundSource));
        }

        public void PlaySound(Sound sound)
        {
            if (Engine.Sound.IsSoundEnabled)
            {
                if (IsAvailable)
                {
                    try
                    {
                        Engine.Sound.Instance.PlaySound(sound);
                    }
                    catch
                    {
                        IsAvailable = false;
                    }
                }
            }
        }

        public void StopSound(Sound sound)
        {
            if (Engine.Sound.IsSoundEnabled)
            {
                if (IsAvailable)
                {
                    try
                    {
                        Engine.Sound.Instance.StopSound(sound);
                    }
                    catch
                    { }
                }
            }
        }

        public override void Destroy()
        {
            try
            {
                Engine.LoadingScene.AllComponents.Remove(this.Name);
            }
            catch
            { }

            this.Parent = null;
        }
    }

    internal class CachedSoundSampleProvider : ISampleProvider
    {
        public Sound cachedSound;
        private long position;

        public CachedSoundSampleProvider(Sound cachedSound)
        {
            this.cachedSound = cachedSound;
        }

        public int Read(float[] buffer, int offset, int count)
        {
            var availableSamples = cachedSound.AudioData.Length - position;
            var samplesToCopy = Math.Min(availableSamples, count);
            Array.Copy(cachedSound.AudioData, position, buffer, offset, samplesToCopy);
            position += samplesToCopy;

            return (int)samplesToCopy;
        }

        public WaveFormat WaveFormat { get { return cachedSound.WaveFormat; } }
    }

    /// <summary>
    /// Class holding specified audio file
    /// </summary>
    public class Sound
    {
        public float[] AudioData { get; private set; }
        public WaveFormat WaveFormat { get; private set; }
        public AudioFileReader FileReader { get; private set; }
        internal ISampleProvider _SampleProvider { get; set; }

        public Sound(string audioFileName)
        {
            using (FileReader = new AudioFileReader(audioFileName))
            {
                FileReader.Volume = Engine.Sound.SoundVolume;
                // TODO: could add resampling in here if required
```

```
                WaveFormat = FileReader.WaveFormat;
                var wholeFile = new List<float>((int)(FileReader.Length / 4));
                var readBuffer = new float[FileReader.WaveFormat.SampleRate * FileReader.WaveFormat.Channels];
                int samplesRead;
                while ((samplesRead = FileReader.Read(readBuffer, 0, readBuffer.Length)) > 0)
                {
                    wholeFile.AddRange(readBuffer.Take(samplesRead));
                }
                AudioData = wholeFile.ToArray();
            }
        }
    }
}
```

2.2.1.11 Core/Components/Transform.cs

```
/*
* (C) 2017 David Knieradl
*/

namespace DKEngine.Core.Components
{
    /// <summary>
    /// Transformation class holding information about position, scale and sizes of GameObject
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Component" />
    public sealed class Transform : Component
    {
        private Vector3 _Dimensions;
        private Vector3 _Position;
        private Vector3 _Scale;

        public Vector3 Dimensions
        {
            get { return _Dimensions; }
            set
            {
                Vector3 tmp = value - _Dimensions;
                _Dimensions = value;
                _ScaledDimensions = _Dimensions * _Scale;

                int childCount = Parent.Child.Count;
                for (int i = 0; i < childCount; i++)
                    Parent.Child[i].Transform.Dimensions += tmp;
            }
        }

        public Vector3 Position
        {
            get { return _Position; }
            set
            {
                Vector3 tmp = value - _Position;
                _Position = value;

                int childCount = Parent.Child.Count;
                for (int i = 0; i < childCount; i++)
                    Parent.Child[i].Transform.Position += tmp;
            }
        }

        public Vector3 Scale
        {
            get { return _Scale; }
            set
            {
                Vector3 tmp = value / _Scale;
                _Scale = value;
                _ScaledDimensions = _Dimensions * _Scale;

                int childCount = Parent.Child.Count;
                for (int i = 0; i < childCount; i++)
                    Parent.Child[i].Transform.Scale *= tmp;
            }
        }

        internal Vector3 _ScaledDimensions;
        
        public Transform(GameObject Parent)
            : base(Parent)
        {
            _Position = new Vector3();
            _Dimensions = new Vector3();
            _Scale = new Vector3();
            _ScaledDimensions = new Vector3();
        }

        public override void Destroy()
        {
            try
            {
                Engine.LoadingScene.AllComponents.Remove(this.Name);
            }
            catch { }
        }

        /// <summary>
        /// Possible directions
        /// </summary>
        public enum Direction
        {
            Up,
            Left,
            Down,
            Right
        }
    }
}
```

2.2.1.12 Core/Components/Vector3.cs

```
namespace DKEngine.Core.Components
{
#pragma warning disable CS0660 // Type defines operator == or operator != but does not override Object.Equals(object o)
#pragma warning disable CS0661 // Type defines operator == or operator != but does not override Object.GetHashCode()

    /// <summary>
    /// Three-dimensional vector
    /// </summary>
    public struct Vector3
#pragma warning restore CS0661 // Type defines operator == or operator != but does not override Object.GetHashCode()
#pragma warning restore CS0660 // Type defines operator == or operator != but does not override Object.Equals(object o)
    {
        /// <summary>
        /// The X vector
        /// </summary>
        public float X;

        /// <summary>
        /// The Y vector
        /// </summary>
        public float Y;

        /// <summary>
        /// The Z vector
        /// </summary>
        public float Z;

        public Vector3(float X, float Y, float Z)
        {
            this.X = X;
            this.Y = Y;
            this.Z = Z;
        }

        public static Vector3 operator -(Vector3 left, Vector3 right)
        {
            return new Vector3(left.X - right.X, left.Y - right.Y, left.Z - right.Z);
        }

```

```
public static Vector3 operator -(Vector3 left, float right)
{
    return new Vector3(left.X - right, left.Y - right, left.Z - right);
}

public static Vector3 operator +(Vector3 left, Vector3 right)
{
    return new Vector3(left.X + right.X, left.Y + right.Y, left.Z + right.Z);
}

public static Vector3 operator +(Vector3 left, float right)
{
    return new Vector3(left.X + right, left.Y + right, left.Z + right);
}

public static Vector3 operator *(Vector3 left, Vector3 right)
{
    return new Vector3(left.X * right.X, left.Y * right.Y, left.Z * right.Z);
}

public static Vector3 operator *(Vector3 left, float right)
{
    return new Vector3(left.X * right, left.Y * right, left.Z * right);
}

public static Vector3 operator /(Vector3 left, Vector3 right)
{
    return new Vector3(left.X / right.X, left.Y / right.Y, left.Z / right.Z);
}

public static Vector3 operator /(Vector3 left, float right)
{
    return new Vector3(left.X / right, left.Y / right, left.Z / right);
}

public static bool operator ==(Vector3 left, Vector3 right)
{
    return left.X == right.X && left.Y == right.Y && left.Z == right.Z;
}

public static bool operator !=(Vector3 left, Vector3 right)
{
    return left.X != right.X || left.Y != right.Y || left.Z != right.Z;
}

public Vector3 Add(Vector3 Value)
{
    return this + Value;
}

public Vector3 Add(float X, float Y, float Z)
{
    return this + new Vector3(X, Y, Z);
}

public Vector3 Add(float Value)
{
    return this + Value;
}

public Vector3 Decrease(Vector3 Value)
{
    return this - Value;
}

public Vector3 Decrease(float X, float Y, float Z)
{
    return this - new Vector3(X, Y, Z);
}

public Vector3 Decrease(float Value)
{
    return this - Value;
}

public Vector3 Multiply(Vector3 Value)
{
```

```
            return this * Value;
        }

        public Vector3 Multiply(float X, float Y, float Z)
        {
            return this * new Vector3(X, Y, Z);
        }

        public Vector3 Multiply(float Value)
        {
            return this * Value;
        }

        public Vector3 Divide(Vector3 Value)
        {
            return this / Value;
        }

        public Vector3 Divide(float X, float Y, float Z)
        {
            return this / new Vector3(X, Y, Z);
        }

        public Vector3 Divide(float Value)
        {
            return this / Value;
        }

        private static Vector3 _zero = new Vector3(0, 0, 0);

        public static Vector3 Zero
        {
            get { return _zero; }
        }
    }
}
```

2.2.1.13 Core/Scripts/TextControlScript.cs

```
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using System.Collections.Generic;
using static DKEngine.Core.UI.Text;

namespace DKEngine.Core.Scripts
{
    internal sealed class TextControlScript : Script
    {
        internal TextBlock _Parent { get { return (TextBlock)Parent; } }

        public TextControlScript(TextBlock Parent)
            : base(Parent)
        { }

        protected internal override void Start()
        {
            if (_Parent.Text.Length > 0)
            {
                Text();
            }
        }

        protected internal override void Update()
        {
            if (_Parent != null)
            {
                if (_Parent._changed)
                {
                    Text();
                }
            }
        }

        private void Text()
        {
            int _textCount = _Parent._text.Count;
            for (int i = _textCount - 1; i >= 0; i--)
            _Parent._text[i].Destroy();

        List<Letter> retValue = new List<Letter>();
        List<List<Letter>> textAligned = new List<List<Letter>>() { new List<Letter>() };

        float Xoffset = 0;
        float Yoffset = 0;
        int rows = 0;

        if (_Parent.Transform.Dimensions.X > 0)
        {
            for (int i = 0; i < _Parent._textStr.Length; i++)
            {
                if (_Parent._textStr[i] == ' ')
                {
                    Xoffset += 3 * _Parent.Transform.Scale.X * _Parent.FontSize;
                }
                else
                {
                    if (_Parent._textStr[i] == '\r' || _Parent._textStr[i] == '\n')
                    {
                        Xoffset = 0;
                        Yoffset += 6 * _Parent.Transform.Scale.Y * _Parent.FontSize;
                        rows++;

                        textAligned.Add(new List<Letter>());

                        continue;
                    }

                    Material newLetterMaterial = Database.GetLetter(_Parent._textStr[i]);

                    if (Xoffset + newLetterMaterial.Width * _Parent.FontSize > _Parent.Transform.Dimensions.X)
                    {
                        Xoffset = 0;
                        Yoffset += 6 * _Parent.Transform.Scale.Y * _Parent.FontSize;
                        rows++;

                        textAligned.Add(new List<Letter>());
                    }

                    Letter l = new Letter(_Parent);

                    l.Transform.Position += new Vector3(Xoffset, Yoffset, _Parent.Transform.Position.Z);
                    l.Foreground = _Parent.Foreground;
                    l.Model = newLetterMaterial;
                    l.Transform.Scale *= _Parent.FontSize;
                    l.Name = _Parent._textStr[i].ToString();
                    l.HasShadow = _Parent.TextShadow;
                    textAligned[rows].Add(l);

                    Xoffset += (l.Transform.Dimensions.X + 1) * l.Transform.Scale.X;
                }
            }
        }

        int textAlignedCount = textAligned.Count;
        float maxHeight = textAlignedCount * 6 * _Parent.FontSize * _Parent.Transform.Scale.Y;
        float startY = 0;

        switch (_Parent._TVA)
        {
            case VerticalAlignment.Top:
                startY = 0;
                break;

            case VerticalAlignment.Center:
                startY = (_Parent.Transform.Dimensions.Y * _Parent.Transform.Scale.Y * _Parent.FontSize - maxHeight) /
2;
                break;

            case VerticalAlignment.Bottom:
                startY = _Parent.Transform.Dimensions.Y * _Parent.Transform.Scale.Y * _Parent.FontSize - maxHeight;
                break;

            default:
                break;
            }

            for (int i = 0; i < textAlignedCount; i++)
            {
                float maxWidth = 0;
                int textAlignedRowCount = textAligned[i].Count;

                if (textAlignedRowCount > 0)
                    maxWidth = textAligned[i][textAlignedRowCount - 1].Model.Width * _Parent.Transform.Scale.X *
_Parent.FontSize + textAligned[i][textAlignedRowCount - 1].Transform.Position.X - textAligned[i][0].Transform.Position.X;

                if (maxWidth != 0)
                {
                    float startX = 0;

                    switch (_Parent._THA)
                    {
                        case HorizontalAlignment.Left:
                            startX = 0;
                            break;

                        case HorizontalAlignment.Center:
                            startX = (_Parent.Transform.Dimensions.X * _Parent.Transform.Scale.X - maxWidth) / 2;
                            break;

                        case HorizontalAlignment.Right:
                            startX = _Parent.Transform.Dimensions.X * _Parent.Transform.Scale.X - maxWidth;
                            break;
                    }

                    for (int j = 0; j < textAlignedRowCount; j++)//foreach (Letter letter in row)
                    {
                        if (startX != 0 || startY != 0)
                            textAligned[i][j].Transform.Position += new Vector3(startX, startY, 0);

                        retValue.Add(textAligned[i][j]);
                    }
                }
            }

            _Parent._text = retValue;

            _Parent._changed = false;
        }

        protected internal override void OnColliderEnter(Collider e)
        { }
    }
}
```

2.2.1.14 Core/SystemExt/Extensions.cs

```
/*
* (C) 2017 David Knieradl
*/

using DKEngine.Core.Components;
using System.Collections.Generic;
using System.Linq;

namespace DKEngine.Core.Ext
{
    public static class Extensions
    {
        public static void AddSafe<DataValue>(this Dictionary<string, DataValue> Destination, Component Target)
            where DataValue : Component
        {
            string Key = Target.Name;

            if (Engine.LoadingScene.ComponentCount.ContainsKey(Target.Name))
            {
                Target.Name = string.Format("{0}_(Copy {1})", Key, Engine.LoadingScene.ComponentCount[Target.Name]++);
                Key = Target.Name;
            }
            else
            {
                Engine.LoadingScene.ComponentCount.Add(Key, 1);
            }

            Destination.Add(Key, Target as DataValue);
        }

        public static void AddAll<T>(this List<T> list, params T[] stuff)
        {
            foreach (var item in stuff)
            {
                list.Add(item);
            }
        }

        public static float FindMaxZ(this List<GameObject> list)
        {
            return list.Max(obj => obj.Transform.Position.Z);
        }

        public static List<GameObject> GetGameObjectsInView(this IEnumerable<GameObject> list)
        {
            return list.Where(obj => obj.IsInView).ToList();
        }
    }
}
```

2.2.1.15 Core/SystemExt/WindowControl.cs

```
/*
* (C) 2017 David Knieradl
*/

using System;
using System.IO;
using System.Runtime.InteropServices;
using System.Timers;

namespace DKEngine.Core.Ext
{
    /// <summary>
    /// DKEngine window controller
    /// </summary>
    public static class WindowControl
    {
        [StructLayout(LayoutKind.Sequential)]
        private struct COORD
        {
            public short X;
            public short Y;

            public COORD(short x, short y)
            {
                this.X = x;
                this.Y = y;
            }
        }

        [DllImport("kernel32.dll")]
        private static extern IntPtr GetStdHandle(int handle);

        [DllImport("kernel32.dll", SetLastError = true)]
        private static extern bool SetConsoleDisplayMode(IntPtr ConsoleOutput, uint Flags, out COORD NewScreenBuffer-
Dimensions);

        [DllImport("user32.dll")]
        public static extern bool ShowWindow(IntPtr hWnd, int cmdShow);

        private static readonly IntPtr hConsole = GetStdHandle(-11);
        private static readonly IntPtr INVALID_HANDLE_VALUE = new IntPtr(-1);
        private static COORD xy = new COORD(100, 100);
        private static bool ConsoleStateChangeAvailable = true;

        internal static void WindowInit()
        {
            Console.CursorVisible = false;
            Console.SetOut(TextWriter.Null);
            Console.SetIn(TextReader.Null);

            Console.BufferHeight = Console.LargestWindowHeight;
            Console.BufferWidth = Console.LargestWindowWidth;

            Console.ForegroundColor = ConsoleColor.White;
            Console.BackgroundColor = ConsoleColor.Black;

            Console.Clear();

            WindowSizeChecker(null, null);

            Timer windowChecker = new Timer()
            {
                AutoReset = true,
                Enabled = true,
                Interval = 1000f
            };
            windowChecker.Elapsed += WindowSizeChecker;

            windowChecker.Start();
        }

        private static void WindowSizeChecker(object sender, ElapsedEventArgs e)
        {
            if (Console.WindowHeight != Console.LargestWindowHeight || Console.WindowWidth != Console.LargestWindowWidth)
            {
                if (ConsoleStateChangeAvailable)
                {
                    if (!SetConsoleDisplayMode(hConsole, 1, out xy))
                    {
                        ConsoleStateChangeAvailable = false;
                    }
                }

                Console.CursorVisible = false;
            }
        }
    }
}
```

2.2.1.16 Core/UI/Letter.cs

```
/*
* (C) 2017 David Knieradl
*/

namespace DKEngine.Core.UI
{
    internal sealed class Letter : GameObject
    {
        private Letter()
        { }

        public Letter(TextBlock Parent)
            : base(Parent)
        { }

        public override void Destroy()
        {
            try
            {
                if (Engine.LoadingScene.NewlyGeneratedComponents.Contains(this))
                {
                    Engine.LoadingScene.NewlyGeneratedComponents.Pop();
                }
            }
            catch
            { }

            try
            {
                Engine.RenderObjects.Remove(this);
            }
            catch { }

            Parent?.Child.Remove(this);

            Animator?.Destroy();

            Parent = null;
            Animator = null;
            Model = null;
        }
    }
}
```

2.2.1.17 Core/UI/Text.cs

```
namespace DKEngine.Core.UI
{
    public static class Text
    {
        public enum HorizontalAlignment
        {
            Left,
            Center,
            Right
        };

        public enum VerticalAlignment
        {
            Top,
            Center,
            Bottom
        };

        public enum InputType
        {
            All,
            AlphaNumerical,
            Alpha,
            Numerical
        };
    }
}
```

2.2.1.18 Core/UI/TextBlock.cs

```
/*
* (C) 2017 David Knieradl
*/

using DKEngine.Core.Components;
using DKEngine.Core.Scripts;
using System.Collections.Generic;
using System.Drawing;
using static DKEngine.Core.UI.Text;

namespace DKEngine.Core.UI
{
    public class TextBlock : GameObject, IText
    {
        public virtual string Text
        {
            get
            {
                return _textStr ?? throw new System.Exception("WTF PROC KDY KDE A JAK");
            }
            set
            {
                if (value != _textStr)
                {
                    _textStr = value ?? throw new System.Exception("WTF PROC KDY KDE A JAK");
                    _changed = true;
                }
            }
        }

        public Color? Background
        {
            get { return _bg; }
            set
            {
                _bg = value;

                if (value != null)
                    Model = new Material((Color)value, this);
            }
        }

        public float FontSize
        {
            get { return _FontSize; }
            set
            {
                if (value <= 0)
                {
                    _FontSize = 0.01f;
                    _changed = true;
                }
                else
                {
                    _FontSize = value;
                    _changed = true;
                }
            }
        }

        public HorizontalAlignment HAlignment
        {
            set
            {
                _HA = value;

                //if (_IsGUI)
                //{
                this.Transform.Position -= new Vector3(horiOffset, 0, 0);

                switch (value)
                {
                    case HorizontalAlignment.Left:
                        horiOffset = 0;
                        break;

                    case HorizontalAlignment.Center:
                        horiOffset = (Engine.Render.RenderWidth - this.Transform._ScaledDimensions.X) / 2;
                        break;

                    case HorizontalAlignment.Right:
                        horiOffset = Engine.Render.RenderWidth - this.Transform._ScaledDimensions.X;
                        break;
                }

                this.Transform.Position += new Vector3(horiOffset, 0, 0);
                //}

                //_changed = true;
            }
        }

        public VerticalAlignment VAlignment
        {
            set
            {
                _VA = value;

                //if (_IsGUI)
                //{
                this.Transform.Position -= new Vector3(0, vertOffset, 0);

                switch (value)
                {
                    case VerticalAlignment.Top:
                        vertOffset = 0;
                        break;

                    case VerticalAlignment.Center:
                        vertOffset = (Engine.Render.RenderHeight - this.Transform._ScaledDimensions.Y) / 2;
                        break;

                    case VerticalAlignment.Bottom:
```

```
                    vertOffset = Engine.Render.RenderHeight - this.Transform._ScaledDimensions.Y;
                    break;
            }

            this.Transform.Position += new Vector3(0, vertOffset, 0);
            //}

            //_changed = true;
        }
    }

    public HorizontalAlignment TextHAlignment
    {
        set
        {
            _THA = value;
            _changed = true;
        }
    }

    public VerticalAlignment TextVAlignment
    {
        set
        {
            _TVA = value;
            _changed = true;
        }
    }

    public bool TextShadow = false;

    internal List<Letter> _text = new List<Letter>();
    internal float _FontSize = 1;
    internal Color? _bg;
    internal string _textStr = "";

    internal HorizontalAlignment _HA = HorizontalAlignment.Left;
    internal VerticalAlignment _VA = VerticalAlignment.Top;
    internal HorizontalAlignment _THA = HorizontalAlignment.Left;
    internal VerticalAlignment _TVA = VerticalAlignment.Top;

    internal float vertOffset = 0;
    internal float horiOffset = 0;
    internal bool _changed = false;

    public TextBlock()
        : base()
    { }

    public TextBlock(GameObject Parent)
        : base(Parent)
    { }

    protected override void Initialize()
    {
        this.VAlignment = _VA;
        this.HAlignment = _HA;
        this.InitNewScript<TextControlScript>();
    }

    internal override void Render()
    { Model?.Render(this, _bg); }

    public override void Destroy()
    {
        base.Destroy();
    }
  }
}
```

2.2.1.19 Core/Database.cs

```
using DKEngine.Core.Components;
using DKEngine.Properties;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Resources;

namespace DKEngine.Core
{
    /// <summary>
    /// DKEngine library database holding all loaded materials, scenes, etc.
    /// </summary>
    public static class Database
    {
        private enum Font
        {
            Num0,
            Num1,
            Num2,
            Num3,
            Num4,
            Num5,
            Num6,
            Num7,
            Num8,
            Num9,
            A,
            AngleBracketLeft,
            AngleBracketRight,
            ArrowToLeft,
            ArrowToRight,
            ArrowToTop,
            B,
            Backslash,
            BraceLeft,
            BraceRight,
            BracketLeft,
            BracketRight,
            C,
            Colon,
            Comma,
            D,
            Dot,
            E,
            Equals,
            ExclamationMark,
            F,
            G,
            H,
            Hashtag,
            I,
            J,
            K,
            L,
            M,
            Minus,
            N,
            O,
            P,
            Percent,
            Q,
            QuestionMark,
            QuotationMarks,
            R,
            S,
            Semicolon,
            Slash,
            StarLarge,
            StarSmall,
            T,
            U,
            Underscore,
            V,
            W,
            X,
            Y,
            Z,
            NumberOfTypes
```

```
};

private static Dictionary<char, Font> font = new Dictionary<char, Font>()
{
    { '0' , Font.Num0 },
    { '1' , Font.Num1 },
    { '2' , Font.Num2 },
    { '3' , Font.Num3 },
    { '4' , Font.Num4 },
    { '5' , Font.Num5 },
    { '6' , Font.Num6 },
    { '7' , Font.Num7 },
    { '8' , Font.Num8 },
    { '9' , Font.Num9 },
    { 'A' , Font.A },
    { 'B' , Font.B },
    { 'C' , Font.C },
    { 'D' , Font.D },
    { 'E' , Font.E },
    { 'F' , Font.F },
    { 'G' , Font.G },
    { 'H' , Font.H },
    { 'I' , Font.I },
    { 'J' , Font.J },
    { 'K' , Font.K },
    { 'L' , Font.L },
    { 'M' , Font.M },
    { 'N' , Font.N },
    { 'O' , Font.O },
    { 'P' , Font.P },
    { 'Q' , Font.Q },
    { 'R' , Font.R },
    { 'S' , Font.S },
    { 'T' , Font.T },
    { 'U' , Font.U },
    { 'V' , Font.V },
    { 'W' , Font.W },
    { 'X' , Font.X },
    { 'Y' , Font.Y },
    { 'Z' , Font.Z },
    { '-' , Font.Minus },
    { '?' , Font.QuestionMark },
    { '!' , Font.ExclamationMark },
    { '.' , Font.Dot },
    { ':' , Font.Colon },
    { ',' , Font.Comma },
    { '[' , Font.AngleBracketLeft },
    { ']' , Font.AngleBracketRight },
    { '>' , Font.ArrowToRight },
    { '<' , Font.ArrowToLeft },
    { '^' , Font.ArrowToTop },
    { '{' , Font.BraceLeft },
    { '}' , Font.BraceRight },
    { '(' , Font.BracketLeft },
    { ')' , Font.BracketRight },
    { '=' , Font.Equals },
    { '#' , Font.Hashtag },
    { '%' , Font.Percent },
    { '"' , Font.QuotationMarks },
    { ';' , Font.Semicolon },
    { '☼' , Font.StarLarge },
    { '*' , Font.StarSmall },
    { '_' , Font.Underscore },
    { '/' , Font.Slash },
    { '\\' , Font.Backslash }
};

private static List<Material> letterMaterial = new List<Material>();

private static void CreateLetterReferences()
{
    using (BinaryReader br = new BinaryReader(new MemoryStream(Resources.FontFile)))
    {
        int lenght = br.ReadInt32();

        for (int index = 0; index < lenght; index++)
        {
            byte[] byteArray = br.ReadBytes(br.ReadInt32());

            using (MemoryStream ms = new MemoryStream(byteArray))
            {
                letterMaterial.Add(new Material((Bitmap)Image.FromStream(ms)));
            }
        }
    }
}

private static Dictionary<string, Material> CachedMaterials = new Dictionary<string, Material>();
private static Dictionary<string, Scene> CachedScenes = new Dictionary<string, Scene>();

internal static void InitDatabase()
{
    AddNewGameObjectMaterial("border", new Material(Resources.border));
    AddNewGameObjectMaterial("splashScreen", new Material(Resources.DKEngine_splash2));

    CreateLetterReferences();
}

internal static Scene GetScene(string Key)
{
    Scene retValue = null;

    try
    {
        retValue = CachedScenes[Key];
    }
    catch
    { }

    return retValue;
}

public static Material GetLetter(this char ch)
{
    Material retValue = null;

    try
    {
        retValue = letterMaterial[(int)font[Char.ToUpper(ch)]];
    }
    catch
    {
        retValue = letterMaterial[(int)font['?']];
    }

    return retValue;
}

public static void AddNewGameObjectMaterial(string ObjectName, Material Object)
{
    try
    {
        if (Object != null)
        {
            CachedMaterials.Add(ObjectName, Object);
        }
        else
            throw new Exception("Material is null\n" + Object.ToString());
    }
    catch (Exception e)
    {
        Debug.WriteLine("Object not found\n" + e);
    }
}

public static Material GetGameObjectMaterial(string Key)
{
    Material retValue = null;

    try
    {
        retValue = CachedMaterials[Key];
    }
    catch (Exception e)
    {
        Debug.WriteLine("Object not found\n" + e);
    }

    return retValue;
}

public static Material GetGameObjectMaterial(int Position)
{
    Material retValue = null;

    try
    {
        retValue = CachedMaterials.ElementAtOrDefault(Position).Value;
    }
    catch (Exception e)
    {
        Debug.WriteLine("Object not found\n" + e);
    }

    return retValue;
}

public static string GetMaterialDatabaseKey(int Position)
{
    return CachedMaterials.ElementAtOrDefault(Position).Key; //.FirstOrDefault(x => x.Value == Position).Key;
}

public static void LoadResources(ResourceSet source)
{
    foreach (DictionaryEntry entry in source)
    {
        if (entry.Value is Image)
        {
            AddNewGameObjectMaterial((string)entry.Key, new Material((Bitmap)entry.Value));
        }
    }
}

internal static void RewriteWorld(string Name, object[] argsPreLoad = null)
{
    try
    {
        Engine.LoadingScene = CachedScenes[Name];

        object[] preArgs = argsPreLoad ?? Engine.LoadingScene.argsPreLoad;
        object[] postArgs = Engine.LoadingScene.argsPostLoad;

        var list = Engine.LoadingScene.AllComponents.ToList();
        for (int i = 0; i < list.Count; i++)
        {
            list[0].Value.Destroy();
            list.RemoveAt(0);
            list = Engine.LoadingScene.AllComponents.ToList();
        }

        for (int i = 0; i < Engine.LoadingScene.AllBehaviors.Count; i++)
        {
            Engine.LoadingScene.AllBehaviors[0].Destroy();
        }

        Engine.LoadingScene = (Scene)Activator.CreateInstance(Engine.LoadingScene.GetType());

        Engine.LoadingScene.argsPreLoad = preArgs;
        Engine.LoadingScene.argsPostLoad = postArgs;

        Engine.LoadingScene.Set(Engine.LoadingScene.argsPreLoad);
        Engine.LoadingScene.Init();
        CachedScenes[Name] = Engine.LoadingScene;
    }
    catch
    { }
}

internal static void AddScene(Scene Source)
{
    try
```

```
            {
                CachedScenes.Add(Source.Name, Source);
            }
            catch { }
        }
    }
}
```

2.2.1.20 Core/GameObject.cs

```
using DKEngine.Core.Components;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Reflection;

namespace DKEngine.Core
{
    /// <summary>
    /// Primitive type for all renderable objects
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Component" />
    public class GameObject : Component
    {
        /// <summary>
        /// The GameObject has shadow
        /// </summary>
        public bool HasShadow = false;

        /// <summary>
        /// Gets a value indicating whether this instance is in view.
        /// </summary>
        /// <value>
        ///   <c>true</c> if this instance is in view; otherwise, <c>false</c>.
        /// </value>
        public bool IsInView
        {
            get
            {
                float X = this.IsGUI ? 0 : Engine.BaseCam != null ? Engine.BaseCam.X : 0;
                float Y = this.IsGUI ? 0 : Engine.BaseCam != null ? Engine.BaseCam.Y : 0;

                return (this.Transform.Position.X + this.Transform._ScaledDimensions.X >= X && this.Transform.Position.X < X
+ Engine.Render.RenderWidth && this.Transform.Position.Y + this.Transform._ScaledDimensions.Y >= Y && this.Trans-
form.Position.Y < Y + Engine.Render.RenderHeight);
            }
        }

        /// <summary>
        /// Gets or sets a value indicating whether this instance is GUI.
        /// </summary>
        /// <value>
        ///   <c>true</c> if this instance is GUI; otherwise, <c>false</c>.
        /// </value>
        public bool IsGUI
        {
            get { return Parent != null ? Parent.IsGUI : _IsGUI; }
            set { _IsGUI = value; }
        }

        /// <summary>
        /// Gets or sets the name of the type.
        /// </summary>
        /// <value>
        /// The name of the type.
        /// </value>
        public string TypeName
        {
            get { return _typeName; }
            set
            {
                _typeName = value;
                this.Model = Database.GetGameObjectMaterial(value);
            }
        }

```

```
/// <summary>
/// Gets or sets the model.
/// </summary>
/// <value>
/// The model.
/// </value>
public Material Model
{
    get { return _Model; }
    set
    {
        if (value != _Model && value != null)
        {
            _Model = value;
            this.Transform.Dimensions = new Vector3(value.Width, value.Height, 1);

            if (Animator?.Animations.Count == 0)
            {
                Animator.AddAnimation("default", _Model);
                Animator.Play("default");
            }
        }
    }
}

/// <summary>
/// Gets or sets the collider.
/// </summary>
/// <value>
/// The collider.
/// </value>
public Collider Collider
{
    get { return _collider; }
    set
    {
        if (_collider != value)
        {
            if (_collider != null)
            {
                foreach (Script scr in this.Scripts)
                {
                    _collider.CollisionEvent -= scr.CollisionHandler;
                    scr.CollisionHandler = null;
                }
            }

            if (value != null)
            {
                foreach (Script scr in this.Scripts)
                {
                    scr.CollisionHandler = new Collider.CollisionEnterHandler(scr.OnColliderEnter);
                    value.CollisionEvent += scr.CollisionHandler;
                }
            }

            _collider = value;
        }
    }
}

/// <summary>
/// Gets or sets the animator.
/// </summary>
/// <value>
/// The animator.
/// </value>
public Animator Animator { get; set; }

/// <summary>
/// Gets or sets the sound source.
/// </summary>
/// <value>
/// The sound source.
/// </value>
public SoundSource SoundSource { get; set; }

```

```
/// <summary>
/// Gets or sets the foreground.
/// </summary>
/// <value>
/// The foreground.
/// </value>
public Color? Foreground { get; set; }

/// <summary>
/// Gets the transform.
/// </summary>
/// <value>
/// The transform.
/// </value>
public Transform Transform { get; }

/// <summary>
/// Gets the list of childs.
/// </summary>
/// <value>
/// The child.
/// </value>
public List<GameObject> Child { get; }

internal List<Script> Scripts { get; }
internal bool _IsGUI = false;
internal string _typeName = "";
internal Material _Model = null;
internal Collider _collider = null;

public GameObject()
    : base(null)
{
    this.Child = new List<GameObject>();
    this.Scripts = new List<Script>();
    this.Transform = new Transform(this)
    {
        Dimensions = new Vector3(1, 1, 1),
        Scale = new Vector3(1, 1, 1),
        Position = new Vector3(0, 0, 0)
    };
}

public GameObject(GameObject Parent)
    : base(Parent)
{
    this.Child = new List<GameObject>();
    this.Scripts = new List<Script>();
    this.Transform = new Transform(this)
    {
        Dimensions = new Vector3(1, 1, 1),
        Scale = new Vector3(1, 1, 1),
        Position = new Vector3(0, 0, 0)
    };

    if (Parent != null)
    {
        this.Parent = Parent;

        Parent.Child.Add(this);
        this.Transform.Position = Parent.Transform.Position;
        this.Transform.Scale = Parent.Transform.Scale;
    }
}

internal override void Init()
{
    Initialize();

    try
    {
        if (Parent == null)
            Engine.LoadingScene.Model.Add(this);

        Engine.LoadingScene.GameObjectsToAddToRender.Push(this);
    }
    catch (Exception e)
```

```
            {
                Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
            }
        }

        protected virtual void Initialize()
        { }

        /// <summary>
        /// Initializes the new script.
        /// </summary>
        /// <typeparam name="T">Scirpt</typeparam>
        public void InitNewScript<T>() where T : Script
        {
            this.Scripts.Add((T)Activator.CreateInstance(typeof(T), this));
        }

        /// <summary>
        /// Initializes the new component.
        /// </summary>
        /// <typeparam name="T">Component</typeparam>
        public void InitNewComponent<T>() where T : Component
        {
            if (typeof(T) == typeof(Animator) || typeof(T).IsSubclassOf(typeof(Animator)))
            {
                if (this.Animator == null)
                {
                    this.Animator = new Animator(this);
                }

                return;
            }

            if (typeof(T) == typeof(Collider) || typeof(T).IsSubclassOf(typeof(Collider)))
            {
                if (this.Collider == null)
                {
                    Type t = typeof(T);
                    this.Collider = (Collider)t.Assembly.CreateInstance(t.FullName, false, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public, null, new object[] { this }, null, null);
                }

                return;
            }

            if (typeof(T) == typeof(SoundSource) || typeof(T).IsSubclassOf(typeof(SoundSource)))
            {
                if (this.SoundSource == null)
                {
                    Type t = typeof(T);
                    this.SoundSource = (SoundSource)t.Assembly.CreateInstance(t.FullName, false, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public, null, new object[] { this }, null, null);
                }

                return;
            }
        }

        public override void Destroy()
        {
            try
            {
                if (Engine.LoadingScene.NewlyGeneratedComponents.Contains(this))
                {
                    Engine.LoadingScene.DestroyObjectAwaitList.Add(this);
                    return;
                }
            }
            catch { }

            try
            {
                Engine.LoadingScene.AllComponents.Remove(this.Name);
            }
            catch { }

            try
            {
                Engine.RenderObjects.Remove(this);
            }
            catch { }

            try
            {
                Engine.LoadingScene.Model.Remove(this);
            }
            catch { }

            int ScriptCount = this.Scripts.Count;
            for (int i = 0; i < ScriptCount; i++)
                Scripts[0].Destroy();

            int ChildCount = this.Child.Count;
            for (int i = 0; i < ChildCount; i++)
                Child[0].Destroy();

            this.Animator?.Destroy();
            this.Animator = null;

            this.Collider?.Destroy();
            this.Collider = null;

            this.Parent = null;
        }

        internal virtual void Render()
        { Model?.Render(this, Foreground); }

        /// <summary>
        /// Finds the specified GameObject of desired name.
        /// </summary>
        /// <typeparam name="T">Type</typeparam>
        /// <param name="Name">Desired name</param>
        /// <returns></returns>
        public static new T Find<T>(string Name) where T : GameObject
        {
            T retValue = null;

            try
            {
                retValue = (T)Engine.LoadingScene.AllComponents[Name];
            }
            catch (Exception ex)
            {
                Debug.WriteLine("Object not found\n" + ex);
            }

            return retValue;
        }

        /// <summary>
        /// Finds the specified GameObject of desired name.
        /// </summary>
        /// <param name="Name">Desired name</param>
        /// <returns></returns>
        public static GameObject Find(string Name)
        {
            GameObject retValue = null;

            try
            {
                retValue = (GameObject)Engine.LoadingScene.AllComponents[Name];
            }
            catch (Exception ex)
            {
                Debug.WriteLine("Object not found\n" + ex);
            }

            return retValue;
        }

        /// <summary>
        /// Instantiates GameObject.
        /// </summary>
        /// <typeparam name="T">Type</typeparam>
        /// <param name="Position">The position</param>
        /// <param name="Dimensions">The dimensions</param>
        /// <param name="Scale">The scale</param>
        /// <returns></returns>
        public static T Instantiate<T>(Vector3 Position, Vector3 Dimensions, Vector3 Scale)
            where T : GameObject, new()
        {
            T retValue = new T();

            retValue.Transform.Position = Position;
            retValue.Transform.Dimensions = Dimensions;
            retValue.Transform.Scale = Scale;

            return retValue;
        }

        /// <summary>
        /// Instantiates GameObject.
        /// </summary>
        /// <typeparam name="T">Type</typeparam>
        /// <param name="Transform">The transform</param>
        /// <returns></returns>
        public static T Instantiate<T>(Transform @Transform)
            where T : GameObject, new()
        {
            return Instantiate<T>(@Transform.Position, @Transform.Dimensions, @Transform.Scale);
        }
    }
}
```

2.2.1.21 Core/Scene.cs

```
using DKEngine.Core.Components;
using System.Collections.Generic;

namespace DKEngine.Core
{
    /// <summary>
    /// DKEngine library scene
    /// </summary>
    /// <seealso cref="DKEngine.IPage" />
    public abstract class Scene : IPage
    {
        public string Name = "";

        internal Camera BaseCamera;

        internal readonly Dictionary<string, Component> AllComponents;
        internal readonly Dictionary<string, int> ComponentCount;
        //internal readonly Dictionary<string, GameObject> AllGameObjects;

        internal readonly List<GameObject> Model;
        internal readonly List<Behavior> AllBehaviors;
        internal readonly List<Collider> AllGameObjectsColliders;

        internal readonly Stack<Component> NewlyGeneratedComponents;
        internal readonly Stack<Behavior> NewlyGeneratedBehaviors;

        internal readonly Stack<GameObject> GameObjectsToAddToRender;
        internal readonly Stack<GameObject> GameObjectsAddedToRender;

        internal readonly List<GameObject> DestroyObjectAwaitList;

        internal object[] argsPreLoad;
        internal object[] argsPostLoad;

        public Scene()
        {
            AllComponents = new Dictionary<string, Component>(0xFFFF);
            ComponentCount = new Dictionary<string, int>(0xFFFF);

            AllBehaviors = new List<Behavior>(0xFFFF);
            Model = new List<GameObject>(0xFFFF);
            AllGameObjectsColliders = new List<Collider>(0xFFFF);

            NewlyGeneratedComponents = new Stack<Component>(0xFFFF);
            NewlyGeneratedBehaviors = new Stack<Behavior>(0xFFFF);

            GameObjectsToAddToRender = new Stack<GameObject>(0xFFFF);
            GameObjectsAddedToRender = new Stack<GameObject>(0xFFFF);

            DestroyObjectAwaitList = new List<GameObject>(0xFFFF);
        }

        /// <summary>
        /// Initializes model of Scene.
        /// </summary>
        public abstract void Init();

        /// <summary>
        /// Sets the specified arguments.
        /// </summary>
        /// <param name="args">The arguments</param>
        public virtual void Set(params object[] args)
        { }

        /// <summary>
        /// Unloads this instance.
        /// </summary>
        public abstract void Unload();

        public static T Find<T>(string name)
            where T : Scene
        {
            return (T)Database.GetScene(name);
        }
    }
}
```

2.2.1.22 Core/Script.cs

```
using DKEngine.Core.Components;

namespace DKEngine.Core
{
    /// <summary>
    /// Script base class
    /// </summary>
    /// <seealso cref="DKEngine.Core.Components.Behavior" />
    public abstract class Script : Behavior
    {
        internal Collider.CollisionEnterHandler CollisionHandler;

        public Script(GameObject Parent)
            : base(Parent)
        {
            if (Parent.Collider != null)
            {
                CollisionHandler = new Collider.CollisionEnterHandler(OnColliderEnter);
                Parent.Collider.CollisionEvent += CollisionHandler;
            }
        }

        protected internal abstract void OnColliderEnter(Collider e);

        public override void Destroy()
        {
            try
            {
                Engine.LoadingScene.AllComponents.Remove(this.Name);
            }
            catch { }

            try
            {
                Engine.LoadingScene.AllBehaviors.Remove(this);
            }
            catch { }

            if (UpdateHandle != null)
                Engine.UpdateEvent -= UpdateHandle;

            if (CollisionHandler != null)
```

```
                Parent.Collider.CollisionEvent -= CollisionHandler;

            Parent.Scripts.Remove(this);
            Parent = null;
            UpdateHandle = null;
        }
    }
}
```

2.2.1.23 Data/SplashScreen.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace DKEngine
{
    internal sealed class SplashScreen : GameObject
    {
        public SplashScreen()
        {
            this.TypeName = "splashScreen";
            this.InitNewComponent<Animator>();
        }

        public SplashScreen(GameObject Parent)
            : base(Parent)
        {
            this.TypeName = "splashScreen";
            this.InitNewComponent<Animator>();
        }

        protected override void Initialize()
        { }
    }
}
```

2.2.1.24 Data/SplashScreenScene.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace DKEngine.Data
{
    internal class SplashScreenScene : Scene
    {
        internal SplashScreen Splash;

        public override void Init()
        {
            Splash = new SplashScreen();
            Splash.Transform.Position = new Vector3(-32, 0, 0);
            Splash.Transform.Scale = new Vector3(0.5f, 0.5f, 0);
            Camera splashScreenCam = new Camera();
        }

        public override void Unload()
        { }
    }
}
```

2.2.2 MarIO

2.2.2.1 Program.cs

```
using DKEngine;
using DKEngine.Core;
using MarIO.Assets.Scenes;
using MarIO.Assets.Sprites;
using System.Globalization;

namespace MarIO
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Engine.Init();

            Engine.Sound.SoundVolume = 0.5f;

            Database.LoadResources(Sprites.ResourceManager.GetResourceSet(CultureInfo.CurrentCulture, true, true));
            Database.LoadResources(Enemies.ResourceManager.GetResourceSet(CultureInfo.CurrentCulture, true, true));

            Engine.LoadSceneToMemory<MainMenu>();
            Engine.LoadSceneToMemory<Level_1_1>();
            Engine.LoadSceneToMemory<GameOver>();
            Engine.LoadSceneToMemory<WorldScreen>();

            Engine.ChangeScene(nameof(MainMenu));
        }
    }
}
```

2.2.2.2 Shared.cs

```
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using MarIO.Assets.Models.Miscellaneous;
using System;
using System.Collections.Generic;
using System.Diagnostics;

namespace MarIO
{
    public static class Shared
    {
        public static class Mechanics
        {
            public static SoundOutput FXPlayer;
            public static SoundSource FXSoundSource { get { return FXPlayer.SoundSource; } }

            private static byte _coinsCount = 0;

            public static string GameScoreStr { get { return string.Format($"{GameScore:00000000}"); } }
            public static short GameScore { get; set; } = 0;
            public static byte Lives { get; set; } = 3;

            public static byte CoinsCount
            {
                get { return _coinsCount; }
                set
                {
                    _coinsCount = value;
                    if (_coinsCount > 99)
                    {
                        Lives++;
                        _coinsCount = 0;
                    }
                }
            }

            public readonly static Stopwatch TimeCounter = new Stopwatch();

            private readonly static TimeSpan LevelTime = new TimeSpan(0, 5, 0);

            public static TimeSpan TimeLeft
            {
                get { return LevelTime - TimeCounter.Elapsed; }
            }

            //public static
            public static Type LastWorldType;

            public static Mario.State MarioCurrentState
            {
                get;
                set;
            } = Mario.State.Super;

            public const uint OverworldBackground = 0xFF30A0DD;
            public const uint WorldChangeBackground = 0x00000000;

            public const int GOOMBA_POINTS = 100;
            public const int COIN_SCORE = 100;
            public const int MUSHROOM_SCORE = 200;
            public const int FLOWER_SCORE = 300;
            public const int STAR_SCORE = 500;
        }

        public static class AnimatedWorldReferences
        {
            public static List<Block> BlocksToUpdate = new List<Block>();
            public static List<float> BlocksStartPositions = new List<float>();

            public static List<TextBlock> FloatingTexts = new List<TextBlock>();
            public static List<float> FloatingTextStartPosition = new List<float>();

            public static Stack<Block> SpecialActions = new Stack<Block>();

            public static List<Coin> FloatingCoins = new List<Coin>();
            public static List<float> FloatingCoinsStartPosition = new List<float>();
        }

        public static class Assets
        {
            public static class Sounds
            {
                public const string OVERWORLD_THEME = @".\Assets\Sounds\Overworld_Theme.mp3";
                public const string MARIO_JUMP_FX = @".\Assets\Sounds\smb_jump-small.mp3";
                public const string PIPE_ENTER_FX = @".\Assets\Sounds\smb_pipe.mp3";
                public const string COIN_GET_FX = @".\Assets\Sounds\smb_coin.mp3";
                public const string UP_1_FX = @".\Assets\Sounds\smb_1-up.mp3";
                public const string BREAK_BLOCK_FX = @".\Assets\Sounds\smb_breakblock.mp3";
                public const string MARIO_DIE_FX = @".\Assets\Sounds\smb_mariodie.mp3";
                public const string POWER_UP_FX = @".\Assets\Sounds\smb_powerup.mp3";
                public const string STOMP_FX = @".\Assets\Sounds\smb_stomp.mp3";

                public static readonly Sound OVERWORLD_THEME_SOUND = new Sound(OVERWORLD_THEME);
                public static readonly Sound FX_MARIO_JUMP_SOUND = new Sound(MARIO_JUMP_FX);
                public static readonly Sound FX_PIPE_ENTER_SOUND = new Sound(PIPE_ENTER_FX);
                public static readonly Sound FX_1_UP_SOUND = new Sound(UP_1_FX);
                public static readonly Sound FX_BREAK_BLOCK_SOUND = new Sound(BREAK_BLOCK_FX);
                public static readonly Sound FX_MARIO_DIE_SOUND = new Sound(MARIO_DIE_FX);
                public static readonly Sound FX_POWER_UP_SOUND = new Sound(POWER_UP_FX);
                public static readonly Sound FX_STOMP_SOUND = new Sound(STOMP_FX);
            }

            public static class Animations
            {
                #region Mario

                private const string POWERUP_LEFT = "powerup_left";
                private const string POWERUP_LEFT_MAT = "mario_powerup_left";

                private const string POWERUP_RIGHT = "powerup_right";
                private const string POWERUP_RIGHT_MAT = "mario_powerup_right";

                private const string CROUCHING_LEFT = "crouch_left";
                private const string CROUCHING_LEFT_MAT = "mario_crouch_left";

                private const string CROUCHING_RIGHT = "crouch_right";
                private const string CROUCHING_RIGHT_MAT = "mario_crouch_right";

                /*--------------- SMALL -----------------*/

                public const string MARIO_IDLE_LEFT = "idle_left";
                public const string MARIO_IDLE_LEFT_MAT = "mario_left";

                public const string MARIO_IDLE_RIGHT = "idle_right";
                public const string MARIO_IDLE_RIGHT_MAT = "mario_right";

                public const string MARIO_MOVE_LEFT = "move_left";
                public const string MARIO_MOVE_LEFT_MAT = "mario_move_left";

                public const string MARIO_MOVE_RIGHT = "move_right";
                public const string MARIO_MOVE_RIGHT_MAT = "mario_move_right";

                public const string MARIO_JUMP_LEFT = "jump_left";
                public const string MARIO_JUMP_LEFT_MAT = "mario_jump_left";

                public const string MARIO_JUMP_RIGHT = "jump_right";
                public const string MARIO_JUMP_RIGHT_MAT = "mario_jump_right";

                public const string MARIO_DEAD = "dead";
                public const string MARIO_DEAD_MAT = "mario_dead";

                public const string MARIO_CROUCHING_LEFT = CROUCHING_LEFT;
                public const string MARIO_CROUCHING_LEFT_MAT = MARIO_IDLE_LEFT_MAT;

                public const string MARIO_CROUCHING_RIGHT = CROUCHING_RIGHT;
                public const string MARIO_CROUCHING_RIGHT_MAT = MARIO_IDLE_RIGHT_MAT;

                /*--------------- SUPER -----------------*/

                public const string MARIO_SUPER_IDLE_LEFT = "super_" + MARIO_IDLE_LEFT;
                public const string MARIO_SUPER_IDLE_LEFT_MAT = "super_" + MARIO_IDLE_LEFT_MAT;

                public const string MARIO_SUPER_IDLE_RIGHT = "super_" + MARIO_IDLE_RIGHT;
                public const string MARIO_SUPER_IDLE_RIGHT_MAT = "super_" + MARIO_IDLE_RIGHT_MAT;

                public const string MARIO_SUPER_MOVE_LEFT = "super_" + MARIO_MOVE_LEFT;
                public const string MARIO_SUPER_MOVE_LEFT_MAT = "super_" + MARIO_MOVE_LEFT_MAT;

                public const string MARIO_SUPER_MOVE_RIGHT = "super_" + MARIO_MOVE_RIGHT;
                public const string MARIO_SUPER_MOVE_RIGHT_MAT = "super_" + MARIO_MOVE_RIGHT_MAT;

                public const string MARIO_SUPER_JUMP_LEFT = "super_" + MARIO_JUMP_LEFT;
                public const string MARIO_SUPER_JUMP_LEFT_MAT = "super_" + MARIO_JUMP_LEFT_MAT;

                public const string MARIO_SUPER_JUMP_RIGHT = "super_" + MARIO_JUMP_RIGHT;
                public const string MARIO_SUPER_JUMP_RIGHT_MAT = "super_" + MARIO_JUMP_RIGHT_MAT;

                public const string MARIO_SUPER_POWERUP_LEFT = "super_" + POWERUP_LEFT;
                public const string MARIO_SUPER_POWERUP_LEFT_MAT = "super_" + POWERUP_LEFT_MAT;

                public const string MARIO_SUPER_POWERUP_RIGHT = "super_" + POWERUP_RIGHT;
                public const string MARIO_SUPER_POWERUP_RIGHT_MAT = "super_" + POWERUP_RIGHT_MAT;

                public const string MARIO_SUPER_CROUCHING_LEFT = "super_" + CROUCHING_LEFT;
                public const string MARIO_SUPER_CROUCHING_LEFT_MAT = "super_" + CROUCHING_LEFT_MAT;

                public const string MARIO_SUPER_CROUCHING_RIGHT = "super_" + CROUCHING_RIGHT;
                public const string MARIO_SUPER_CROUCHING_RIGHT_MAT = "super_" + CROUCHING_RIGHT_MAT;

                /*--------------- FIRE -----------------*/

                public const string MARIO_FIRE_IDLE_LEFT = "fire_" + MARIO_IDLE_LEFT;
                public const string MARIO_FIRE_IDLE_LEFT_MAT = "fire_" + MARIO_IDLE_LEFT_MAT;

                public const string MARIO_FIRE_IDLE_RIGHT = "fire_" + MARIO_IDLE_RIGHT;
                public const string MARIO_FIRE_IDLE_RIGHT_MAT = "fire_" + MARIO_IDLE_RIGHT_MAT;

                public const string MARIO_FIRE_MOVE_LEFT = "fire_" + MARIO_MOVE_LEFT;
                public const string MARIO_FIRE_MOVE_LEFT_MAT = "fire_" + MARIO_MOVE_LEFT_MAT;

                public const string MARIO_FIRE_MOVE_RIGHT = "fire_" + MARIO_MOVE_RIGHT;
                public const string MARIO_FIRE_MOVE_RIGHT_MAT = "fire_" + MARIO_MOVE_RIGHT_MAT;

                public const string MARIO_FIRE_JUMP_LEFT = "fire_" + MARIO_JUMP_LEFT;
                public const string MARIO_FIRE_JUMP_LEFT_MAT = "fire_" + MARIO_JUMP_LEFT_MAT;

                public const string MARIO_FIRE_JUMP_RIGHT = "fire_" + MARIO_JUMP_RIGHT;
                public const string MARIO_FIRE_JUMP_RIGHT_MAT = "fire_" + MARIO_JUMP_RIGHT_MAT;

                public const string MARIO_FIRE_POWERUP_LEFT = "fire_" + POWERUP_LEFT;
                public const string MARIO_FIRE_POWERUP_LEFT_MAT = "fire_" + POWERUP_LEFT_MAT;

                public const string MARIO_FIRE_POWERUP_RIGHT = "fire_" + POWERUP_RIGHT;
                public const string MARIO_FIRE_POWERUP_RIGHT_MAT = "fire_" + POWERUP_RIGHT_MAT;

                public const string MARIO_FIRE_CROUCHING_LEFT = "fire_" + CROUCHING_LEFT;
                public const string MARIO_FIRE_CROUCHING_LEFT_MAT = "fire_" + CROUCHING_LEFT_MAT;

                public const string MARIO_FIRE_CROUCHING_RIGHT = "fire_" + CROUCHING_RIGHT;
                public const string MARIO_FIRE_CROUCHING_RIGHT_MAT = "fire_" + CROUCHING_RIGHT_MAT;
```

```
            
            /*--------------- INVINCIBLE -----------------*/
            
            /*public const string MARIO_INVINCIBLE_IDLE_LEFT;
            public const string MARIO_INVINCIBLE_IDLE_LEFT_MAT;
            
            public const string MARIO_INVINCIBLE_IDLE_RIGHT;
            public const string MARIO_INVINCIBLE_IDLE_RIGHT_MAT;
            
            public const string MARIO_INVINCIBLE_MOVE_LEFT;
            public const string MARIO_INVINCIBLE_MOVE_LEFT_MAT;
            
            public const string MARIO_INVINCIBLE_MOVE_RIGHT;
            public const string MARIO_INVINCIBLE_MOVE_RIGHT_MAT;
            
            public const string MARIO_INVINCIBLE_JUMP_LEFT;
            public const string MARIO_INVINCIBLE_JUMP_LEFT_MAT;
            
            public const string MARIO_INVINCIBLE_JUMP_RIGHT;
            public const string MARIO_INVINCIBLE_JUMP_RIGHT_MAT;
            
            public const string MARIO_INVINCIBLE_DEAD;
            public const string MARIO_INVINCIBLE_DEAD_MAT;*/
            
            #endregion Mario
        }
    }
  }
}
```

2.2.2.3 SystemExt.cs

```
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using MarIO.Assets.Models.Miscellaneous;
using System.Drawing;

namespace MarIO
{
    public static class SystemExt
    {
        public static void AddAsFloatingText(this TextBlock txBlock)
        {
            Shared.AnimatedWorldReferences.FloatingTexts.Add(txBlock);
            Shared.AnimatedWorldReferences.FloatingTextStartPosition.Add(txBlock.Transform.Position.Y);
        }

        public static void AnimateBlockCollision(this Block block)
        {
            if (Shared.Mechanics.MarioCurrentState == Mario.State.Small || block.HadBonus)
            {
                block.State = Block.CollisionState.Up;

                Shared.AnimatedWorldReferences.BlocksToUpdate.Add(block);
                Shared.AnimatedWorldReferences.BlocksStartPositions.Add(block.Transform.Position.Y);
            }
            else
            {
            }
        }

        public static void AddAsFloatingCoin(this Coin coin)
        {
            Shared.AnimatedWorldReferences.FloatingCoins.Add(coin);
            Shared.AnimatedWorldReferences.FloatingCoinsStartPosition.Add(coin.Transform.Position.Y);
        }

        public static Color ToColor(this uint color)
        {
            byte a = (byte)(color >> 24);
            byte r = (byte)(color >> 16);
            byte g = (byte)(color >> 8);
            byte b = (byte)(color >> 0);
            return Color.FromArgb(a, r, g, b);
        }
    }
}
```

2.2.2.4 Assets/Models/Miscellaneous/Coin.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Models.Miscellaneous
{
public class Coin : GameObject
{
public static Sound COIN_FX = new Sound(Shared.Assets.Sounds.COIN_GET_FX);

public Coin()
{ }

public Coin(GameObject Parent)
: base(Parent)
{ }

protected override void Initialize()
{
this.Name = "coin";
//this.TypeName = "coin";
this.InitNewComponent<Animator>();
this.Animator.AddAnimation("default", Database.GetGameObjectMaterial("coin"));
}
}
}
```

2.2.2.5 Assets/Models/Miscellaneous/Heart.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Models.Miscellaneous
{
    public class Heart : GameObject
    {
        public Heart()
        { }

        public Heart(GameObject Parent)
            : base(Parent)
        { }

        protected override void Initialize()
        {
            this.Name = "heart";
            //this.TypeName = "coin";
            this.InitNewComponent<Animator>();
            this.Animator.AddAnimation("default", Database.GetGameObjectMaterial("heart"));
        }
    }
}
```

2.2.2.6 Assets/Models/Miscellaneous/PowerUp.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Scripts;
using System;
using System.Drawing;

namespace MarIO.Assets.Models.Miscellaneous
{
    public class PowerUp : GameObject
    {
        public Mario PlayerReference;

        public enum PowerUpType
        {
            Mushroom,
            Flower,
            Star
        }

        public Action OnPickedUp { get; private set; }
        public PowerUpType Type { get; private set; }

        protected override void Initialize()
        {
            this.Name = nameof(PowerUp);

            this.InitNewComponent<Collider>();
            this.Collider.Area = new RectangleF(0, 0, 16, 16);
            this.Collider.Enabled = false;

            this.InitNewScript<PowerUpScript>();

            switch (Shared.Mechanics.MarioCurrentState)
            {
                case Mario.State.Small:
                    this.TypeName = "mushroom";
                    Type = PowerUpType.Mushroom;
                    OnPickedUp = () =>
                    {
                        Shared.Mechanics.GameScore += Shared.Mechanics.MUSHROOM_SCORE;
                        TextBlock FloatingText = new TextBlock()
                        {
                            Text = string.Format("{0}", Shared.Mechanics.MUSHROOM_SCORE),
                            TextShadow = true
                        };
                        FloatingText.Transform.Position = this.Transform.Position;
                        FloatingText.Transform.Dimensions = new Vector3(20, 6, 0);
                        FloatingText.AddAsFloatingText();
                        PlayerReference.CurrentState = Mario.State.Super;

                        OnPickedUp = null;

                        this.Destroy();
                    };
                    break;

                case Mario.State.Super:
                    this.TypeName = "flower";
                    Type = PowerUpType.Flower;
                    this.InitNewComponent<Animator>();
                    this.Animator.AddAnimation("default", "flower");
                    this.Animator.Play("default");
                    OnPickedUp = () =>
                    {
                        Shared.Mechanics.GameScore += Shared.Mechanics.FLOWER_SCORE;
                        TextBlock FloatingText = new TextBlock()
                        {
                            Text = string.Format("{0}", Shared.Mechanics.FLOWER_SCORE),
                            TextShadow = true
                        };
                        FloatingText.Transform.Position = this.Transform.Position;
                        FloatingText.Transform.Dimensions = new Vector3(20, 6, 0);
                        FloatingText.AddAsFloatingText();

                        PlayerReference.CurrentState = Mario.State.Fire;

                        OnPickedUp = null;

                        this.Destroy();
                    };
                    this.Collider.IsTrigger = true;
                    break;

                case Mario.State.Fire:
                case Mario.State.Invincible:
                    this.TypeName = "1-UP";
                    Type = PowerUpType.Star;
                    this.InitNewComponent<Animator>();
                    this.Animator.AddAnimation("default", "star");
                    this.Animator.Play("default");
                    OnPickedUp = () =>
                    {
                        Shared.Mechanics.GameScore += Shared.Mechanics.STAR_SCORE;
                        TextBlock FloatingText = new TextBlock()
                        {
                            Text = string.Format("{0}", Shared.Mechanics.STAR_SCORE),
                        TextShadow = true
                    };
                    FloatingText.Transform.Position = this.Transform.Position;
                    FloatingText.Transform.Dimensions = new Vector3(20, 6, 0);
                    FloatingText.AddAsFloatingText();

                    PlayerReference.CurrentState = Mario.State.Invincible;
                    Shared.Mechanics.Lives++;

                    OnPickedUp = null;

                    this.Destroy();
                };
                break;

            default:
                throw new Exception("JAK");
        }
    }
}
}
```

2.2.2.7 Assets/Models/AnimatedObject.cs

```
using DKEngine.Core;

namespace MarIO.Assets.Models
{
    public abstract class AnimatedObject : GameObject
    {
        public virtual bool IsDestroyed { get; set; }
        public bool ChangeState = false;

        public AnimatedObject()
            : base()
        { }

        public AnimatedObject(GameObject Parent)
            : base(Parent)
        { }
    }
}
```

2.2.2.8 Assets/Models/BackgroundWorker.cs

```
using DKEngine.Core;
using MarIO.Assets.Scripts;

namespace MarIO.Assets.Models
{
    public class BackgroundWorker : GameObject
    {
        protected override void Initialize()
        {
            this.InitNewScript<BlockAnimatorScript>();
            this.InitNewScript<FloatingCoinAnimatorScript>();
            this.InitNewScript<FloatingTextAnimatorScript>();
            this.InitNewScript<SpecialBlocksUpdateScript>();
            this.InitNewScript<WorldChangeManagerScript>();
        }
    }
}
```

2.2.2.9 Assets/Models/Block.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models.Miscellaneous;
using MarIO.Assets.Scripts;
using System;
using System.Collections.Generic;
using static DKEngine.Core.Components.Transform;

namespace MarIO.Assets.Models
{
    public class Block : AnimatedObject
```

```
{
    public enum BlockType
    {
        Ground1,
        Ground2,
        Ground3,
        Ground4,
        Bridge,
        Bush1,
        Bush2,
        Bush3,
        BushSmall,
        CastleBig,
        CastleSmall,
        Cloud1,
        Cloud2,
        Cloud3,
        Fence,
        Finish,
        Flag,
        FlagPole,
        Mountain,
        NoCoin,
        Sky,
        Water1,
        Water2,
        Pipe1,
        Pipe2,
        Pipe3,
        Pipe4,
        Pipe5,
        UnderGround1,
        UnderGround2,
        UnderGround3,
        UnderGround4,
        UnderGroundBackground1,
        UnderGroundBackground2,
        NumberOfObjects
    }

    public static Dictionary<BlockType, string> BlockTypeNames = new Dictionary<BlockType, string>()
    {
        { BlockType.Bridge, "bridge" },
        { BlockType.Bush1, "bush_01" },
        { BlockType.Bush2, "bush_02" },
        { BlockType.Bush3, "bush_03" },
        { BlockType.BushSmall, "bush_small" },
        { BlockType.CastleBig, "castle_big" },
        { BlockType.CastleSmall, "castle_small" },
        { BlockType.Cloud1, "cloud_01" },
        { BlockType.Cloud2, "cloud_02" },
        { BlockType.Cloud3, "cloud_03" },
        { BlockType.Fence, "fence" },
        { BlockType.Flag, "finish_flag" },
        { BlockType.FlagPole, "flag_pole" },
        { BlockType.Finish, "" },
        { BlockType.Ground1, "block_1_with_coin" },
        { BlockType.Ground2, "block_02" },
        { BlockType.Ground3, "block_03" },
        { BlockType.Ground4, "block_04" },
        { BlockType.Mountain, "mountain" },
        { BlockType.NoCoin, "block_nocoins" },
        { BlockType.Pipe1, "pipe_01" },
        { BlockType.Pipe2, "pipe_02" },
        { BlockType.Pipe3, "pipe_03" },
        { BlockType.Pipe4, "pipe_04" },
        { BlockType.Pipe5, "pipe_05" },
        { BlockType.Sky, "sky" },
        { BlockType.UnderGround1, "underground_block_01" },
        { BlockType.UnderGround2, "underground_block_02" },
        { BlockType.UnderGround3, "underground_block_03" },
        { BlockType.UnderGround4, "underground_block_04" },
        { BlockType.UnderGroundBackground1, "background_01" },
        { BlockType.UnderGroundBackground2, "background_02" },
        { BlockType.Water1, "water_01" },
        { BlockType.Water2, "water_02" },
    };
```

```

        public enum CollisionState
        {
            Stay,
            Up,
            Down
        }

        public BlockType Type { get; set; }
        public bool InitCollider { get; set; }
        public CollisionState State { get; set; }

        public bool SpecialActionActivate
        {
            get { return _specialAction; }
            set
            {
                if (value)
                {
                    Shared.AnimatedWorldReferences.SpecialActions.Push(this);
                }

                _specialAction = value;
            }
        }

        public Action SpecialAction { get; set; }
        public Direction PipeEnterDirection { get; set; }
        public bool CoinGot { get; set; }

        public bool PowerUp
        {
            get { return _powerUp; }
            set
            {
                _powerUp = value;
                if (value)
                    _hadBonus = true;
            }
        }

        public byte CoinCount
        {
            get { return _coinCount; }
            set
            {
                _coinCount = value;
                if (value > 0)
                    _hadBonus = true;
            }
        }

        public bool HadBonus
        {
            get { return _hadBonus; }
        }

        private bool _powerUp = false;
        private byte _coinCount = 0;
        private bool _hadBonus = false;
        private bool _specialAction = false;
        private SoundOutput FX_Player;

        public Block()
            : base()
        { }

        public Block(GameObject Parent)
            : base(Parent)
        { }

        protected override void Initialize()
        {
            this.TypeName = BlockTypeNames[Type];
            if (InitCollider)
                this.InitNewComponent<Collider>();

            switch (Type)
            {
                case BlockType.Finish:
                    {
                        this.Transform.Dimensions = new Vector3(32, 200, 0);

                        Block part1 = new Block(this)
                        {
                            Name = string.Format("{0}_Flag", this.Name),
                            Type = BlockType.Flag
                        };
                        Block part2 = new Block(this)
                        {
                            Name = string.Format("{0}_Pole", this.Name),
                            Type = BlockType.FlagPole
                        };
                        part2.Transform.Position -= new Vector3(16, 0, 0);
                    }
                    break;

                case BlockType.Pipe1:
                    {
                        PipeEnterDirection = Direction.Right;
                        this.InitNewComponent<Collider>();
                        this.Collider.IsTrigger = true;
                        this.Collider.Area = new System.Drawing.RectangleF(-1, 15, 1, 1);

                        this.InitNewScript<PipePort>();

                        Blocker block = new Blocker(this)
                        {
                            Name = string.Format("{0}_Blocker", this.Name)
                        };
                        block.InitNewComponent<Collider>();
                        block.Collider.Area = new System.Drawing.RectangleF(0, 0, this.Transform.Dimensions.X, this.Transform.Dimensions.Y);
                    }
                    break;

                case BlockType.Pipe3:
                    {
                        PipeEnterDirection = Direction.Down;
                        this.InitNewComponent<Collider>();
                        this.Collider.IsTrigger = true;
                        this.Collider.Area = new System.Drawing.RectangleF(15, -1, 1, 1);

                        this.InitNewScript<PipePort>();

                        Blocker block = new Blocker(this)
                        {
                            Name = string.Format("{0}_Blocker", this.Name)
                        };
                        block.InitNewComponent<Collider>();
                        block.Collider.Area = new System.Drawing.RectangleF(0, 0, this.Transform.Dimensions.X, this.Transform.Dimensions.Y);
                    }
                    break;
            }

            if (CoinCount > 0 || PowerUp)
            {
                this.InitNewComponent<Animator>();
                this.Animator.AddAnimation("default", this.TypeName);
                this.Animator.AddAnimation("nocoin", BlockTypeNames[BlockType.NoCoin]);
            }

            FX_Player = GameObject.Find<SoundOutput>(nameof(SoundOutput));
        }

        public void GetContent()
        {
            if (PowerUp)
            {
                GameObject.Instantiate<PowerUp>(new Vector3(this.Transform.Position.X + 4, this.Transform.Position.Y, this.Transform.Position.Z - 1), new Vector3(), new Vector3(1, 1, 1));
                PowerUp = false;
                this.Animator.Play("nocoin");
            }
            else if (CoinCount > 0 && !CoinGot)
            {
                GameObject.Instantiate<Coin>(new Vector3(this.Transform.Position.X + 4, this.Transform.Position.Y,
this.Transform.Position.Z - 1), new Vector3(), new Vector3(1, 1, 1)).AddAsFloatingCoin();
                CoinCount--;
                Shared.Mechanics.GameScore += Shared.Mechanics.COIN_SCORE;
                Shared.Mechanics.FXSoundSource.PlaySound(Coin.COIN_FX);
                CoinGot = true;

                if (CoinCount == 0)
                {
                    this.Animator.Play("nocoin");
                }
            }
        }

        public void DestroyAnim()
        { }
    }
}
```

2.2.2.10 Assets/Models/Blocker.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Models
{
    public class Blocker : GameObject
    {
        public Blocker()
            : base()
        { }

        public Blocker(GameObject Parent)
            : base(Parent)
        { }

        protected override void Initialize()
        {
            this.InitNewComponent<Collider>();
        }
    }
}
```

2.2.2.11 Assets/Models/Delayer.cs

```
using DKEngine.Core;
using MarIO.Assets.Scripts;
using System;

namespace MarIO.Assets.Models
{
    public class Delayer : GameObject
    {
        public TimeSpan TimeToWait;
        public Action CalledAction;

        public Delayer()
        {
            Name = nameof(Delayer);
        }

        protected override void Initialize()
        {
            this.InitNewScript<DelayScript>();
        }
    }
}
```

2.2.2.12 Assets/Models/Enemy.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Scripts;
using System.Collections.Generic;

namespace MarIO.Assets.Models
{
    public abstract class Enemy : AnimatedObject
    {
        public enum EnemyType
        {
            Goomba,
            GoombaBlue,
            GoombaSilver,
            KoopaTroopa,
            KoopaParatroopa,
            PiranhaPlant,
            Spiny,
            BuzzyBeatle,
            BuzzyBeatleBlue,
            BuzzyBeatleSilver,
            FireBar,
            BulletBill,
            BillBlasterLarge,
            BillBlasterSmall
        }

        protected static Dictionary<EnemyType, string> EnemyTypeNames = new Dictionary<EnemyType, string>()
        {
            { EnemyType.Goomba, "goomba" },
            { EnemyType.GoombaBlue, "" },
            { EnemyType.GoombaSilver, "" },
            { EnemyType.KoopaTroopa, "" },
            { EnemyType.KoopaParatroopa, "" },
            { EnemyType.PiranhaPlant, "" },
            { EnemyType.Spiny, "" },
            { EnemyType.BuzzyBeatle, "" },
            { EnemyType.BuzzyBeatleBlue, "" },
            { EnemyType.BuzzyBeatleSilver, "" },
            { EnemyType.FireBar, "" },
            { EnemyType.BulletBill, "" },
            { EnemyType.BillBlasterLarge, "" },
            { EnemyType.BillBlasterSmall, "" }
        };

        public EnemyType Type { get; set; }

        public Enemy()
            : base()
        { }

        public Enemy(GameObject Parent)
            : base(Parent)
        { }
    }

    internal class Goomba : Enemy
    {
        protected override void Initialize()
        {
            this.Name = "Goomba";
            this.Type = EnemyType.Goomba;

            this.InitNewComponent<Collider>();
            this.Collider.Area = new System.Drawing.RectangleF(0, 0, 16, 16);

            this.InitNewScript<GoombaController>();
            this.InitNewComponent<Animator>();
            this.Animator.AddAnimation("default", Database.GetGameObjectMaterial(EnemyTypeNames[Type]));
            this.Animator.AddAnimation("dead", Database.GetGameObjectMaterial(EnemyTypeNames[Type] + "_dead"));
        }
    }
}
```

2.2.2.13 Assets/Models/Group.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Models
```

```
{
    public class Group : GameObject
    {
        public bool InitCollider = false;

        public Group()
            : base()
        { }

        public Group(GameObject Parent)
            : base(Parent)
        { }

        public Block.BlockType Type { get; set; }
        public Vector3 SizeInBlocks { get; set; }

        protected override void Initialize()
        {
            Material tmp = Database.GetGameObjectMaterial(Block.BlockTypeNames[Type]);

            this.Transform.Dimensions = new Vector3(SizeInBlocks.X * tmp.Width, SizeInBlocks.Y * tmp.Height, 0);
            for (int i = 0; i < SizeInBlocks.Y; i++)
            {
                for (int j = 0; j < SizeInBlocks.X; j++)
                {
                    Block newBlock = new Block(this);

                    newBlock.Type = Type;
                    newBlock.Transform.Position += new Vector3(j * tmp.Width * this.Transform.Scale.X, i * tmp.Height *
this.Transform.Scale.Y, this.Transform.Position.Z);
                    newBlock.Name = string.Format("{0}_{1}_{2}", Name, j, i);
                }
            }

            if (InitCollider)
                this.InitNewComponent<Collider>();
        }
    }
}
```

2.2.2.14 Assets/Models/GUIUpdater.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models.Miscellaneous;
using MarIO.Assets.Scripts;

namespace MarIO.Assets.Models
{
    public class GUIUpdater : GameObject
    {
        protected override void Initialize()
        {
            Name = "GUI";

            this.IsGUI = true;

            /*------------ TIME TEXT ----------------*/

            #region TIME

            TextBlock _time = new TextBlock(this)
            {
                IsGUI = true,
                TextShadow = true,
                Text = "TIME",
                FontSize = 2
            };
            _time.Transform.Dimensions = new Vector3(100, 20, 1);
            _time.Transform.Position += new Vector3(16, 4, 128);

            TextBlock Time = new TextBlock(this)
            {
                Name = "txt_Time",
                IsGUI = true,
                TextShadow = true,
                Text = "",
                FontSize = 2
            };
            Time.Transform.Dimensions = new Vector3(100, 20, 1);
            Time.Transform.Position += new Vector3(22, 16, 128);

            #endregion TIME

            /*------------ SCORE TEXT ----------------*/

            #region SCORE

            TextBlock Score = new TextBlock(this)
            {
                Name = "txt_Score",
                Text = "",
                IsGUI = true,
                TextShadow = true,
                FontSize = 2,
                HAlignment = Text.HorizontalAlignment.Right,
                TextHAlignment = Text.HorizontalAlignment.Right
            };
            Score.Transform.Dimensions = new Vector3(100, 20, 1);
            Score.Transform.Position += new Vector3(-16, 4, 128);

            #endregion SCORE

            /*------------ COINS TEXT ----------------*/

            #region COINS

            Coin UICoin = new Coin(this)
            {
                HasShadow = true
            };
            UICoin.Transform.Position += new Vector3(75, 4, 128);

            TextBlock _coins = new TextBlock(this)
            {
                Name = "txt_Coins",
                Text = "",
                IsGUI = true,
                TextShadow = true,
                FontSize = 1.5f
            };
            _coins.Transform.Dimensions = new Vector3(100, 20, 1);
            _coins.Transform.Position += new Vector3(85, 4, 128);

            #endregion COINS

            /*------------ LIVES TEXT ----------------*/

            #region LIVES

            Heart UIHeart = new Heart(this)
            {
                HasShadow = true
            };
            UIHeart.Transform.Position += new Vector3(73, 16, 128);

            TextBlock _lives = new TextBlock(this)
            {
                Name = "txt_Lives",
                Text = "",
                IsGUI = true,
                TextShadow = true,
                FontSize = 1.5f
            };
            _lives.Transform.Dimensions = new Vector3(100, 20, 1);
            _lives.Transform.Position += new Vector3(85, 18, 128);

            #endregion LIVES

            /*------------ WORLD TEXT ----------------*/

            #region WORLD

```

```
            TextBlock _world = new TextBlock(this)
            {
                Text = "WORLD",
                IsGUI = true,
                TextShadow = true,
                FontSize = 2,
                HAlignment = Text.HorizontalAlignment.Right,
                TextHAlignment = Text.HorizontalAlignment.Center
            };
            _world.Transform.Dimensions = new Vector3(50, 20, 1);
            _world.Transform.Position += new Vector3(-90, 4, 128);

            TextBlock World = new TextBlock(this)
            {
                Name = "txt_World",
                Text = "",
                IsGUI = true,
                TextShadow = true,
                FontSize = 2,
                HAlignment = Text.HorizontalAlignment.Right,
                TextHAlignment = Text.HorizontalAlignment.Center
            };
            World.Transform.Dimensions = new Vector3(50, 20, 1);
            World.Transform.Position += new Vector3(-90, 16, 128);

            #endregion WORLD

            this.InitNewScript<GUIUpdateScript>();
        }
    }
}
```

2.2.2.15 Assets/Models/Mario.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Scripts;
using System.Drawing;
using static DKEngine.Core.Components.Transform;
using static MarIO.Shared.Assets.Animations;

namespace MarIO.Assets.Models
{
    public class Mario : AnimatedObject
    {
        private State _currentState;
        private bool _isDestroyed;

        public override bool IsDestroyed
        {
            get { return _isDestroyed; }
            set
            {
                _isDestroyed = value;
                if (value)
                    CurrentState = State.Dead;
            }
        }

        public bool KilledEnemy = false;
        public Trigger LeftTrigger { get; private set; }
        public Trigger RightTrigger { get; private set; }
        public Trigger TopTrigger { get; private set; }
        public Trigger BottomTrigger { get; private set; }

        public bool InitCharacterController { get; set; }
        public bool InitCameraController { get; set; }
        public bool InitCollider { get; set; }

        public State CurrentState
        {
            get { return _currentState; }
            set
            {
                _currentState = value;
                Shared.Mechanics.MarioCurrentState = value;

                Vector3 tmp = this.Transform.Position;

                switch (value)
                {
                    case State.Dead:
                    case State.Small:
                        this.Collider.Area = new RectangleF(2, 0, 12, 16);

                        TopTrigger.Transform.Position = tmp.Add(2.5f, -1, 0);//new Vector3(tmp.X + 2.5f, tmp.Y - 1, tmp.Z);
                        TopTrigger.Transform.Dimensions = new Vector3(11, 1, 0);

                        RightTrigger.Transform.Position = tmp.Add(14, 0, 0); //new Vector3(tmp.X + 14, tmp.Y, tmp.Z);
                        RightTrigger.Transform.Dimensions = new Vector3(1, 14, 0);

                        LeftTrigger.Transform.Position = tmp.Add(1, 0, 0); //new Vector3(tmp.X + 1, tmp.Y, tmp.Z);
                        LeftTrigger.Transform.Dimensions = new Vector3(1, 14, 0);

                        BottomTrigger.Transform.Position = tmp.Add(1, 16, 0); //new Vector3(tmp.X + 1, tmp.Y + 16, tmp.Z);
                        BottomTrigger.Transform.Dimensions = new Vector3(14, 2, 0);

                        TopTrigger.Collider.Area = new RectangleF(0, 0, 11, 1);
                        RightTrigger.Collider.Area = new RectangleF(0, 0, 1, 14);
                        LeftTrigger.Collider.Area = new RectangleF(0, 0, 1, 14);
                        BottomTrigger.Collider.Area = new RectangleF(0, 0, 14, 2);

                        break;

                    case State.Super:
                    case State.Fire:
                    case State.Invincible:
                        this.Collider.Area = new RectangleF(0, 0, 16, 32);

                        TopTrigger.Transform.Position = tmp.Add(0.5f, -1, 0); //new Vector3(tmp.X + 0.5f, tmp.Y - 1, tmp.Z + 0);
                        TopTrigger.Transform.Dimensions = new Vector3(15, 1, 0);

                        RightTrigger.Transform.Position = tmp.Add(16, 0, 0); //new Vector3(tmp.X + 16, tmp.Y + 0, tmp.Z + 0);
                        RightTrigger.Transform.Dimensions = new Vector3(1, 30, 0);

                        LeftTrigger.Transform.Position = tmp.Add(-1, 0, 0); //new Vector3(tmp.X - 1, tmp.Y + 0, tmp.Z + 0);
                        LeftTrigger.Transform.Dimensions = new Vector3(1, 30, 0);

                        BottomTrigger.Transform.Position = tmp.Add(0, 32, 0); //new Vector3(tmp.X + 0, tmp.Y + 32, tmp.Z + 0);
                        BottomTrigger.Transform.Dimensions = new Vector3(16, 2, 0);

                        TopTrigger.Collider.Area = new RectangleF(0, 0, 15, 1);
                        RightTrigger.Collider.Area = new RectangleF(0, 0, 1, 30);
                        LeftTrigger.Collider.Area = new RectangleF(0, 0, 1, 30);
                        BottomTrigger.Collider.Area = new RectangleF(0, 0, 16, 2);

                        break;

                    default:
                        break;
                }

#if DEBUG
                TopTrigger.Model = new Material(Color.Black, TopTrigger);
                RightTrigger.Model = new Material(Color.Black, RightTrigger);
                LeftTrigger.Model = new Material(Color.Black, LeftTrigger);
                BottomTrigger.Model = new Material(Color.Black, BottomTrigger);
#endif
            }
        }

        public Movement CurrentMovement { get; set; }
        public Direction PipeEnteredInDirection { get { return EnteredPipe.PipeEnterDirection; } }
        public Block EnteredPipe { get; set; }

        public WorldChangeManagerScript WorldManager { get; set; }

        public Mario()
        {
            InitTriggers();
        }

        public Mario(GameObject Parent)
            : base(Parent)
        {
            InitTriggers();
        }

        public enum State
        {
            Dead,
            Small,
            Super,
            Fire,
            Invincible
        }

        public enum Movement
        {
            Standing,
            Crouching
        }

        protected override void Initialize()
        {
            this.Name = "Player";

            this.InitNewComponent<Animator>();
            this.Animator.AddAnimation(MARIO_IDLE_LEFT, MARIO_IDLE_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_IDLE_RIGHT, MARIO_IDLE_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_JUMP_LEFT, MARIO_JUMP_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_JUMP_RIGHT, MARIO_JUMP_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_MOVE_LEFT, MARIO_MOVE_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_MOVE_RIGHT, MARIO_MOVE_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_DEAD, MARIO_DEAD_MAT);
            this.Animator.AddAnimation(MARIO_CROUCHING_LEFT, MARIO_CROUCHING_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_CROUCHING_RIGHT, MARIO_CROUCHING_RIGHT_MAT);

            this.Animator.AddAnimation(MARIO_SUPER_IDLE_LEFT, MARIO_SUPER_IDLE_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_IDLE_RIGHT, MARIO_SUPER_IDLE_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_JUMP_LEFT, MARIO_SUPER_JUMP_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_JUMP_RIGHT, MARIO_SUPER_JUMP_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_MOVE_LEFT, MARIO_SUPER_MOVE_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_MOVE_RIGHT, MARIO_SUPER_MOVE_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_POWERUP_LEFT, MARIO_SUPER_POWERUP_LEFT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_POWERUP_RIGHT, MARIO_SUPER_POWERUP_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_CROUCHING_RIGHT, MARIO_SUPER_CROUCHING_RIGHT_MAT);
            this.Animator.AddAnimation(MARIO_SUPER_CROUCHING_LEFT, MARIO_SUPER_CROUCHING_LEFT_MAT);

            /*this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT, Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_IDLE_RIGHT, Shared.Assets.Animations.MARIO_FIRE_IDLE_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_JUMP_LEFT, Shared.Assets.Animations.MARIO_FIRE_JUMP_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_JUMP_RIGHT, Shared.Assets.Animations.MARIO_FIRE_JUMP_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_MOVE_LEFT, Shared.Assets.Animations.MARIO_FIRE_MOVE_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT, Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_POWERUP_LEFT, Shared.Assets.Animations.MARIO_FIRE_POWERUP_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_POWERUP_RIGHT, Shared.Assets.Animations.MARIO_FIRE_POWERUP_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_CROUCHING_RIGHT, Shared.Assets.Animations.MARIO_FIRE_CROUCHING_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_CROUCHING_LEFT, Shared.Assets.Animations.MARIO_FIRE_CROUCHING_LEFT_MAT);*/

            /*this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_IDLE_LEFT, Shared.Assets.Animations.MARIO_IDLE_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_IDLE_RIGHT, Shared.Assets.Animations.MARIO_IDLE_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_JUMP_LEFT, Shared.Assets.Animations.MARIO_JUMP_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_JUMP_RIGHT, Shared.Assets.Animations.MARIO_JUMP_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_MOVE_LEFT, Shared.Assets.Animations.MA-
RIO_MOVE_LEFT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_MOVE_RIGHT, Shared.Assets.Animations.MA-
RIO_MOVE_RIGHT_MAT);
            this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_DEAD, Shared.Assets.Animations.MA-
RIO_DEAD_MAT);*/

            if (InitCharacterController)
                this.InitNewScript<CharacterController>();

            if (InitCameraController)
                this.InitNewScript<CameraController>();

            if (InitCollider)
            {
                this.InitNewComponent<Collider>();
            }

            BottomTrigger.InitNewScript<BottomMarioChecker>();
            LeftTrigger.InitNewScript<LeftMarioChecker>();
            RightTrigger.InitNewScript<RightMarioChecker>();
            TopTrigger.InitNewScript<TopMarioChecker>();

            CurrentState = Shared.Mechanics.MarioCurrentState;

            WorldManager = Behavior.Find<WorldChangeManagerScript>("worldManager");
        }

        private void InitTriggers()
        {
            BottomTrigger = new Trigger(this)
            {
                Name = "Bottom_Trigger"
            };
            LeftTrigger = new Trigger(this)
            {
                Name = "Left_Trigger"
            };
            TopTrigger = new Trigger(this)
            {
                Name = "Top_Trigger"
            };
            RightTrigger = new Trigger(this)
            {
                Name = "Right_Trigger"
            };
        }

        public void PipeEnter(Block Pipe)
        {
            ChangeState = true;
            EnteredPipe = Pipe;
        }
    }
}
```

2.2.2.16 Assets/Models/MusicPlayer.cs

```
using DKEngine.Core;
using MarIO.Assets.Scripts;

namespace MarIO.Assets.Models
{
    public class MusicPlayer : GameObject
    {
        protected override void Initialize()
        {
            this.Name = "MusicPlayer";
            this.InitNewScript<MusicScript>();
        }
    }
}
```

2.2.2.17 Assets/Models/SoundOutput.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
```

```

namespace MarIO.Assets.Models
{
    public class SoundOutput : GameObject
    {
        protected override void Initialize()
        {
            this.Name = nameof(SoundOutput);
            this.InitNewComponent<SoundSource>();
            Shared.Mechanics.FXPlayer = this;
        }
    }
}
```

2.2.2.18 Assets/Models/Trigger.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Models
{
    public class Trigger : GameObject
    {
        public Trigger()
            : base()
        { }

        public Trigger(GameObject Parent)
            : base(Parent)
        { }

        protected override void Initialize()
        {
            this.InitNewComponent<Collider>();
            this.Collider.IsTrigger = true;
        }
    }
}
```

2.2.2.19 Assets/Scenes/About.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MarIO.Assets.Scenes
{
    class About : Scene
    {
        public override void Init()
        {
            new Camera()
            {
                BackGround = Shared.Mechanics.OverworldBackground.ToColor()
            };

            new Group()
            {
                SizeInBlocks = new Vector3(1, 20, 0),
                Type = Block.BlockType.Ground2,
                InitCollider = true
            }.Transform.Position = new Vector3(0, 0, 0);

            new Group()
            {
                SizeInBlocks = new Vector3(1, 20, 0),
                Type = Block.BlockType.Ground2,
                InitCollider = true
            }.Transform.Position = new Vector3(48, 0, 0);
```

```

new Group()
{
    SizeInBlocks = new Vector3(2, 1, 0),
    Type = Block.BlockType.Ground2,
    InitCollider = true
}.Transform.Position = new Vector3(16, 224, 0);

new Block()
{
    InitCollider = true,
    Type = Block.BlockType.Pipe3,
    SpecialAction = GoBack
}.Transform.Position = new Vector3(16, 192, 1);

new Mario()
{
    InitCollider = true,
    InitCharacterController = true
}.Transform.Position = new Vector3(16, 80, 0);

var _Mario = new TextBlock()
{
    Foreground = Color.LawnGreen,
    FontSize = 6,
    HAlignment = Text.HorizontalAlignment.Center,
    IsGUI = true,
    Text = "MARIO",
    TextShadow = true,
    TextHAlignment = Text.HorizontalAlignment.Center
};
_Mario.Transform.Position += new Vector3(30, 20, 0);
_Mario.Transform.Dimensions = new Vector3(200, 30, 0);

var _author = new TextBlock()
{
    FontSize = 2,
    HAlignment = Text.HorizontalAlignment.Center,
    IsGUI = true,
    Text = "BY DAVID KNIERADL 2017",
    TextShadow = true,
    TextHAlignment = Text.HorizontalAlignment.Center
};
_author.Transform.Position += new Vector3(30, 80, 0);
_author.Transform.Dimensions = new Vector3(200, 30, 0);

var _using = new TextBlock()
{
    Foreground = Color.YellowGreen,
    FontSize = 3,
    HAlignment = Text.HorizontalAlignment.Center,
    IsGUI = true,
    Text = "Made with",
    TextShadow = true,
    TextHAlignment = Text.HorizontalAlignment.Center
};
_using.Transform.Position += new Vector3(30, 110, 0);
_using.Transform.Dimensions = new Vector3(200, 30, 0);

var _dkengine = new TextBlock()
{
    FontSize = 2,
    HAlignment = Text.HorizontalAlignment.Center,
    IsGUI = true,
    Text = "DKENGINE",
    TextShadow = true,
    TextHAlignment = Text.HorizontalAlignment.Center
};
_dkengine.Transform.Position += new Vector3(30, 140, 0);
_dkengine.Transform.Dimensions = new Vector3(200, 30, 0);

var _naudio = new TextBlock()
{
    FontSize = 2,
    HAlignment = Text.HorizontalAlignment.Center,
    IsGUI = true,
    Text = "NAUDIO",
                TextShadow = true,
                TextHAlignment = Text.HorizontalAlignment.Center
            };
            _naudio.Transform.Position += new Vector3(30, 155, 0);
            _naudio.Transform.Dimensions = new Vector3(200, 30, 0);

            var _ver = new TextBlock()
            {
                FontSize = 1,
                HAlignment = Text.HorizontalAlignment.Center,
                IsGUI = true,
                Text = "version 0.0.1 alpha",
                TextShadow = true,
                TextHAlignment = Text.HorizontalAlignment.Center
            };
            _ver.Transform.Position += new Vector3(30, 190, 0);
            _ver.Transform.Dimensions = new Vector3(200, 30, 0);

            new MusicPlayer();
            new SoundOutput();
            new BackgroundWorker();
        }

        public override void Unload()
        { }

        private void GoBack()
        {
            Engine.ChangeScene(nameof(MainMenu), true);
        }
    }
}
```

2.2.2.20 Assets/Scenes/GameOver.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using MarIO.Assets.Models.Miscellaneous;
using System;
using System.Drawing;

namespace MarIO.Assets.Scenes
{
    internal class GameOver : Scene
    {
        public GameOver()
        {
            Name = nameof(GameOver);
        }

        public override void Init()
        {
            TextBlock GameOver = new TextBlock()
            {
                FontSize = 5,
                Foreground = Color.White,
                HAlignment = Text.HorizontalAlignment.Center,
                IsGUI = true,
                Name = "tx_GameOver",
                Text = "GAME OVER",
                TextHAlignment = Text.HorizontalAlignment.Center,
                VAlignment = Text.VerticalAlignment.Center,
            };
            GameOver.Transform.Dimensions = new Vector3(200, 30, 0);
            GameOver.Transform.Position += new Vector3(0, -30, 0);

            TextBlock Score = new TextBlock()
            {
                FontSize = 2.5f,
                Foreground = Color.White,
                HAlignment = Text.HorizontalAlignment.Center,
                IsGUI = true,
                Name = "tx_Score",
                Text = Shared.Mechanics.GameScoreStr,
                TextHAlignment = Text.HorizontalAlignment.Center,
                VAlignment = Text.VerticalAlignment.Center
            };
            Score.Transform.Dimensions = new Vector3(100, 30, 0);
            Score.Transform.Position += new Vector3(0, 30, 0);

            GameObject holder = new GameObject();
            holder.Transform.Position = new Vector3(136, 156, 0);

            Coin CoinIcon = new Coin(holder)
            {
                IsGUI = true,
                Name = "coin_icon"
            };
            CoinIcon.Transform.Scale = new Vector3(2f, 2f, 0);

            TextBlock Coins = new TextBlock(holder)
            {
                FontSize = 2.5f,
                IsGUI = true,
                TextHAlignment = Text.HorizontalAlignment.Center,
                Text = string.Format($"*{Shared.Mechanics.CoinsCount:00}")
            };
            Coins.Transform.Dimensions = new Vector3(40, 15, 0);
            Coins.Transform.Position += new Vector3(12, 2, 0);

            new Delayer()
            {
                CalledAction = () => Engine.LoadScene<MainMenu>(),
                TimeToWait = new TimeSpan(0, 0, 5)
            };

            new Camera()
            {
                BackGround = Shared.Mechanics.WorldChangeBackground.ToColor()
            };

            Shared.Mechanics.MarioCurrentState = Mario.State.Super;
        }

        public override void Unload()
        { }
    }
}
```

2.2.2.21 Assets/Scenes/Level_1_1.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using MarIO.Assets.Scripts;

namespace MarIO.Assets.Scenes
{
    public class Level_1_1 : MapBase
    {
        private const int offset = 520;

        public Level_1_1()
        {
            Name = MapBase.LevelsNames[nameof(Level_1_1)];
            Shared.Mechanics.LastWorldType = typeof(Level_1_1);
        }

        public override void Load()
        {
            /*-------------- BG PRESET ----------------*/
            for (int i = 0; i < 8; i++)
            {
                new Block()
                {
                    Name = $"cloud_1_{i}",
                    Type = Block.BlockType.Cloud1,
                }.Transform.Position = new Vector3(50 + i * offset, -82, -2);

                new Block()
                {
```

```
            Name = $"cloud_2_{i}",
            Type = Block.BlockType.Cloud1,
        }.Transform.Position = new Vector3(160 + i * offset, -122, -2);

        new Block()
        {
            Name = $"cloud_3_{i}",
            Type = Block.BlockType.Cloud3,
        }.Transform.Position = new Vector3(260 + i * offset, -70, -2);

        new Block()
        {
            Name = $"cloud_4_{i}",
            Type = Block.BlockType.Cloud2,
        }.Transform.Position = new Vector3(370 + i * offset, -103, -2);

        new Block()
        {
            Name = $"mountain_1_{i}",
            Type = Block.BlockType.Mountain
        }.Transform.Position = new Vector3(120 + i * offset, 16, -2);

        new Block()
        {
            Name = $"mountain_2_{i}",
            Type = Block.BlockType.Mountain
        }.Transform.Position = new Vector3(250 + i * offset, 16, -2);

        new Block()
        {
            Name = $"bush_1_{i}",
            Type = Block.BlockType.Bush1
        }.Transform.Position = new Vector3(5 + i * offset, 24, -1);

        new Block()
        {
            Name = $"bush_2_{i}",
            Type = Block.BlockType.Bush3
        }.Transform.Position = new Vector3(210 + i * offset, 24, -1);

        new Block()
        {
            Name = $"bush_3_{i}",
            Type = Block.BlockType.Bush2
        }.Transform.Position = new Vector3(375 + i * offset, 24, -1);
    }

    #region GROUND

    Group _1 = new Group()
    {
        Name = "ground1",
        InitCollider = true,
        Type = Block.BlockType.Ground2
    };
    _1.SizeInBlocks = new Vector3(64, 3, 0);
    _1.Transform.Position = new Vector3(0, 48, 0);

    Group _2 = new Group()
    {
        Name = "ground2",
        InitCollider = true,
        Type = Block.BlockType.Ground2
    };
    _2.SizeInBlocks = new Vector3(20, 3, 0);
    _2.Transform.Position = new Vector3(1056, 48, 0);

    Group _3 = new Group()
    {
        Name = "ground3",
        InitCollider = true,
        Type = Block.BlockType.Ground2
    };
    _3.SizeInBlocks = new Vector3(68, 3, 0);
    _3.Transform.Position = new Vector3(1424, 48, 0);

    Group _4 = new Group()
```

```
{
    Name = "ground4",
    InitCollider = true,
    Type = Block.BlockType.Ground2
};
_4.SizeInBlocks = new Vector3(100, 3, 0);
_4.Transform.Position = new Vector3(2544, 48, 0);

#endregion GROUND

#region Platform1

new Block()
{
    Name = "bonus_1",
    Type = Block.BlockType.Ground1,
    CoinCount = 1,
    InitCollider = true
}.Transform.Position = new Vector3(320, -12, 0);

new Block()
{
    Name = "platform_1",
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(400, -12, 0);

new Block()
{
    Name = "platform_1",
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 0,
    PowerUp = true
}.Transform.Position = new Vector3(416, -12, 0);

new Block()
{
    Name = "platform_1",
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(432, -12, 0);

new Block()
{
    Name = "platform_1",
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(432, -76, 0);

new Block()
{
    Name = "platform_1",
    CoinCount = 1,
    Type = Block.BlockType.Ground1,
    InitCollider = true
}.Transform.Position = new Vector3(448, -12, 0);

new Block()
{
    Name = "platform_1",
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(464, -12, 0);

#endregion Platform1

new Block()
{
    Name = "pipe",
    Type = Block.BlockType.Pipe3
}.Transform.Position = new Vector3(544, 16, 1);

new Goomba().Transform.Position = new Vector3(600, 18, 0);

{
    GameObject holder = new GameObject();
    holder.Transform.Dimensions = new Vector3(32, 64, 0);
    holder.Transform.Position = new Vector3(700, 32, 0);
    holder.InitNewComponent<Collider>();

    new Block(holder)
    {
        Name = "pipe",
        Type = Block.BlockType.Pipe4
    }.Transform.Position += new Vector3(0, 0, -1);

    new Block(holder)
    {
        Name = "pipe",
        Type = Block.BlockType.Pipe3
    }.Transform.Position += new Vector3(0, -32, 1);
}

new Goomba().Transform.Position = new Vector3(760, 18, 0);
new Goomba().Transform.Position = new Vector3(800, 18, 0);

{
    GameObject holder = new GameObject();
    holder.Transform.Dimensions = new Vector3(32, 64, 0);
    holder.Transform.Position = new Vector3(860, 16, 0);
    holder.InitNewComponent<Collider>();

    new Block(holder)
    {
        Name = "pipe",
        Type = Block.BlockType.Pipe4
    }.Transform.Position += new Vector3(0, 0, -1);

    new Block(holder)
    {
        Name = "pipe",
        Type = Block.BlockType.Pipe3
    }.Transform.Position += new Vector3(0, -32, 1);
}

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(1184, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    CoinCount = 3,
    InitCollider = true
}.Transform.Position = new Vector3(1200, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(1216, -12, 0);

for (int i = 0; i < 10; i++)
{
    new Block()
    {
        Type = Block.BlockType.Ground4,
        InitCollider = true
    }.Transform.Position = new Vector3(1232 + i * 16, -76, 0);
}

for (int i = 0; i < 3; i++)
{
    new Block()
    {
        Type = Block.BlockType.Ground4,
        InitCollider = true
    }.Transform.Position = new Vector3(1440 + i * 16, -76, 0);
}

```

```
new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(1488, -76, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true,
    CoinCount = 5
}.Transform.Position = new Vector3(1488, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true,
    CoinCount = 5
}.Transform.Position = new Vector3(1616, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(1632, -12, 0);

#region Bonus Field

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(1680, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(1744, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    PowerUp = true
}.Transform.Position = new Vector3(1744, -76, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(1808, -12, 0);

#endregion Bonus Field

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(1968, -12, 0);

for (int i = 0; i < 3; i++)
{
    new Block()
    {
        Type = Block.BlockType.Ground4,
        InitCollider = true
    }.Transform.Position = new Vector3(2000 + i * 16, -76, 0);
}

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2080, -76, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(2096, -76, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(2112, -76, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2096, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2112, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2128, -12, 0);

#region Stairs1

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(4, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2192, 32, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(3, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2208, 16, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(2, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2224, 0, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(1, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2240, -16, 0);

#endregion Stairs1

#region Stairs2

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(4, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2288, 32, 0);

```

```
new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(3, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2288, 16, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(2, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2288, 0, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(1, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2288, -16, 0);

#endregion Stairs2

#region Stairs3

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(5, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2432, 32, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(4, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2448, 16, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(3, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2464, 0, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(2, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2480, -16, 0);

#endregion Stairs3

#region Stairs4

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(4, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2544, 32, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(3, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2544, 16, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(2, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2544, 0, 0);
```

```

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(1, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2544, -16, 0);

#endregion Stairs4

new Block()
{
    Type = Block.BlockType.Pipe3
}.Transform.Position = new Vector3(2704, 16, 1);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2768, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2784, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground1,
    InitCollider = true,
    CoinCount = 1
}.Transform.Position = new Vector3(2800, -12, 0);

new Block()
{
    Type = Block.BlockType.Ground4,
    InitCollider = true
}.Transform.Position = new Vector3(2816, -12, 0);

new Block()
{
    Type = Block.BlockType.Pipe3
}.Transform.Position = new Vector3(2928, 16, 1);

#region Stairs5

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(7, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2960, 32, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(6, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2976, 16, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(5, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(2992, 0, 0);

new Group()
{
    InitCollider = true,
    SizeInBlocks = new Vector3(4, 1, 0),
    Type = Block.BlockType.Ground3
}.Transform.Position = new Vector3(3008, -16, 0);

new Group()
{
```

```
                InitCollider = true,
                SizeInBlocks = new Vector3(3, 1, 0),
                Type = Block.BlockType.Ground3
            }.Transform.Position = new Vector3(3024, -32, 0);

            new Group()
            {
                InitCollider = true,
                SizeInBlocks = new Vector3(2, 1, 0),
                Type = Block.BlockType.Ground3
            }.Transform.Position = new Vector3(3040, -48, 0);

            new Group()
            {
                InitCollider = true,
                SizeInBlocks = new Vector3(1, 1, 0),
                Type = Block.BlockType.Ground3
            }.Transform.Position = new Vector3(3056, -64, 0);

            #endregion Stairs5

            new Block()
            {
                Type = Block.BlockType.CastleBig
            }.Transform.Position = new Vector3(3216, -152, -1);

            Trigger EndOfWorld = new Trigger();
            EndOfWorld.Transform.Position = new Vector3(3216, -40, 0);
            EndOfWorld.Transform.Dimensions = new Vector3(200, 80, 0);
            EndOfWorld.InitNewScript<WorldEnd>();

            Mario m = new Mario()
            {
                InitCameraController = true,
                InitCharacterController = true,
                InitCollider = true
            };
            m.Transform.Position = new Vector3(10, -10, 0);

            Camera c = new Camera()
            {
                BackGround = Shared.Mechanics.OverworldBackground.ToColor()
            };

            new SoundOutput();
            new GUIUpdater();
            new BackgroundWorker();
            new MusicPlayer();

            Trigger DeathZone = new Trigger();
            DeathZone.InitNewScript<DeathZoneScript>();
            DeathZone.Transform.Dimensions = new Vector3(5000, 10, 0);
            DeathZone.Transform.Position = new Vector3(0, 100, 0);
        }
    }
}
```

2.2.2.22 Assets/Scenes/MainMenu.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using MarIO.Assets.Scripts;
using System;
using System.Drawing;

namespace MarIO.Assets.Scenes
{
    public class MainMenu : Scene
    {
        public MainMenu()
        {
            Name = nameof(MainMenu);
        }

```

```
public override void Init()
{
    Group wall4 = new Group()
    {
        InitCollider = true,
        Name = "Wall_4",
        SizeInBlocks = new Vector3(21, 2, 0),
        Type = Block.BlockType.Ground2
    };
    wall4.Transform.Position = new Vector3(0, 16 * 13, 0);

    Group wall5 = new Group()
    {
        InitCollider = true,
        Name = "Wall_5",
        SizeInBlocks = new Vector3(22, 1, 0),
        Type = Block.BlockType.Ground2
    };
    wall5.Transform.Position = new Vector3(0, 16 * 9, 0);

    Block pipe1 = new Block()
    {
        Name = "Pipe_1_Play",
        Type = Block.BlockType.Pipe3
    };
    pipe1.Transform.Position = new Vector3(32, 16 * 7, 1);
    pipe1.SpecialAction = Play;

    Block pipe2 = new Block()
    {
        Name = "Pipe_2_About",
        Type = Block.BlockType.Pipe3
    };
    pipe2.Transform.Position = new Vector3(143, 16 * 7, 1);

    Block pipe3 = new Block()
    {
        Name = "Pipe_3_Exit",
        Type = Block.BlockType.Pipe3
    };
    pipe3.Transform.Position = new Vector3(256, 16 * 7, 1);
    pipe3.SpecialAction = Exit;

    Camera baseCam = new Camera()
    {
        BackGround = Shared.Mechanics.OverworldBackground.ToColor()
    };

    Mario player = new Mario()
    {
        InitCharacterController = true,
        InitCollider = true
    };

    TextBlock MainMenuHeader = new TextBlock()
    {
        FontSize = 6,
        HAlignment = Text.HorizontalAlignment.Center,
        Name = "tx_MainMenuHeader",
        Text = "MARIO",
        TextHAlignment = Text.HorizontalAlignment.Center,
        TextShadow = true
    };
    MainMenuHeader.Transform.Position += new Vector3(0, 10, 0);
    MainMenuHeader.Transform.Dimensions = new Vector3(200, 50, 0);

    TextBlock PlayText = new TextBlock()
    {
        Name = "tx_Play",
        Text = "Play",
        TextHAlignment = Text.HorizontalAlignment.Center,
        TextShadow = true
    };
    PlayText.Transform.Position = new Vector3(9, 96, -1);
    PlayText.Transform.Dimensions = new Vector3(80, 20, 0);

    TextBlock OptionsText = new TextBlock()
```

```
{
    Name = "tx_Options",
    Text = "About",
    TextHAlignment = Text.HorizontalAlignment.Center,
    TextShadow = true,
    HAlignment = Text.HorizontalAlignment.Center
};
OptionsText.Transform.Position += new Vector3(0, 96, -1);
OptionsText.Transform.Dimensions = new Vector3(80, 20, 0);

TextBlock ExitText = new TextBlock()
{
    Name = "tx_Exit",
    Text = "Exit",
    TextHAlignment = Text.HorizontalAlignment.Center,
    TextShadow = true,
    HAlignment = Text.HorizontalAlignment.Right
};
ExitText.Transform.Position += new Vector3(-8, 96, -1);
ExitText.Transform.Dimensions = new Vector3(80, 20, 0);

Block cloud1 = new Block()
{
    Name = "cloud_1",
    Type = Block.BlockType.Cloud3
};
cloud1.Transform.Position = new Vector3(-10, 20, -1);

Block cloud2 = new Block()
{
    Name = "cloud_2",
    Type = Block.BlockType.Cloud1
};
cloud2.Transform.Position = new Vector3(120, -15, -1);

Block cloud3 = new Block()
{
    Name = "cloud_3",
    Type = Block.BlockType.Cloud2
};
cloud3.Transform.Position = new Vector3(180, 34, -1);

Block mountain = new Block()
{
    Name = "mountain",
    Type = Block.BlockType.Mountain
};
mountain.Transform.Position = new Vector3(100, 152, -1);
mountain.Transform.Scale = new Vector3(2, 2, 0);

Block bush1 = new Block()
{
    Name = "bush_1",
    Type = Block.BlockType.Bush3
};
bush1.Transform.Position = new Vector3(180, 182, -1);

Block bush2 = new Block()
{
    Name = "bush_2",
    Type = Block.BlockType.Bush2
};
bush2.Transform.Position = new Vector3(25, 182, -1);

Block fence1 = new Block()
{
    Name = "fence_1",
    Type = Block.BlockType.Fence
};
fence1.Transform.Position = new Vector3(90, 192, -1);

Block fence2 = new Block()
{
    Name = "fence_2",
    Type = Block.BlockType.Fence
};
fence2.Transform.Position = new Vector3(106, 192, -1);
```

```

            Block fence3 = new Block()
            {
                Name = "fence_3",
                Type = Block.BlockType.Fence
            };
            fence3.Transform.Position = new Vector3(122, 192, -1);

            Blocker leftSide = new Blocker()
            {
                Name = "LeftSideBlocker"
            };
            leftSide.Transform.Position = new Vector3(-10, -20, 0);
            leftSide.Transform.Dimensions = new Vector3(10, 148, 0);

            Blocker rightSide = new Blocker()
            {
                Name = "LeftSideBlocker"
            };
            rightSide.Transform.Position = new Vector3(320, -20, 0);
            rightSide.Transform.Dimensions = new Vector3(10, 148, 0);

            BackgroundWorker BW = new BackgroundWorker();
            BW.InitNewComponent<Collider>();
            BW.Collider.Area = new RectangleF(-10, 160, 10, 30);
            BW.Collider.IsTrigger = true;
            BW.InitNewScript<MainMenuSpawnScript>();

            new MusicPlayer();
            new SoundOutput();
        }

        public override void Set(params object[] Args)
        { }

        public override void Unload()
        { }

        private void Exit()
        {
            Environment.Exit(1);
        }

        private void Play()
        {
            Shared.Mechanics.MarioCurrentState = Mario.State.Small;
            Shared.Mechanics.CoinsCount = 0;
            Shared.Mechanics.GameScore = 0;
            Shared.Mechanics.Lives = 3;
            Shared.Mechanics.TimeCounter.Reset();

            Engine.ChangeScene(nameof(WorldScreen), true, new object[] { (Action)(() => Engine.ChangeS-
cene(MapBase.LevelsNames[nameof(Level_1_1)], true)), $"world:get|{nameof(Level_1_1)}" });
        }
    }
}
```

2.2.2.23 Assets/Scenes/MapBase.cs

```
using DKEngine.Core;
using System.Collections.Generic;

namespace MarIO.Assets.Scenes
{
    public abstract class MapBase : Scene
    {
        public static Dictionary<string, string> LevelsNames = new Dictionary<string, string>()
        {
            { nameof(Test), "test" },
            { nameof(Level_1_1), "1-1" }
        };

        public sealed override void Init()
        {
            Load();

            Shared.Mechanics.TimeCounter.Start();
        }

        public sealed override void Unload()
        {
            Shared.Mechanics.TimeCounter.Reset();
        }

        public abstract void Load();
    }
}
```

2.2.2.24 Assets/Scenes/Test.cs

```
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using MarIO.Assets.Scripts;
using System.Drawing;

namespace MarIO.Assets.Scenes
{
    public class Test : MapBase
    {
        public static string StaticName = "test";

        public Test()
        {
            Name = StaticName;
            Shared.Mechanics.LastWorldType = typeof(Test);
        }

        public override void Load()
        {
            Group _1 = new Group()
            {
                Name = "ground1",
                InitCollider = true,
                Type = Block.BlockType.Ground2
            };
            _1.SizeInBlocks = new Vector3(50, 3, 0);
            _1.Transform.Position = new Vector3(0, 0, 0);

            Group _2 = new Group()
            {
                Name = "ground2",
                InitCollider = true,
                Type = Block.BlockType.Ground2
            };
            _2.SizeInBlocks = new Vector3(10, 3, 0);
            _2.Transform.Position = new Vector3(60 * 16, 0, 0);

            Group _3 = new Group()
            {
                Name = "ground3",
                Type = Block.BlockType.Ground2,
                InitCollider = true
            };
            _3.SizeInBlocks = new Vector3(50, 3, 0);
            _3.Transform.Position = new Vector3(80 * 16, 0, 0);

            for (int i = 0; i < 10; i++)
            {
                Block tmp = new Block()
                {
                    Type = Block.BlockType.Ground2,
                    Name = string.Format("PlatformTest_{0:00}", i)
                };
                tmp.Transform.Position = new Vector3(80 + 16 * i, -80, 0);
                tmp.InitCollider = true;
            }

            Block pipe = new Block()
            {
                Name = "pipe1",
                Type = Block.BlockType.Pipe1
            };
            pipe.Transform.Position = new Vector3(240, -32, 0);

            Block blck = new Block()
            {
                Name = "random1",
                Type = Block.BlockType.Ground2
            };
            blck.InitNewComponent<Collider>();
            blck.Collider.Area = new System.Drawing.RectangleF(0, 0, 16, 16);
            blck.Transform.Position = new Vector3(400, -16, 0);

            Block blck2 = new Block()
            {
                Type = Block.BlockType.Ground2,
                Name = "random2"
            };

            blck2.Transform.Position = new Vector3(600, -16, 0);
            blck2.InitNewComponent<Collider>();
            blck2.Collider.Area = new System.Drawing.RectangleF(0, 0, 16, 16);

            Goomba goomba = new Goomba();
            goomba.Transform.Position = new Vector3(500, -20, 0);

            Mario m = new Mario()
            {
                InitCameraController = true,
                InitCharacterController = true,
                InitCollider = true
            };
            m.Transform.Position = new Vector3(10, -10, 0);

            new MusicPlayer();

            Camera c = new Camera()
            {
                BackGround = Shared.Mechanics.OverworldBackground.ToColor()
            };

            new GUIUpdater();
            new SoundOutput();
            new BackgroundWorker();

            Trigger DeathZone = new Trigger();
            DeathZone.InitNewScript<DeathZoneScript>();
            DeathZone.Transform.Dimensions = new Vector3(3200, 10, 0);
            DeathZone.Transform.Position = new Vector3(0, 50, 0);
            DeathZone.Model = new Material(Color.Black, DeathZone);
        }
    }
}
```

2.2.2.25 Assets/Scenes/WorldScreen.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using MarIO.Assets.Models.Miscellaneous;
using System;
using System.Drawing;
using System.Linq;

namespace MarIO.Assets.Scenes
{
    public class WorldScreen : Scene
    {
        private static readonly TimeSpan _defautlTimeSpan = new TimeSpan(0, 0, 5);

        private TextBlock World;
        private TextBlock Lives;
        private Delayer Delayer;

        private static string RemainingLives = "";
        private static string WorldName = "";
        public static Action WorldChange;
        public static TimeSpan? Delay;

        public WorldScreen()
        {
            Name = nameof(WorldScreen);
        }

        public override void Init()
        {
            TextBlock _World = new TextBlock()
            {
                FontSize = 5,
                Foreground = Color.White,
                HAlignment = Text.HorizontalAlignment.Center,
                IsGUI = true,
                Name = "tx_const_world",
                Text = "WORLD",
                TextHAlignment = Text.HorizontalAlignment.Center,
                VAlignment = Text.VerticalAlignment.Center
            };
            _World.Transform.Position += new Vector3(0, -40, 0);
            _World.Transform.Dimensions = new Vector3(120, 30, 0);

            World = new TextBlock()
            {
                FontSize = 4,
                Foreground = Color.White,
                HAlignment = Text.HorizontalAlignment.Center,
                IsGUI = true,
                Name = "tx_world",
                TextHAlignment = Text.HorizontalAlignment.Center,
                VAlignment = Text.VerticalAlignment.Center,
                Text = WorldName
            };
            World.Transform.Position += new Vector3(0, -5, 0);
            World.Transform.Dimensions = new Vector3(100, 30, 0);

            GameObject holder = new GameObject();
            holder.Transform.Position = new Vector3(120, 140, 0);

            Heart _HeartIcon = new Heart(holder)
            {
                IsGUI = true,
                Name = "heart_icon"
            };

            _HeartIcon.Transform.Scale = new Vector3(3, 3, 0);

            Lives = new TextBlock(holder)
            {
                FontSize = 3.5f,
                IsGUI = true,
                TextHAlignment = Text.HorizontalAlignment.Center,
                Text = RemainingLives
            };
            Lives.Transform.Dimensions = new Vector3(40, 15, 0);
            Lives.Transform.Position += new Vector3(32, 8, 0);

            Delayer = new Delayer()
            {
                CalledAction = WorldChange,
                TimeToWait = Delay ?? _defautlTimeSpan
            };

            new Camera()
            {
                BackGround = Shared.Mechanics.WorldChangeBackground.ToColor()
            };

            if (Shared.Mechanics.MarioCurrentState == Mario.State.Dead)
                Shared.Mechanics.MarioCurrentState = Mario.State.Small;
        }

        public override void Set(params object[] args)
        {
            if (args == null)
                return;

            string[] stringParameters = args.Where(obj => obj is string).ToList().Cast<string>().ToArray();
            object[] otherParameters = args.Where(obj => !(obj is string)).ToArray();

            for (int i = 0; i < stringParameters.Length; i++)
            {
                string[] parameters = stringParameters[i].Split(':');

                switch (parameters[0])
                {
                    case "world":
                        if (parameters[1].Split('|')[0] == "get")
                        {
                            WorldName = MapBase.LevelsNames[parameters[1].Split('|')[1]];
                        }
                        else
                        {
                            WorldName = parameters[1];
                        }

                        break;

                    case "time":
                        Delay = TimeSpan.Parse(parameters[1]);
                        break;
                }
            }

            foreach (object item in otherParameters)
            {
                if (item is Action)
                {
                    WorldChange = ((Action)item);
                }
            }

            RemainingLives = string.Format($"*{Shared.Mechanics.Lives:00}");
        }

        public override void Unload()
        { }
    }
}
```

2.2.2.26 Assets/Scripts/BlockAnimatorScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;

namespace MarIO.Assets.Scripts
{
    public class BlockAnimatorScript : Script
    {
        private float AnimationHeight = 2;
        private float AnimationSpeed = 20;

        public BlockAnimatorScript(GameObject Parent)
            : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        {
        }

        protected override void Start()
        {
        }

        protected override void Update()
        {
            if (Shared.AnimatedWorldReferences.BlocksToUpdate.Count > 0)
            {
                for (int i = 0; i < Shared.AnimatedWorldReferences.BlocksToUpdate.Count; i++)
                {
                    float StartBlockY = Shared.AnimatedWorldReferences.BlocksStartPositions[i];
                    Block CurrentBlock = Shared.AnimatedWorldReferences.BlocksToUpdate[i];

                    if (CurrentBlock.State == Block.CollisionState.Up && StartBlockY - AnimationHeight < CurrentBlock.Transform.Position.Y)
                    {
                        CurrentBlock.Transform.Position -= new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);

                        if (CurrentBlock.Transform.Position.Y <= StartBlockY - AnimationHeight)
                        {
                            CurrentBlock.State = Block.CollisionState.Down;
                        }
                    }
                    else if (CurrentBlock.State == Block.CollisionState.Down && CurrentBlock.Transform.Position.Y < StartBlockY)
                    {
                        CurrentBlock.Transform.Position += new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);

                        if (CurrentBlock.Transform.Position.Y > StartBlockY)
                        {
                            CurrentBlock.State = Block.CollisionState.Stay;
                            CurrentBlock.Transform.Position = new Vector3(CurrentBlock.Transform.Position.X, StartBlockY, CurrentBlock.Transform.Position.Z);

                            Shared.AnimatedWorldReferences.BlocksStartPositions.RemoveAt(i);
                            Shared.AnimatedWorldReferences.BlocksToUpdate.RemoveAt(i);

                            CurrentBlock.CoinGot = false;

                            i--;
                        }
                    }
                }
            }
        }
    }
}
```

2.2.2.27 Assets/Scripts/CameraController.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Scripts
{
    public class CameraController : Script
    {
        private GameObject Player;

        private Camera TargetCam;
        private float PositionX;
        private float MaxCameraDistance;

        private Vector3 Offset;

        public CameraController(GameObject Parent)
            : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            MaxCameraDistance = Engine.Render.RenderWidth / 3;
            Offset = new Vector3(20, 0, 0);

            Player = GameObject.Find<GameObject>("Player");
            TargetCam = Component.Find<Camera>("Camera");
            TargetCam.Position = new Vector3(0, -160, 0);
        }

        protected override void Update()
        {
            if (Player.Transform.Position.X - TargetCam.Position.X > MaxCameraDistance)
            {
                TargetCam.Position += new Vector3(Player.Transform.Position.X - PositionX, 0, 0);
            }

            if (Player.Transform.Position.X < TargetCam.Position.X)
            {
                Player.Transform.Position = Player.Transform.Position.Add(TargetCam.Position.X - Player.Transform.Posi-
tion.X, 0, 0);
            }

            PositionX = Player.Transform.Position.X;
        }
    }
}
```

2.2.2.28 Assets/Scripts/CharacterController.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using MarIO.Assets.Scenes;
using System;
using static DKEngine.Core.Components.Transform;

namespace MarIO.Assets.Scripts
{
    public class CharacterController : Script
    {
        public bool Enabled = true;

        private Animator PlayerAnimator;
        private Mario Player;
        //private SoundSource SoundOutput;

        private float horiSpeed = 0;
        private float vertSpeed = 0;

        private const float MovementSpeed = 120f;
        private const float FloatSpeed = 300f;

        private const float Acceleration = 3.5f;

        private const float DeathAnimSpeed = 120f;

        private bool CanJump = true;
        private bool IsFalling = false;
        private bool Jumped = false;
        private bool IsFacingLeft = false;
        private bool EnemyKilledAnim = false;
        private bool FirstTimeDeadAnimPlay = true;

        private bool FirstTimePipeEnter = true;
        private float PipeEnterStartPosition;
        private float PipeEnterSpeed = 50f;

        private readonly TimeSpan WorldReload = new TimeSpan(0, 0, 3);
        private TimeSpan WorldReloadNow = new TimeSpan();

        private Mario.State LastState;
        private bool ChangingState = false;

        private string _idle
        {
            get
            {
                switch (Player.CurrentState)
                {
                    case Mario.State.Dead:
                    case Mario.State.Small:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_IDLE_LEFT : Shared.Assets.Animations.MA-
RIO_IDLE_RIGHT;

                    case Mario.State.Super:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_IDLE_LEFT : Shared.Assets.Animati-
ons.MARIO_SUPER_IDLE_RIGHT;

                    case Mario.State.Fire:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT : Shared.Assets.Animati-
ons.MARIO_FIRE_IDLE_RIGHT;

```

```
                    /*case Mario.State.Invincible:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_LEFT : Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_RIGHT;*/

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        private string _crouch
        {
            get
            {
                switch (Player.CurrentState)
                {
                    case Mario.State.Small:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_CROUCHING_LEFT : Shared.Assets.Animations.MARIO_CROUCHING_RIGHT;

                    case Mario.State.Super:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_CROUCHING_LEFT : Shared.Assets.Animations.MARIO_SUPER_CROUCHING_RIGHT;

                    case Mario.State.Fire:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_CROUCHING_LEFT : Shared.Assets.Animations.MARIO_FIRE_CROUCHING_RIGHT;

                    /*case Mario.State.Invincible:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_LEFT : Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_RIGHT;*/

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        private string _superPowerUp
        {
            get { return IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_POWERUP_LEFT : Shared.Assets.Animations.MARIO_SUPER_POWERUP_RIGHT; }
        }

        private string _firePowerUp
        {
            get { return IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_POWERUP_LEFT : Shared.Assets.Animations.MARIO_FIRE_POWERUP_RIGHT; }
        }

        private string IDLE
        {
            get
            {
                return Player.CurrentMovement == Mario.Movement.Crouching ? _crouch : _idle;
            }
        }

        private string MOVE
        {
            get
            {
                switch (Player.CurrentState)
                {
                    case Mario.State.Small:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_MOVE_RIGHT : Shared.Assets.Animations.MARIO_MOVE_LEFT;

                    case Mario.State.Super:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_SUPER_MOVE_RIGHT : Shared.Assets.Animations.MARIO_SUPER_MOVE_LEFT;

                    case Mario.State.Fire:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT : Shared.Assets.Animations.MARIO_FIRE_MOVE_LEFT;

                    /*case Mario.State.Invincible:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_RIGHT : Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_LEFT;*/

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        private string JUMP
        {
            get
            {
                switch (Player.CurrentState)
                {
                    case Mario.State.Small:
                        return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_JUMP_RIGHT : Shared.Assets.Animations.MARIO_JUMP_LEFT)
                                              : (IsFacingLeft ? Shared.Assets.Animations.MARIO_JUMP_LEFT : Shared.Assets.Animations.MARIO_JUMP_RIGHT);

                    case Mario.State.Super:
                        return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_SUPER_JUMP_RIGHT : Shared.Assets.Animations.MARIO_SUPER_JUMP_LEFT)
                                              : (IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_JUMP_LEFT : Shared.Assets.Animations.MARIO_SUPER_JUMP_RIGHT);

                    case Mario.State.Fire:
                        return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_FIRE_JUMP_RIGHT : Shared.Assets.Animations.MARIO_FIRE_JUMP_LEFT)
                                              : (IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_JUMP_LEFT : Shared.Assets.Animations.MARIO_FIRE_JUMP_RIGHT);

                    /*case Mario.State.Invincible:
                        return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_RIGHT : Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_LEFT)
                                              : (IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_LEFT : Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_RIGHT);*/

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        private string POWERUP
        {
            get
            {
                switch (LastState)
                {
                    case Mario.State.Small:
                        return _superPowerUp;

                    case Mario.State.Super:
                        return LastState < Player.CurrentState ? _firePowerUp : _superPowerUp;

                    case Mario.State.Fire:
                        return LastState < Player.CurrentState ? "" : _firePowerUp;

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        public CharacterController(GameObject Parent)
            : base(Parent)
        {
            this.Name = nameof(CharacterController);
            this.Parent.InitNewComponent<Collider>();
        }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            Player = GameObject.Find<Mario>("Player");
            PlayerAnimator = Component.Find<Animator>("Player_Animator");
            //SoundOutput = Component.Find<SoundSource>("Player_SoundSource");

            LastState = Player.CurrentState;

            Player.Animator.Play(Shared.Assets.Animations.MARIO_IDLE_RIGHT);
        }

        protected override void Update()
        {
            if (!Enabled)
                return;

            if (LastState != Player.CurrentState && Player.CurrentState != Mario.State.Dead)
            {
                if (!ChangingState)
                {
                    PlayerAnimator.Play(POWERUP);
                    Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_POWER_UP_SOUND);
                    bool FromSmalltoLarge = Player.CurrentState > Mario.State.Small && LastState == Mario.State.Small;
                    bool FromLargeToSmall = Player.CurrentState == Mario.State.Small && LastState == Mario.State.Super;
                    float YtoAdd = FromSmalltoLarge ? -16 : (FromLargeToSmall ? 0 : 16);
                    Player.Transform.Position += new Vector3(0, YtoAdd, 0);
                    ChangingState = true;

                    Player.LeftTrigger.Collider.Enabled = false;
                    Player.RightTrigger.Collider.Enabled = false;
                    Player.TopTrigger.Collider.Enabled = false;
                    Player.BottomTrigger.Collider.Enabled = false;

                    Player.Collider.Enabled = false;

                    return;
                }

                if (PlayerAnimator.NumberOfPlays > 5)
                {
                    LastState = Player.CurrentState;

                    Player.LeftTrigger.Collider.Enabled = true;
                    Player.RightTrigger.Collider.Enabled = true;
                    Player.TopTrigger.Collider.Enabled = true;
                    Player.BottomTrigger.Collider.Enabled = true;

                    Player.Collider.Enabled = true;

                    ChangingState = false;
                }
                else
                    return;
            }
            else if (Player.CurrentState == Mario.State.Dead)
            {
                DeadAnimation();
            }
            else if (Player.KilledEnemy)
            {
                Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_STOMP_SOUND);
                Player.KilledEnemy = false;
                EnemyKilledAnim = true;
                Jumped = true;
                IsFalling = false;
                vertSpeed = -FloatSpeed;
            }
            else if (Player.ChangeState)
            {
                if (FirstTimePipeEnter)
                {
                    Shared.Mechanics.FXSoundSource.StopSound(Shared.Assets.Sounds.OVERWORLD_THEME_SOUND);
                    Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_PIPE_ENTER_SOUND);
                    Player.Collider.Enabled = false;
                    PipeEnterStartPosition = Player.PipeEnteredInDirection == Direction.Down ? Player.Transform.Position.Y : Player.Transform.Position.X;
                    horiSpeed = 0;
                    vertSpeed = 0;
                    FirstTimePipeEnter = false;
                }

                if (Player.PipeEnteredInDirection == Direction.Right)
                {
                    if (Player.Transform.Position.X < PipeEnterStartPosition + 16)
                    {
                        horiSpeed = PipeEnterSpeed;
                    }
                    else
                    {
                        Player.WorldManager.CurrentlyEnteredPipeScript = Player.EnteredPipe;
                    }
                }
                else if (Player.PipeEnteredInDirection == Direction.Down)
                {
                    if (Player.Transform.Position.Y < PipeEnterStartPosition + 16)
                    {
                        vertSpeed = PipeEnterSpeed;
                    }
                    else
                    {
                        Player.WorldManager.CurrentlyEnteredPipeScript = Player.EnteredPipe;
                    }
                }
            }
            else if (Player.CurrentState > Mario.State.Dead)
            {
                Movement();
            }

            Player.Transform.Position = Player.Transform.Position.Add(horiSpeed * Engine.DeltaTime, vertSpeed * Engine.DeltaTime, 0);

            AnimationControl();
        }

        private void DeadAnimation()
        {
            horiSpeed = 0;

            if (FirstTimeDeadAnimPlay)
            {
                Player.Collider.Enabled = false;
                Player.BottomTrigger.Collider.Enabled = false;
                Player.LeftTrigger.Collider.Enabled = false;
                Player.RightTrigger.Collider.Enabled = false;
                Player.TopTrigger.Collider.Enabled = false;

                vertSpeed = -FloatSpeed;

                FirstTimeDeadAnimPlay = false;

                Shared.Mechanics.FXSoundSource.StopSound(Shared.Assets.Sounds.OVERWORLD_THEME_SOUND);
                Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_MARIO_DIE_SOUND);
            }
            else
            {
                vertSpeed += Engine.DeltaTime * DeathAnimSpeed * Acceleration;

                WorldReloadNow += new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime * 1000));

                if (WorldReloadNow > WorldReload)
                {
                    Shared.Mechanics.Lives--;

                    if (Shared.Mechanics.Lives == 0)
                        Engine.ChangeScene(nameof(GameOver), true);
                    else
                        Engine.ChangeScene(nameof(WorldScreen), true);
                }
            }
        }

        private void Movement()
        {
            if (Player.Collider.Collision(Direction.Down))
            {
                IsFalling = false;
                Jumped = false;

                vertSpeed = 0;
            }

            if (Engine.Input.IsKeyDown(ConsoleKey.A) || horiSpeed < 0)
            {
                Left();
            }

            if (Engine.Input.IsKeyDown(ConsoleKey.W) || Jumped)
            {
                Jump();
            }

            if (Engine.Input.IsKeyDown(ConsoleKey.D) || horiSpeed > 0)
            {
                Right();
            }

            if (Engine.Input.IsKeyDown(ConsoleKey.S))
            {
                if (vertSpeed == 0)
                {
                    horiSpeed = 0;
                    Player.CurrentMovement = Mario.Movement.Crouching;
                }
            }
            else
            {
                Player.CurrentMovement = Mario.Movement.Standing;
            }

            if (!Player.Collider.Collision(Direction.Down))
            {
                Fall();
            }
        }

        private void Jump()
        {
            if (Engine.Input.IsKeyDown(ConsoleKey.W))
            {
                Player.CurrentMovement = Mario.Movement.Standing;

                if (CanJump)
                {
                    if (EnemyKilledAnim)
                    {
                        vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed * 2;

                        if (vertSpeed <= 0)
                        {
                            IsFalling = true;
                            EnemyKilledAnim = false;
                        }
                    }
                    else if (!IsFalling)
                    {
                        if (vertSpeed == 0 && !Jumped)
                        {
                            Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_MARIO_JUMP_SOUND);
                            vertSpeed = -FloatSpeed * 1.5f;
                            Jumped = true;
                        }
                        else if (!Player.Collider.Collision(Direction.Up) && vertSpeed < 0)
                        {
                            vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
                        }
                        else
                        {
                            vertSpeed = 0;
                            IsFalling = true;
                        }
                    }
                }
            }
            else if (Jumped)
            {
                if (EnemyKilledAnim)
                {
                    vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed * 4;

                    if (vertSpeed <= 0)
                    {
                        IsFalling = true;
                        EnemyKilledAnim = false;
                    }
                }
                else if (!IsFalling)
                {
                    vertSpeed = -vertSpeed;
                    IsFalling = true;
                    EnemyKilledAnim = false;
                }
            }
        }

        private void Left()
        {
            if (Engine.Input.IsKeyDown(ConsoleKey.A))
            {
                Player.CurrentMovement = Mario.Movement.Standing;

                IsFacingLeft = true;
                if (!Player.Collider.Collision(Direction.Left) && horiSpeed > -MovementSpeed)
                {
                    horiSpeed -= Engine.DeltaTime * Acceleration * MovementSpeed;
                }
                else if (Player.Collider.Collision(Direction.Left))
                {
                    horiSpeed = 0;
                }
                else
                {
                    horiSpeed = -MovementSpeed;
                }
            }
            else if (horiSpeed < 0)
            {
                IsFacingLeft = true;
                horiSpeed += Engine.DeltaTime * Acceleration * MovementSpeed * 4;

                if (horiSpeed >= 0 || Player.Collider.Collision(Direction.Left))
                {
                    horiSpeed = 0;
                }
            }
        }

        private void Right()
        {
            if (Engine.Input.IsKeyDown(ConsoleKey.D))
            {
                Player.CurrentMovement = Mario.Movement.Standing;

                IsFacingLeft = false;
                if (!Player.Collider.Collision(Direction.Right) && horiSpeed < MovementSpeed)
                {
                    horiSpeed += Engine.DeltaTime * Acceleration * MovementSpeed;
                }
                else if (Player.Collider.Collision(Direction.Right))
                {
                    horiSpeed = 0;
                }
                else
                {
                    horiSpeed = MovementSpeed;
                }
            }
            else if (horiSpeed > 0)
            {
                IsFacingLeft = false;
                horiSpeed -= Engine.DeltaTime * Acceleration * MovementSpeed * 2;

                if (horiSpeed <= 0 || Player.Collider.Collision(Direction.Right))
                {
                    horiSpeed = 0;
                }
            }
        }

        private void Fall()
        {
            if (!IsFalling && !Jumped)
            {
                vertSpeed = 0;
                Jumped = true;
                IsFalling = true;
            }
            else if (IsFalling)
            {
                if (vertSpeed < FloatSpeed)
                {
                    vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;

                    if (vertSpeed > FloatSpeed)
                        vertSpeed = FloatSpeed;
                }
                else
                {
                    vertSpeed = FloatSpeed;
                }
            }
        }

        private void AnimationControl()
        {
            if (Player.CurrentState > Mario.State.Dead)
            {
                if (Jumped)
                {
                    PlayerAnimator.Play(JUMP);
                }
                else
                {
                    if (horiSpeed != 0)
                        PlayerAnimator.Play(MOVE);
                    else
                        PlayerAnimator.Play(IDLE);
                }
            }
            else
            {
                PlayerAnimator.Play(Shared.Assets.Animations.MARIO_DEAD);
            }
        }
    }
}
```

2.2.2.29 Assets/Scripts/DeathZoneScript.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using System.Diagnostics;

namespace MarIO.Assets.Scripts
{
    public class DeathZoneScript : Script
    {
        public DeathZoneScript(GameObject Parent) : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        {
            Debug.WriteLine($"{e.Parent}");

            if (e.Parent is AnimatedObject)
            {
                ((AnimatedObject)e.Parent).IsDestroyed = true;
            }
        }

        protected override void Start()
        { }

        protected override void Update()
        { }
    }
}
```

2.2.2.30 Assets/Scripts/DelayScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using System;

namespace MarIO.Assets.Scripts
{
    public class DelayScript : Script
    {
        private TimeSpan Checker;
        private Delayer Source;

        public DelayScript(GameObject Parent) : base(Parent)
        {
            Source = (Delayer)Parent;
        }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            Checker = new TimeSpan();
        }

        protected override void Update()
        {
            Checker += new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime * 1000));

            if (Checker > Source?.TimeToWait)
            {
                Source?.CalledAction?.Invoke();
            }
        }
    }
}
```

2.2.2.31 Assets/Scripts/EnemyControllerScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;
using MarIO.Assets.Models;
using static DKEngine.Core.Components.Transform;

namespace MarIO.Assets.Scripts
{
    public class GoombaController : Script
    {
        private const int Speed = 20;
        private const int FloatSpeed = 60;
        private const int Acceleration = 20;

        private int CurrentSpeed = 0;
        private float vertSpeed = 0;
        private bool IsFalling = false;

        private bool firstTimeDeadAnimation = true;

        private float DeadTimeCurrent = 0f;
        private const float DeadTime = 3f;

        private Enemy Target;

        public GoombaController(GameObject Parent) : base(Parent)
        {
            Target = (Enemy)Parent;
        }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            CurrentSpeed = -Speed;
        }

        protected override void Update()
        {
            if (!Target.IsDestroyed)
            {
                Movement();
            }
            else
            {
                DeadAnimation();
            }
        }

        private void Movement()
        {
            if (Target.Collider.Collision(Direction.Left))
            {
                CurrentSpeed = Speed;
            }

            if (Target.Collider.Collision(Direction.Right))
            {
                CurrentSpeed = -Speed;
            }

            if (!Target.Collider.Collision(Direction.Down))
            {
                if (!IsFalling)
                {
                    vertSpeed = 0;
                    IsFalling = true;
                }
                else
                {
                    if (vertSpeed < FloatSpeed)
                    {
                        vertSpeed += Engine.DeltaTime * Acceleration;
                    }
                    else
                    {
                        vertSpeed = FloatSpeed;
                    }
                }
            }
            else if (IsFalling)
            {
                vertSpeed = 0;
                IsFalling = false;
            }

            Target.Transform.Position += new Vector3(CurrentSpeed * Engine.DeltaTime, vertSpeed * Engine.DeltaTime, 0);
        }

        private void DeadAnimation()
        {
            if (firstTimeDeadAnimation)
            {
                Shared.Mechanics.GameScore += Shared.Mechanics.GOOMBA_POINTS;
                TextBlock FloatingText = new TextBlock()
                {
                    Text = string.Format("{0}", Shared.Mechanics.GOOMBA_POINTS),
                    TextShadow = true
                };
                FloatingText.Transform.Position = Target.Transform.Position;
                FloatingText.Transform.Dimensions = new Vector3(20, 6, 0);
                FloatingText.AddAsFloatingText();

                Target.Collider.Enabled = false;
                Target.Animator.Play("dead");
                firstTimeDeadAnimation = false;
                Target.Transform.Position += new Vector3(0, 8, 0);
            }

            DeadTimeCurrent += Engine.DeltaTime;

            if (DeadTimeCurrent > DeadTime)
            {
                Target.Destroy();
            }
        }
    }
}
```

2.2.2.32 Assets/Scripts/FloatingCoinAnimatorScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models.Miscellaneous;

namespace MarIO.Assets.Scripts
{
    public class FloatingCoinAnimatorScript : Script
    {
        private float AnimationHeight = 60;
        private float AnimationSpeed = 20;

        public FloatingCoinAnimatorScript(GameObject Parent)
            : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        { }

        protected override void Update()
        {
            if (Shared.AnimatedWorldReferences.FloatingCoins.Count > 0)
            {
                for (int i = 0; i < Shared.AnimatedWorldReferences.FloatingCoins.Count; i++)
                {
                    Coin currentCoin = Shared.AnimatedWorldReferences.FloatingCoins[i];
                    float currentCoinStartPosition = Shared.AnimatedWorldReferences.FloatingCoinsStartPosition[i];

                    if (currentCoin.Transform.Position.Y > currentCoinStartPosition - AnimationHeight)
                    {
                        currentCoin.Transform.Position -= new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);

                        if (currentCoin.Transform.Position.Y <= currentCoinStartPosition - AnimationHeight)
                        {
                            currentCoin.Destroy();

                            Shared.AnimatedWorldReferences.FloatingCoins.RemoveAt(i);
                            Shared.AnimatedWorldReferences.FloatingCoinsStartPosition.RemoveAt(i);

                            i--;
                        }
                    }
                }
            }
        }
    }
}
```

2.2.2.33 Assets/Scripts/FloatingTextAnimatorScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;

namespace MarIO.Assets.Scripts
{
    public class FloatingTextAnimatorScript : Script
    {
        private float AnimationHeight = 30;
        private float AnimationSpeed = 20;

        public FloatingTextAnimatorScript(GameObject Parent)
            : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        { }

        protected override void Update()
        {
            if (Shared.AnimatedWorldReferences.FloatingTexts.Count > 0)
            {
                for (int i = 0; i < Shared.AnimatedWorldReferences.FloatingTexts.Count; i++)
                {
                    float StartTextBlockY = Shared.AnimatedWorldReferences.FloatingTextStartPosition[i];
                    TextBlock CurrentTextBlock = Shared.AnimatedWorldReferences.FloatingTexts[i];

                    if (CurrentTextBlock.Transform.Position.Y > StartTextBlockY - AnimationHeight)
                    {
                        CurrentTextBlock.Transform.Position -= new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);

                        if (CurrentTextBlock.Transform.Position.Y < StartTextBlockY - AnimationHeight)
                        {
                            Shared.AnimatedWorldReferences.FloatingTextStartPosition.RemoveAt(i);
                            Shared.AnimatedWorldReferences.FloatingTexts.RemoveAt(i);

                            CurrentTextBlock.Destroy();

                            i--;
                        }
                    }
                }
            }
        }
    }
}
```

2.2.2.34 Assets/Scripts/GUIUpdateScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using DKEngine.Core.UI;

namespace MarIO.Assets.Scripts
{
    public class GUIUpdateScript : Script
    {
        private TextBlock Time;
        private TextBlock Coins;
        private TextBlock World;
        private TextBlock Lives;
        private TextBlock Score;

        public GUIUpdateScript(GameObject Parent) : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            this.World = GameObject.Find<TextBlock>("txt_World");
            this.Time = GameObject.Find<TextBlock>("txt_Time");
            this.Score = GameObject.Find<TextBlock>("txt_Score");
            this.Coins = GameObject.Find<TextBlock>("txt_Coins");
            this.Lives = GameObject.Find<TextBlock>("txt_Lives");

            this.World.Text = Engine.SceneName;
            this.Time.Text = string.Format("{0:000}", Shared.Mechanics.TimeLeft.TotalSeconds);
            this.Score.Text = Shared.Mechanics.GameScoreStr;
            this.Coins.Text = string.Format("*{0:00}", Shared.Mechanics.CoinsCount);
            this.Lives.Text = string.Format("*{0:00}", Shared.Mechanics.Lives);
        }

        protected override void Update()
        {
            this.Time.Text = string.Format("{0:000}", Shared.Mechanics.TimeLeft.TotalSeconds);
            this.Score.Text = Shared.Mechanics.GameScoreStr;
            this.Coins.Text = string.Format("*{0:00}", Shared.Mechanics.CoinsCount);
            this.Lives.Text = string.Format("*{0:00}", Shared.Mechanics.Lives);
        }
    }
}
```

2.2.2.35 Assets/Scripts/MainMenuSpawnScript.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;

namespace MarIO.Assets.Scripts
{
    public class MainMenuSpawnScript : Script
    {
        private Vector3 Position;

        public MainMenuSpawnScript(GameObject Parent) : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        {
            e.Parent.Transform.Position = Position;
        }

        protected override void Start()
        {
            Position = new Vector3(320, 176, 0);

            Goomba e = new Goomba()
            {
                Name = "Bot"
            };
            e.Transform.Position = Position;
        }

        protected override void Update()
        { }
    }
}
```

2.2.2.36 Assets/Scripts/MarioTriggerColliderScript.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using MarIO.Assets.Models.Miscellaneous;
using System.Diagnostics;

namespace MarIO.Assets.Scripts
{
public class BottomMarioChecker : Script
{
private Mario Mario;

public BottomMarioChecker(GameObject Parent) : base(Parent)
{ }
```

```
protected override void OnColliderEnter(Collider e)
{
if (e.Parent is Enemy)
{
Enemy tmp = e.Parent as Enemy;
Debug.WriteLine(string.Format("Zabil jsi {0}", tmp.Name));
tmp.IsDestroyed = true;
Mario.KilledEnemy = true;
}
else if (e.Parent is PowerUp)
{
((PowerUp)e.Parent).OnPickedUp?.Invoke();
}
}

protected override void Start()
{
Mario = GameObject.Find<Mario>("Player");
}

protected override void Update()
{ }
}

internal class TopMarioChecker : Script
{
private Mario Mario;

public TopMarioChecker(GameObject Parent) : base(Parent)
{ }

protected override void OnColliderEnter(Collider e)
{
if (e.Parent is Enemy)
{
Debug.WriteLine(string.Format("Zabilo Tě {0}", e.Parent.TypeName));
Mario.CurrentState--;
}
else if (e.Parent is Block)
{
Block tmp = e.Parent as Block;

if (tmp.State == Block.CollisionState.Stay)
{
    a.   tmp.AnimateBlockCollision();
}

tmp.GetContent();
}
else if (e.Parent is PowerUp)
{
((PowerUp)e.Parent).OnPickedUp?.Invoke();
}
}

protected override void Start()
{
Mario = GameObject.Find<Mario>("Player");
}

protected override void Update()
{ }
}

internal class LeftMarioChecker : Script
{
private Mario Mario;

public LeftMarioChecker(GameObject Parent) : base(Parent)
{ }

protected override void OnColliderEnter(Collider e)
{
if (e.Parent is Enemy)
{
Debug.WriteLine(string.Format("Zabilo Tě {0}", e.Parent.TypeName));
Mario.CurrentState--;
//Mario?.Destroy();
}
else if (e.Parent is PowerUp)
{
((PowerUp)e.Parent).OnPickedUp?.Invoke();
}
}

protected override void Start()
{
Mario = GameObject.Find<Mario>("Player");
}

protected override void Update()
{ }
}

internal class RightMarioChecker : Script
{
private Mario Mario;

public RightMarioChecker(GameObject Parent) : base(Parent)
{ }

protected override void OnColliderEnter(Collider e)
{
if (e.Parent is Enemy)
{
Debug.WriteLine(string.Format("Zabilo Tě {0}", e.Parent.TypeName));
Mario.CurrentState--;
//Mario?.Destroy();
}
else if (e.Parent is PowerUp)
{
((PowerUp)e.Parent).OnPickedUp?.Invoke();
}
}

protected override void Start()
{
Mario = GameObject.Find<Mario>("Player");
}

protected override void Update()
{ }
}
}
```

2.2.2.37 Assets/Scripts/MusicScript.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using System;
using System.Diagnostics;

namespace MarIO.Assets.Scripts
{
    public class MusicScript : Script
    {
        private Sound Music;
        private TimeSpan MusicLenght;

        private Stopwatch Timer;

        public MusicScript(GameObject Parent) : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            Music = Shared.Assets.Sounds.OVERWORLD_THEME_SOUND;
            MusicLenght = Music.FileReader.TotalTime;
            Shared.Mechanics.FXSoundSource.PlaySound(Music);
            Timer = Stopwatch.StartNew();
        }

        protected override void Update()
        {
            if (Timer.Elapsed > MusicLenght)
            {
                Shared.Mechanics.FXSoundSource.PlaySound(Music);

                Timer.Restart();
            }
        }
    }
}
```

2.2.2.38 Assets/Scripts/PipePort.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;

namespace MarIO.Assets.Scripts
{
    public class PipePort : Script
    {
        private Mario Player;
        public Block Pipe;

        public PipePort(GameObject Parent) : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        {
            if (Pipe.SpecialAction != null)
            {
                if (e.Parent == Player)
                {
                    switch (Pipe.PipeEnterDirection)
                    {
                        case Transform.Direction.Up:
                            break;

                        case Transform.Direction.Left:
                            break;

                        case Transform.Direction.Down:
                            if (Player.CurrentMovement == Mario.Movement.Crouching)
                            {
                                Player.PipeEnter(Pipe);
                            }
                            break;

                        case Transform.Direction.Right:
                            if (Player.CurrentMovement == Mario.Movement.Standing)
                            {
                                Player.PipeEnter(Pipe);
                            }
                            break;

                        default:
                            break;
                    }
                }
            }
        }

        protected override void Start()
        {
            Player = GameObject.Find<Mario>("Player");
            Pipe = (Block)Parent;
        }

        protected override void Update()
        { }
    }
}
```

2.2.2.39 Assets/Scripts/PowerUpScript.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using MarIO.Assets.Models.Miscellaneous;
using System;
using static DKEngine.Core.Components.Transform;

namespace MarIO.Assets.Scripts
{
    internal class PowerUpScript : Script
    {
        private PowerUp Target;
        private bool CreatedForFirstTime = true;
        private bool CreatedAnimation = true;
        private float CreatedStartY;
        private const float CreationAnimationSpeed = 20f;

        private const float Speed = 80f;
        private const float FloatSpeed = 250f;
        private const float Acceleration = 3.5f;

        private float CurrentSpeed = 0;
        private float vertSpeed = 0;
        private bool IsFalling = false;
        private bool Jumped = false;

        public PowerUpScript(GameObject Parent) : base(Parent)
        {
            Target = Parent as PowerUp;
        }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
            CurrentSpeed = Speed;

            Target.PlayerReference = GameObject.Find<Mario>("Player");
        }

        protected override void Update()
        {
            if (CreatedForFirstTime)
            {
                Target.Collider.Enabled = false;
                CreatedStartY = Target.Transform.Position.Y;
                CreatedForFirstTime = false;
                return;
            }
            else if (CreatedAnimation)
            {
                if (CreatedStartY < Target.Transform.Position.Y + 16)
                {
                    Target.Transform.Position -= new Vector3(0, Engine.DeltaTime * CreationAnimationSpeed, 0);
                }
                else
                {
                    Target.Transform.Position = new Vector3(Target.Transform.Position.X, CreatedStartY - 16, Target.Trans-
form.Position.Z);
                    Target.Collider.Enabled = true;
                    CreatedAnimation = false;
                }

                return;
            }
            else
            {
                switch (Target.Type)
                {
                    case PowerUp.PowerUpType.Mushroom:
                        MushroomMovement();
                        break;

                    case PowerUp.PowerUpType.Flower:
                        CurrentSpeed = 0;
                        break;

                    case PowerUp.PowerUpType.Star:
                        StarMovement();
                        break;

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        private void MushroomMovement()
        {
            if (Target.Collider.Collision(Direction.Left))
            {
                CurrentSpeed = Speed;
            }

            if (Target.Collider.Collision(Direction.Right))
            {
                CurrentSpeed = -Speed;
            }

            if (!Target.Collider.Collision(Direction.Down))
            {
                if (!IsFalling)
                {
                    vertSpeed = 0;
                    IsFalling = true;
                }
                else
                {
                    if (vertSpeed < FloatSpeed)
                    {
                        vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
                    }
                    else
                    {
                        vertSpeed = FloatSpeed;
                    }
                }
            }
            else if (IsFalling)
            {
                vertSpeed = 0;
                IsFalling = false;
            }

            Target.Transform.Position += new Vector3(CurrentSpeed * Engine.DeltaTime, vertSpeed * Engine.DeltaTime, 0);
        }

        private void StarMovement()
        {
            if (Target.Collider.Collision(Direction.Left))
            {
                CurrentSpeed = Speed;
            }

            if (Target.Collider.Collision(Direction.Right))
            {
                CurrentSpeed = -Speed;
            }

            if (!Target.Collider.Collision(Direction.Down))
            {
                if (vertSpeed == 0 && !Jumped)
                {
                    vertSpeed = -FloatSpeed * 1.5f;
                    Jumped = true;
                }
                else if (!Target.Collider.Collision(Direction.Up) && vertSpeed < 0)
                {
                    vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
                }
                else
                {
                    vertSpeed = 0;
                    IsFalling = true;
                }
            }
            else
            {
                if (!IsFalling && !Jumped)
                {
                    vertSpeed = 0;
                    Jumped = true;
                    IsFalling = true;
                }
                else if (IsFalling)
                {
                    if (vertSpeed < FloatSpeed)
                    {
                        vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
                    }
                    else
                    {
                        vertSpeed = FloatSpeed;
                    }
                }
            }
        }
    }
}
```

2.2.2.40 Assets/Scripts/SpecialBlocksUpdateScript.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;

namespace MarIO.Assets.Scripts
{
    public class SpecialBlocksUpdateScript : Script
    {
        public SpecialBlocksUpdateScript(GameObject Parent)
            : base(Parent)
        { }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        {
        }

        protected override void Update()
        {
            while (Shared.AnimatedWorldReferences.SpecialActions.Count > 0)
            {
                Shared.AnimatedWorldReferences.SpecialActions.Pop().SpecialAction();
            }
        }
    }
}
```

2.2.2.41 Assets/Scripts/WorldChangeManagerScript.cs

```
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;

namespace MarIO.Assets.Scripts
{
    public class WorldChangeManagerScript : Script
    {
        public Block CurrentlyEnteredPipeScript;

        public WorldChangeManagerScript(GameObject Parent) : base(Parent)
        {
            Name = "worldManager";
        }

        protected override void OnColliderEnter(Collider e)
        { }

        protected override void Start()
        { }

        protected override void Update()
        {
            CurrentlyEnteredPipeScript?.SpecialAction();
            CurrentlyEnteredPipeScript = null;
        }
    }
}
```

2.2.2.42 Assets/Scripts/WorldEnd.cs

```
using DKEngine;
using DKEngine.Core;
using DKEngine.Core.Components;
using MarIO.Assets.Models;
using MarIO.Assets.Scenes;
using System;
using static DKEngine.Core.Components.Transform;

namespace MarIO.Assets.Scripts
{
    internal class WorldEnd : Script
    {
        private Mario Player;
        private CharacterController PlayerController;
        private Animator PlayerAnimator;

        private float horiSpeed = 0;
        private float vertSpeed = 0;
        private float Distance = 180;
        private float startX;

        private const float MovementSpeed = 80f;
        private const float FloatSpeed = 300f;

        private const float Acceleration = 3.5f;

        private readonly TimeSpan _delay = new TimeSpan(0, 0, 3);
        private TimeSpan Delay = new TimeSpan();

        private string MOVE
        {
            get
            {
                switch (Player.CurrentState)
                {
                    case Mario.State.Small:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_MOVE_RIGHT : Shared.Assets.Animations.MARIO_MOVE_LEFT;

                    case Mario.State.Super:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_SUPER_MOVE_RIGHT : Shared.Assets.Animations.MARIO_SUPER_MOVE_LEFT;

                    case Mario.State.Fire:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT : Shared.Assets.Animations.MARIO_FIRE_MOVE_LEFT;

                    /*case Mario.State.Invincible:
                        return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_RIGHT : Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_LEFT;*/

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        private string IDLE
        {
            get
            {
                switch (Player.CurrentState)
                {
                    case Mario.State.Dead:
                    case Mario.State.Small:
                        return horiSpeed < 0 ? Shared.Assets.Animations.MARIO_IDLE_LEFT : Shared.Assets.Animations.MARIO_IDLE_RIGHT;

                    case Mario.State.Super:
                        return horiSpeed < 0 ? Shared.Assets.Animations.MARIO_SUPER_IDLE_LEFT : Shared.Assets.Animations.MARIO_SUPER_IDLE_RIGHT;

                    case Mario.State.Fire:
                        return horiSpeed < 0 ? Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT : Shared.Assets.Animations.MARIO_FIRE_IDLE_RIGHT;

                    /*case Mario.State.Invincible:
                        return IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_LEFT : Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_RIGHT;*/

                    default:
                        throw new Exception("JAK");
                }
            }
        }

        public WorldEnd(GameObject Parent) : base(Parent)
        {
            startX = Parent.Transform.Position.X;
        }

        protected override void OnColliderEnter(Collider e)
        {
            if (e.Parent is Mario)
                PlayerController.Enabled = false;
        }

        protected override void Start()
        {
            Player = GameObject.Find<Mario>("Player");
            PlayerAnimator = Component.Find<Animator>("Player_Animator");
            PlayerController = Script.Find<CharacterController>(nameof(CharacterController));
        }

        protected override void Update()
        {
            if (Shared.Mechanics.TimeLeft.TotalSeconds <= 0)
            {
                Player.CurrentState = Mario.State.Dead;
                Shared.Mechanics.TimeCounter.Stop();
            }

            if (!PlayerController.Enabled)
            {
                PlayerAnimator.Play(MOVE);

                if (!Player.Collider.Collision(Direction.Down))
                {
                    if (vertSpeed < FloatSpeed)
                    {
                        vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;

                        if (vertSpeed > FloatSpeed)
                            vertSpeed = FloatSpeed;
                    }
                    else
                    {
                        vertSpeed = FloatSpeed;
                    }
                }

                if (Player.Transform.Position.X > startX + Distance)
                {
                    PlayerAnimator.Play(IDLE);
                    Delay += new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime * 1000));
                    if (_delay < Delay)
                    {
                        Shared.Mechanics.FXSoundSource.StopSound(Shared.As-
sets.Sounds.OVERWORLD_THEME_SOUND);
                        Engine.ChangeScene(nameof(GameOver), true);
                    }
                }
                else
                {
                    this.Player.Transform.Position += new Vector3(MovementSpeed * Engine.DeltaTime, vertSpeed, 0);
                }
            }
        }
    }
}
```

3 Závěr

Celý kód ročníkové práce je dostupný na mém osobím GitHubu, který jste si mohli přečíst v kapitole „Úvod“. Program pracuje s mnoha různými strukturami, třídami, metodami, ať už těch napsaných mnou v jazyku C# nebo různých dalších naimporotvaných C++ knihoven. V základu se jedna o velice jednoduchý 2D herní engine pracující se slušnou vykonností. Zvládá přehrávat animace pomocí obrázků gif, vakreslovat objekty přes sebe s průhledností nebo přehrávat zvukové efekty. Využito bylo různých knihoven, se kterými jsem měl možnost se naučit spousty nových dovedností. Jednou z nich byla knihovna NAudio, sloužící k přehrávání zvuků. O té zde byla zmínka v úvodu této ročníkové práce. Program se povedlo dostat do prezentovatelné podoby a to díky podpoře mých přátel. Tímto bych chtěl poděkovat mým kamarádům, Marianu Dolinskému, Tomáši Lošťákovi a Pavlu Jakubcovi za jejich podporu.

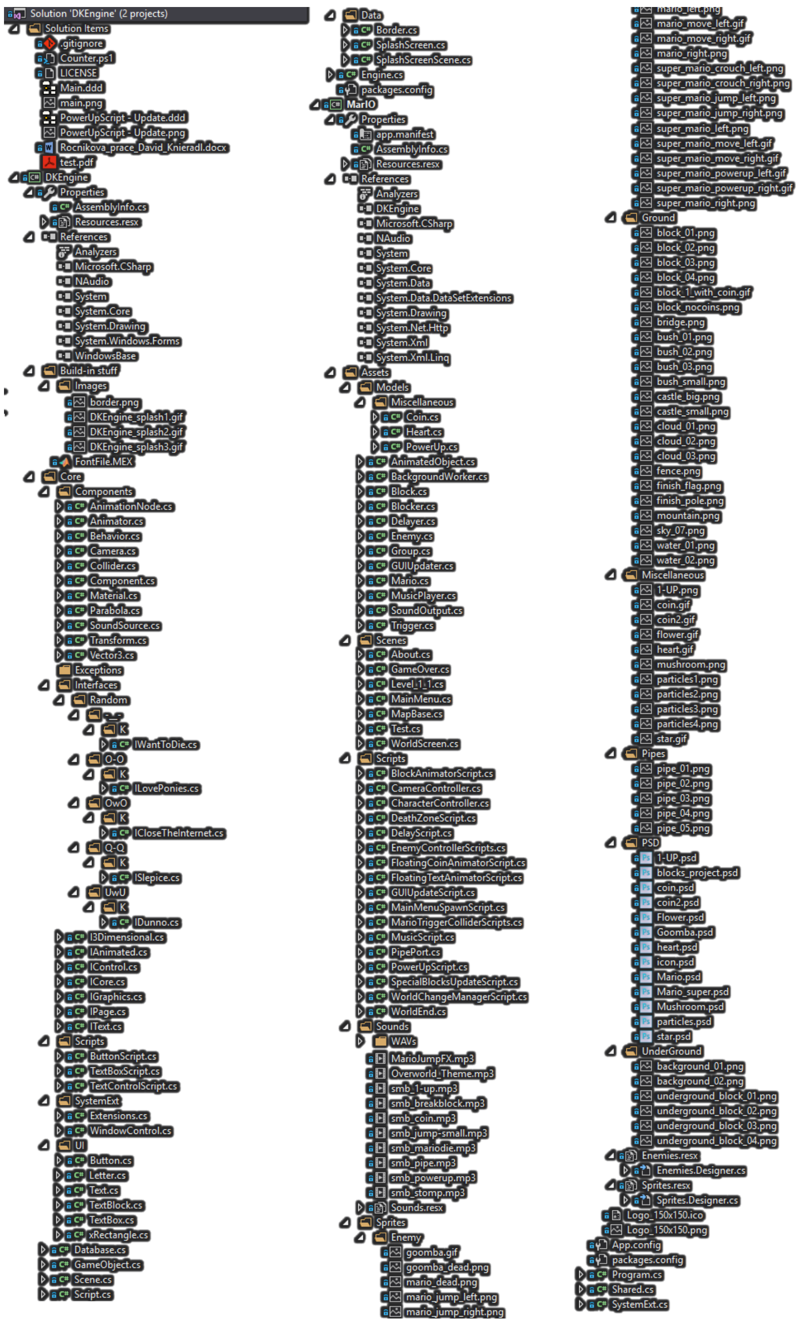
4 Literatura

MSDN [ON-LINE] [CIT. 2017/05/14] DOSTUPNÉ NA MICROSOFT DEVELOPER NETWORK https://msdn.microsoft.com/cs-cz/default.aspx

STACKOVERFLOW [ON-LINE] [CIT. 2017/05/14] DOSTUPNÉ NA STACKOVERFLOW https://stackoverflow.com/

Přílohy

A Stromová struktura solution



```
Solution 'DKEngine' (2 projects)
  Solution Items
    .gitignore
    Counter.ps1
    LICENSE
    Main.ddd
    main.png
    PowerUpScript - Update.ddd
    PowerUpScript - Update.png
    Rocnikova_prace_David_Knieradl.docx
    test.pdf
  DKEngine
    Properties
      AssemblyInfo.cs
      Resources.resx
    References
      Analyzers
      Microsoft.CSharp
      NAudio
      System
      System.Core
      System.Drawing
      System.Windows.Forms
      WindowsBase
    Build-in stuff
      Images
        border.png
        DKEngine_splash1.gif
        DKEngine_splash2.gif
        DKEngine_splash3.gif
      FontFile.MEX
    Core
      Components
        AnimationNode.cs
        Animator.cs
        Behavior.cs
        Camera.cs
        Collider.cs
        Component.cs
        Material.cs
        Parabola.cs
        SoundSource.cs
        Transform.cs
        Vector3.cs
      Exceptions
      Interfaces
        Random
          -_-
            K
              IWantToDie.cs
          O-O
            K
              ILovePonies.cs
          OwO
            K
              ICloseTheInternet.cs
          Q-Q
            K
              ISlepice.cs
          UwU
            K
              IDunno.cs
        I3Dimensional.cs
        IAnimated.cs
        IControl.cs
        ICore.cs
        IGraphics.cs
        IPage.cs
        IText.cs
      Scripts
        ButtonScript.cs
        TextBoxScript.cs
        TextControlScript.cs
      SystemExt
        Extensions.cs
        WindowControl.cs
      UI
        Button.cs
        Letter.cs
        Text.cs
        TextBlock.cs
        TextBox.cs
        xRectangle.cs
    Database.cs
    GameObject.cs
    Scene.cs
    Script.cs
    Data
      Border.cs
      SplashScreen.cs
      SplashScreenScene.cs
    Engine.cs
    packages.config
  MarIO
    Properties
      app.manifest
      AssemblyInfo.cs
      Resources.resx
    References
      Analyzers
      DKEngine
      Microsoft.CSharp
      NAudio
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Drawing
      System.Net.Http
      System.Xml
      System.Xml.Linq
    Assets
      Models
        Miscellaneous
          Coin.cs
          Heart.cs
          PowerUp.cs
        AnimatedObject.cs
        BackgroundWorker.cs
        Block.cs
        Blocker.cs
        Delayer.cs
        Enemy.cs
        Group.cs
        GUIUpdater.cs
        Mario.cs
        MusicPlayer.cs
        SoundOutput.cs
        Trigger.cs
      Scenes
        About.cs
        GameOver.cs
        Level_1_1.cs
        MainMenu.cs
        MapBase.cs
        Test.cs
        WorldScreen.cs
      Scripts
        BlockAnimatorScript.cs
        CameraController.cs
        CharacterController.cs
        DeathZoneScript.cs
        DelayScript.cs
        EnemyControllerScripts.cs
        FloatingCoinAnimatorScript.cs
        FloatingTextAnimatorScript.cs
        GUIUpdateScript.cs
        MainMenuSpawnScript.cs
        MarioTriggerColliderScripts.cs
        MusicScript.cs
        PipePort.cs
        PowerUpScript.cs
        SpecialBlocksUpdateScript.cs
        WorldChangeManagerScript.cs
        WorldEnd.cs
      Sounds
        WAVs
        MarioJumpFX.mp3
        Overworld_Theme.mp3
        smb_1-up.mp3
        smb_breakblock.mp3
        smb_coin.mp3
        smb_jump-small.mp3
        smb_mariodie.mp3
        smb_pipe.mp3
        smb_powerup.mp3
        smb_stomp.mp3
        Sounds.resx
      Sprites
        Enemy
          goomba.gif
          goomba_dead.png
          mario_dead.png
          mario_jump_left.png
          mario_jump_right.png
          mario_left.png
          mario_move_left.gif
          mario_move_right.gif
          mario_right.png
          super_mario_crouch_left.png
          super_mario_crouch_right.png
          super_mario_jump_left.png
          super_mario_jump_right.png
          super_mario_left.png
          super_mario_move_left.gif
          super_mario_move_right.gif
          super_mario_powerup_left.gif
          super_mario_powerup_right.gif
          super_mario_right.png
        Ground
          block_01.png
          block_02.png
          block_03.png
          block_04.png
          block_1_with_coin.gif
          block_nocoins.png
          bridge.png
          bush_01.png
          bush_02.png
          bush_03.png
          bush_small.png
          castle_big.png
          castle_small.png
          cloud_01.png
          cloud_02.png
          cloud_03.png
          fence.png
          finish_flag.png
          finish_pole.png
          mountain.png
          sky_07.png
          water_01.png
          water_02.png
        Miscellaneous
          1-UP.png
          coin.gif
          coin2.gif
          flower.gif
          heart.gif
          mushroom.png
          particles1.png
          particles2.png
          particles3.png
          particles4.png
          star.gif
        Pipes
          pipe_01.png
          pipe_02.png
          pipe_03.png
          pipe_04.png
          pipe_05.png
        PSD
          1-UP.psd
          blocks_project.psd
          coin.psd
          coin2.psd
          Flower.psd
          Goomba.psd
          heart.psd
          icon.psd
          Mario.psd
          Mario_super.psd
          Mushroom.psd
          particles.psd
          star.psd
        UnderGround
          background_01.png
          background_02.png
          underground_block_01.png
          underground_block_02.png
          underground_block_03.png
          underground_block_04.png
        Enemies.resx
          Enemies.Designer.cs
        Sprites.resx
          Sprites.Designer.cs
      Logo_150x150.ico
      Logo_150x150.png
    App.config
    packages.config
    Program.cs
    Shared.cs
    SystemExt.cs
```