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.

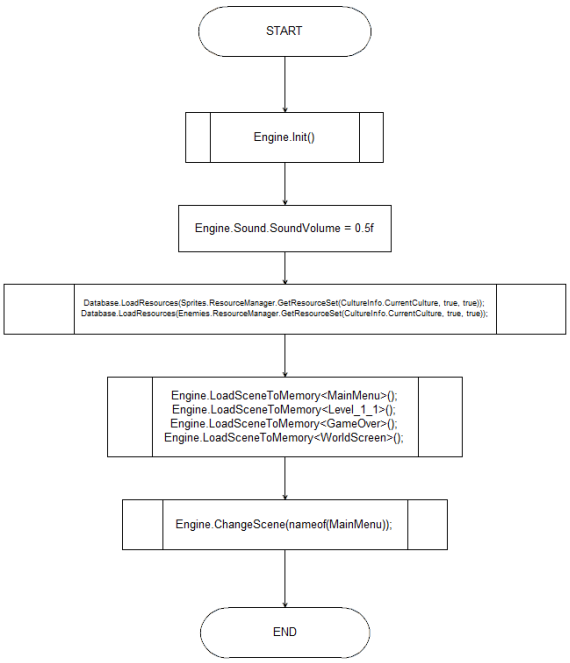
Program dále využívá knihovny NAudio. Jedná se o knihovnu zaměřenou na přehrávání audio záznamů. Pomocí této knihovny jsem vytvočil zvukový systém shopný přehrávat několik zvukových efektů zároveň v reálném čase.

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

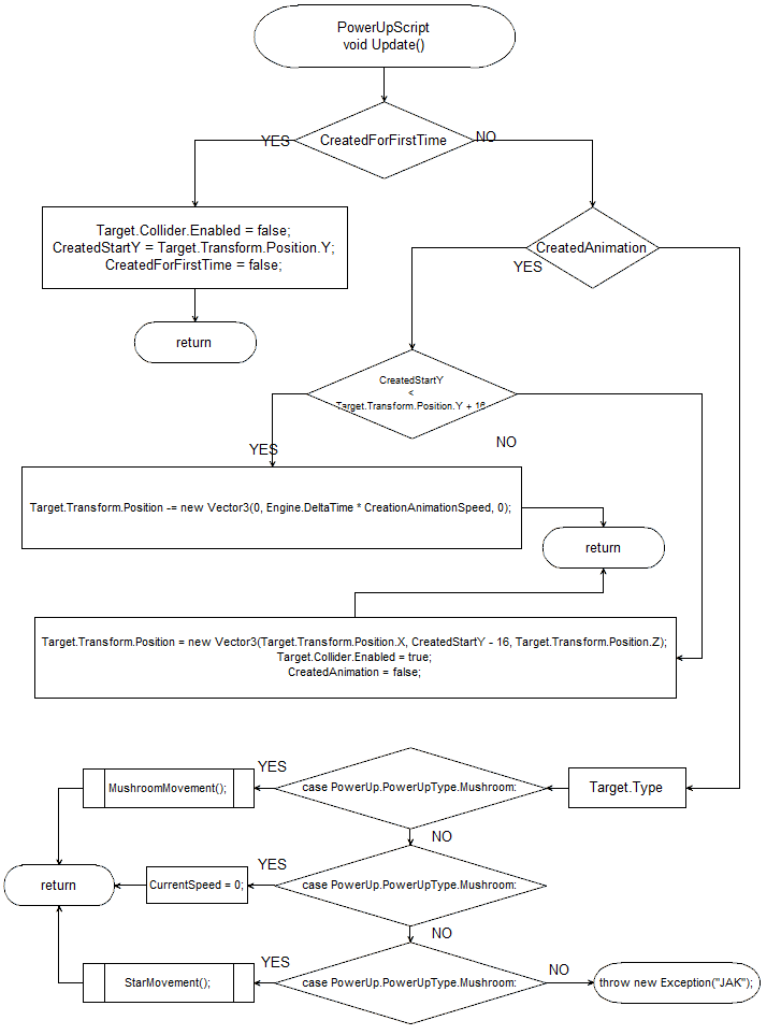
2 Stať

2.1 Vývojový diagram

2.1.1 Metoda „Main“



2.1.2 PowerUpScript – metoda Update



2.2 Zdrojový kód

2.2.1 DKEngine

2.2.1.1 Engine.cs

```
1   /**
2   * (C) 2017 David Knieradl
3   *
4   * For the brave souls who get this far: You are the chosen ones,
5   * the valiant knights of programming who toil away, without rest,
6   * fixing our most awful code. To you, true saviors, kings of men,
7   * I say this: never gonna give you up, never gonna let you down,
8   * never gonna run around and desert you. Never gonna make you cry,
9   * never gonna say goodbye. Never gonna tell a lie and hurt you.
10  */
11
12  using DKEngine.Core;
13  using DKEngine.Core.Components;
14  using DKEngine.Core.Ext;
15  using DKEngine.Core.UI;
16  using DKEngine.Data;
17  using System;
18  using System.Collections.Generic;
19  using System.Diagnostics;
20  using System.Drawing;
21  using System.Linq;
22  using System.Runtime.InteropServices;
23  using System.Threading;
24  using System.Threading.Tasks;
25
26  namespace DKEngine
27  {
28      /// <summary>
29      /// Engine class
30      /// </summary>
31      public static class Engine
32      {
33          /// <summary>
34          /// Sound subclass
35          /// </summary>
36          public static class Sound
37          {
38              /// <summary>
39              /// Enables sound
40              /// </summary>
41              public static bool IsSoundEnabled = true;
42
43              /// <summary>
44              /// Sets volume on sound inicialization
45              /// </summary>
46              public static float SoundVolume = 1f;
47
48              internal readonly static SoundPlayer Instance = new SoundPlayer();
49          }
50
51          /// <summary>
52          /// Render subclass
53          /// </summary>
54          public static class Render
55          {
56              /// <summary>
57              /// Sets resolution scale in %
58              /// </summary>
59              public const int ResolutionScale = 50;
60
61              /// <summary>
62              /// The resolution ratio
63              /// </summary>
64              public const float ResolutionRatio = ResolutionScale / 100f;
65
66              /// <summary>
67              /// The rendered image width
68              /// </summary>
69              public const int RenderWidth = (int)(640 * ResolutionRatio);
```

```
70
71              /// <summary>
72              /// The rendered image height
73              /// </summary>
74              public const int RenderHeight = (int)(480 * ResolutionRatio);
75
76              internal const int ImageBufferSize = 3 * RenderWidth * RenderHeight;
77              internal const int ImageKeyBufferSize = RenderWidth * RenderHeight;
78
79              internal static byte[] imageBuffer;
80              internal static byte[] imageBufferKey;
81              internal static byte[] ImageOutData;
82
83              internal static bool AbortRender = false;
84
85              internal const int Limiter = 1000;
86          }
87
88          /// <summary>
89          /// Input subclass
90          /// </summary>
91          public static class Input
92          {
93              [DllImport("user32.dll")]
94              private static extern ushort GetKeyState(short nVirtKey);
95
96              private const ushort keyDownBit = 0x80;
97
98              internal static bool[] KeysPressed;
99              internal static bool[] KeysDown;
100             internal static bool[] KeysReleased;
101             internal static bool[] KeysUp;
102
103             internal static short NumberOfKeys;
104
105             /// <summary>
106             /// Determines whether [is key pressed] [the specified key].
107             /// </summary>
108             /// <param name="key">The key</param>
109             /// <returns>
110             ///   <c>true</c> if [is key pressed] [the specified key]; otherwise, <c>false</c>.
111             /// </returns>
112             public static bool IsKeyPressed(ConsoleKey key)
113             {
114                 return KeysPressed[(short)key];
115             }
116
117             /// <summary>
118             /// Determines whether [is key down] [the specified key].
119             /// </summary>
120             /// <param name="key">The key</param>
121             /// <returns>
122             ///   <c>true</c> if [is key down] [the specified key]; otherwise, <c>false</c>.
123             /// </returns>
124             public static bool IsKeyDown(ConsoleKey key)
125             {
126                 return KeysDown[(short)key];
127             }
128
129             /// <summary>
130             /// Determines whether [is key up] [the specified key].
131             /// </summary>
132             /// <param name="key">The key</param>
133             /// <returns>
134             ///   <c>true</c> if [is key up] [the specified key]; otherwise, <c>false</c>.
135             /// </returns>
136             public static bool IsKeyUp(ConsoleKey key)
137             {
138                 return KeysUp[(short)key];
139             }
140
141             /// <summary>
142             /// Determines whether [is key released] [the specified key].
143             /// </summary>
144             /// <param name="key">The key</param>
145             /// <returns>
146             ///   <c>true</c> if [is key released] [the specified key]; otherwise, <c>false</c>.
```

```
147             /// </returns>
148             public static bool IsKeyReleased(ConsoleKey key)
149             {
150                 return KeysReleased[(short)key];
151             }
152
153             internal static void CheckForKeys()
154             {
155                 for (int key = 0; key < NumberOfKeys; key++)
156                 {
157                     bool IsDown = ((GetKeyState((short)key) & keyDownBit) == keyDownBit);
158
159                     if (IsDown)
160                     {
161                         if (!KeysDown[key])
162                         {
163                             KeysUp[key] = false;
164                             KeysReleased[key] = false;
165                             KeysPressed[key] = true;
166                             KeysDown[key] = true;
167                         }
168                         else if (KeysPressed[key])
169                         {
170                             KeysPressed[key] = false;
171                         }
172                     }
173                     else
174                     {
175                         if (KeysDown[key])
176                         {
177                             KeysPressed[key] = false;
178                             KeysDown[key] = false;
179                             KeysReleased[key] = true;
180                             KeysUp[key] = true;
181                         }
182                         else if (KeysReleased[key])
183                         {
184                             KeysReleased[key] = false;
185                         }
186                     }
187                 }
188             }
189         }
190
191         private static bool _IsInitialised = false;
192
193         private static Thread BackgroundWorks;
194         private static TextBlock FpsMeter;
195         private static Stopwatch DeltaT;
196         internal static Camera BaseCam;
197
198         internal static Scene CurrentScene { get; set; }
199         internal static Scene LoadingScene { get; set; }
200
201         internal static Type LoadingSceneType { get; set; }
202
203         internal static List<GameObject> RenderObjects;
204
205         private static float deltaT = 0;
206         public static float DeltaTime { get { return deltaT; } }
207
208         private static TimeSpan lastUpdated = new TimeSpan();
209         public static TimeSpan LastUpdated { get { return lastUpdated; } }
210
211         public static string SceneName { get { return Engine.LoadingScene != null ? Engine.LoadingScene.Name : ""; } }
212
213         private static readonly TimeSpan _firstTimeLoadDelay = new TimeSpan(0, 0, 1);
214         private static TimeSpan FirstTimeLoadDelay = new TimeSpan();
215         private static bool FirstTimeLoaded = true;
216
217         internal static event EngineHandler UpdateEvent;
218
219         internal delegate void EngineHandler();
220
221         /// <summary>
222         /// Sets engine to work.
223         /// </summary>
```

```
224         /// <exception cref="System.Exception">
225         /// Engine initialisation failed\n" + e
226         /// or
227         /// Engine is being initialised second time
228         /// </exception>
229         public static void Init()
230         {
231             if (!_IsInitialised)
232             {
233                 try
234                 {
235                     WindowControl.WindowInit();
236                     Database.InitDatabase();
237 
238                     Render.imageBuffer = new byte[Render.ImageBufferSize];
239                     Render.imageBufferKey = new byte[Render.ImageKeyBufferSize];
240                     Render.ImageOutData = new byte[Render.ImageBufferSize];
241 
242                     Input.NumberOfKeys = (short)Enum.GetNames(typeof(ConsoleKey)).Length;
243                     Input.KeysPressed = new bool[Input.NumberOfKeys];
244                     Input.KeysDown = new bool[Input.NumberOfKeys];
245                     Input.KeysUp = new bool[Input.NumberOfKeys];
246                     Input.KeysReleased = new bool[Input.NumberOfKeys];
247 
248                     DeltaT = Stopwatch.StartNew();
249 
250                     RenderObjects = new List<GameObject>(0xFFFF);
251 
252                     //Sound.OutputDevice = new WaveOut();
253 
254                     FpsMeter = new TextBlock();
255                     FpsMeter.Transform.Position = new Vector3(4, -4, 128);
256                     FpsMeter.Transform.Dimensions = new Vector3(50, 5, 1);
257                     FpsMeter.VAlignment = Text.VerticalAlignment.Bottom;
258                     FpsMeter.HAlignment = Text.HorizontalAlignment.Left;
259                     FpsMeter.Text = "0";
260                     FpsMeter.IsGUI = true;
261                     FpsMeter.TextShadow = true;
262                     FpsMeter.Foreground = Color.FromArgb(0xFF, 0x00, 0xFF, 0xFF);
263 
264                     FpsMeter.InitInternal();
265 
266                     UpdateEvent += FpsMeter.Scripts[0].UpdateHandle;
267 
268                     BackgroundWorks = new Thread(Update);
269                     //RenderWorker    = new Thread(RenderImage);
270                     BackgroundWorks.Start();
271                     //RenderWorker.Start();
272 
273 #if !DEBUG
274                     SplashScreen();
275 #endif
276 
277                     _IsInitialised = true;
278                 }
279                 catch (Exception e)
280                 {
281                     throw new Exception("Engine initialisation failed\n" + e);
282                 }
283             }
284             else
285                 throw new Exception("Engine is being initialised second time");
286         }
287 
288         public static void LoadSceneToMemory<T>(object[] argsPreLoad = null, object[] argsPostLoad = null)
289             where T : Scene
290         {
291             Engine.LoadingScene = (T)Activator.CreateInstance(typeof(T));
292             Engine.LoadingScene.argsPreLoad = argsPreLoad;
293             Engine.LoadingScene.argsPostLoad = argsPostLoad;
294             Engine.LoadingScene.Set(argsPreLoad);
295             Engine.LoadingScene.Init();
296 
297             Database.AddScene(Engine.LoadingScene);
298         }
299 
300         /// <summary>
```

```
301         /// Loads the scene to memory.
302         /// </summary>
303         /// <typeparam name="T">Scene</typeparam>
304         /// <param name="argsPreLoad">The arguments pre load.</param>
305         /// <param name="argsPostLoad">The arguments post load.</param>
306         public static void LoadScene<T>(object[] argsPreLoad = null, object[] argsPostLoad = null) where T : Scene
307         {
308             LoadingSceneType = typeof(T);
309             LoadScene(LoadingSceneType, argsPreLoad, argsPostLoad);
310         }
311
312         public static void LoadScene(Type scene, object[] argsPreLoad = null, object[] argsPostLoad = null)
313         {
314             if (!scene.IsSubclassOf(typeof(Scene)))
315                 throw new Exception($"Provided type {scene} is not subclass of Scene");
316
317             Engine.LoadingScene = (Scene)Activator.CreateInstance(LoadingSceneType);
318
319             Engine.LoadingScene.argsPreLoad = argsPreLoad;
320             Engine.LoadingScene.argsPostLoad = argsPostLoad;
321
322             Engine.LoadingScene.Set(argsPreLoad);
323             Engine.LoadingScene.Init();
324
325             UnregisterScene();
326             RegisterScene(Engine.LoadingScene, argsPostLoad);
327         }
328
329         /// <summary>
330         /// Reloads the scene.
331         /// </summary>
332         /// <param name="Name">The name of scene</param>
333         public static void ReloadScene(string Name, object[] argsPreLoad = null)
334         {
335             Database.RewriteWorld(Name, argsPreLoad);
336         }
337
338         /// <summary>
339         /// Changes the scene.
340         /// </summary>
341         /// <param name="Name">The name</param>
342         /// <param name="Reload">if set to <c>true</c> [reload]</param>
343         /// <param name="args">The arguments</param>
344         public static void ChangeScene(string Name, bool Reload = false, object[] argsPreLoad = null, object[] argsPost-
    Load = null)
345         {
346             UnregisterScene();
347             if (Reload)
348             {
349                 ReloadScene(Name, argsPreLoad);
350
351                 if (argsPostLoad != null)
352                     Database.GetScene(Name).argsPostLoad = argsPostLoad;
353             }
354
355             RegisterScene(Database.GetScene(Name), argsPostLoad);
356         }
357
358         private static void UnregisterScene()
359         {
360             try
361             {
362                 Engine.CurrentScene.Unload();
363
364                 foreach (var item in CurrentScene.AllBehaviors)
365                 {
366                     try
367                     {
368                         UpdateEvent -= item.UpdateHandle;
369                     }
370                     catch { }
371                 }
372
373                 while (CurrentScene.GameObjectsAddedToRender.Count > 0)
374                 {
375                     GameObject tmp = CurrentScene.GameObjectsAddedToRender.Pop();
376                     if (Engine.RenderObjects.Contains(tmp))
```

```
377                     Engine.RenderObjects.Remove(tmp);
378
379                 CurrentScene.GameObjectsToAddToRender.Push(tmp);
380             }
381         }
382         catch { }
383     }
384
385     private static void RegisterScene(Scene source, object[] args)
386     {
387         Engine.LoadingScene = source;
388
389         if (args != null)
390         {
391             source.argsPostLoad = args;
392             source.Set(args);
393         }
394         else if (source.argsPostLoad != null)
395             source.Set(source.argsPostLoad);
396
397         foreach (var item in source.AllBehaviors)
398         {
399             try
400             {
401                 UpdateEvent += item.UpdateHandle;
402             }
403             catch { }
404         }
405
406         Engine.BaseCam = Engine.LoadingScene.BaseCamera;
407         Engine.CurrentScene = source;
408     }
409
410     public static void ReloadCurrentScene()
411     {
412         UnregisterScene();
413         LoadScene(LoadingSceneType);
414     }
415
416     private static void SplashScreen()
417     {
418         if (!_IsInitialised)
419         {
420             Engine.LoadScene<SplashScreenScene>();
421
422             SpinWait.SpinUntil(() => ((SplashScreenScene)Engine.CurrentScene).Splash.Animator.NumberOfPlays >= 1);
423         }
424     }
425
426     private static void Update()
427     {
428         Task imageRender = Task.Factory.StartNew(RenderImage);
429
430         int NumberOfFrames = 0;
431         TimeSpan timeOut = new TimeSpan(0, 0, 0, 0, 500);
432         Stopwatch time = Stopwatch.StartNew();
433         Stopwatch fpsLimiter = Stopwatch.StartNew();
434
435         while (true)
436         {
437             Input.CheckForKeys();
438
439             lastUpdated += DeltaT.Elapsed;
440             deltaT = (float)DeltaT.Elapsed.TotalSeconds;
441             DeltaT?.Restart();
442
443             if (!FirstTimeLoaded)
444             {
445                 UpdateEvent?.Invoke();
446
447                 while (Engine.CurrentScene?.NewlyGeneratedComponents.Count > 0)
448                 {
449                     Engine.CurrentScene.NewlyGeneratedComponents.Pop().InitInternal();
450                 }
451
452                 while (Engine.CurrentScene?.NewlyGeneratedBehaviors.Count > 0)
453                 {
```

```
454                     Behavior tmp = Engine.CurrentScene.NewlyGeneratedBehaviors.Pop();
455                     UpdateEvent += tmp.UpdateHandle;
456                     tmp.Start();
457                 }
458 
459                 while (Engine.CurrentScene?.DestroyObjectAwaitList.Count > 0)
460                 {
461                     GameObject tmp = Engine.CurrentScene.DestroyObjectAwaitList[0];
462                     Engine.CurrentScene.DestroyObjectAwaitList.RemoveAt(0);
463                     tmp.Destroy();
464                 }
465 
466                 while (Engine.CurrentScene?.GameObjectsToAddToRender.Count > 0)
467                 {
468                     GameObject tmp = Engine.CurrentScene.GameObjectsToAddToRender.Pop();
469                     Engine.RenderObjects.Add(tmp);
470                     Engine.CurrentScene.GameObjectsAddedToRender.Push(tmp);
471                 }
472 
473                 List<GameObject> reference = Engine.RenderObjects.GetGameObjectsInView();
474 
475                 if (Engine.CurrentScene != null)
476                 {
477                     List<Collider> VisibleTriggers = Engine.CurrentScene?.AllGameObjectsColliders.Where(obj => obj.Is-
    Trigger).ToList();
478                     List<Collider> VisibleColliders = Engine.CurrentScene?.AllGameObjectsColliders.Where(obj => !obj.Is-
    Trigger).ToList();
479                     int ColliderCount = VisibleTriggers.Count;
480                     for (int i = 0; i < ColliderCount; i++)
481                         VisibleTriggers[i]?.TriggerCheck(VisibleColliders);
482                 }
483 
484                 Engine.CurrentScene?.BaseCamera?.BufferImage(reference);
485 
486                 Buffer.BlockCopy(Render.imageBuffer, 0, Render.ImageOutData, 0, Render.ImageBufferSize);
487             }
488             else
489             {
490                 FirstTimeLoadDelay += new TimeSpan(0, 0, 0, 0, (int)(DeltaTime * 1000));
491 
492                 if (FirstTimeLoadDelay > _firstTimeLoadDelay)
493                 {
494                     FirstTimeLoaded = false;
495                     FirstTimeLoadDelay = new TimeSpan();
496                 }
497             }
498 
499             NumberOfFrames++;
500 
501             Vsync(Render.Limiter, (int)fpsLimiter.ElapsedMilliseconds);
502             fpsLimiter.Restart();
503 
504             if (time.ElapsedMilliseconds > timeOut.TotalMilliseconds)
505             {
506                 long t = NumberOfFrames * 1000 / time.ElapsedMilliseconds;
507                 FpsMeter.Text = t.ToString();
508                 /*#if DEBUG
509                         Debug.WriteLine(t);
510                 #endif*/
511                 time.Restart();
512                 NumberOfFrames = 0;
513             }
514         }
515     }
516 
517     private static async void RenderImage()
518     {
519         IntPtr ConsoleWindow = GetConsoleWindow();
520 
521         using (Graphics g = Graphics.FromHwnd(ConsoleWindow))
522         {
523             g.CompositingQuality = System.Drawing.Drawing2D.CompositingQuality.HighSpeed;
524             g.PixelOffsetMode = System.Drawing.Drawing2D.PixelOffsetMode.HighSpeed;
525             g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.None;
526             g.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.NearestNeighbor;
527 
528             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

```
1   namespace DKEngine.Core.Components
2   {
3       /// <summary>
4       /// Node used in Animator Component
5       /// </summary>
6       /// <seealso cref="DKEngine.Core.Components.Component" />
7       public sealed class AnimationNode : Component
8       {
9           public Material Animation = null;
10          public bool IsLoop = false;
11
12          private AnimationNode()
13              : base(null)
14          { }
15
16          public AnimationNode(string Name, Material Source)
17              : base(null)
18          {
19              this.Name = Name;
20              this.Animation = Source;
21          }
22
23          public override void Destroy()
24          { }
25      }
26  }
```

2.2.1.3 Core/Components/Animator.cs

```
1   /*
2   * (C) 2017 David Knieradl
3   */
4
5   using System;
6   using System.Collections.Generic;
7   using System.Diagnostics;
8   using System.Linq;
9
10  namespace DKEngine.Core.Components
11  {
12      /// <summary>
13      /// Used for GameObject material animation
14      /// </summary>
15      /// <seealso cref="DKEngine.Core.Components.Behavior" />
16      /// <seealso cref="DKEngine.IAnimated" />
17      public class Animator : Behavior, IAnimated
18      {
19          public TimeSpan CurrentAnimationTime;
20          internal Dictionary<string, AnimationNode> Animations;
21          private AnimationNode _current;
22
23          public int NumberOfPlays { get; private set; } = 0;
24
25          public AnimationNode Current
26          {
27              get { return _current; }
28              set
29              {
30                  if (value != _current)
31                  {
32                      _current = value;
33                      Parent.Model = _current.Animation;
34                      NumberOfPlays = 0;
35                      CurrentAnimationTime = new TimeSpan(0);
36                  }
37              }
38          }
39
40          public int AnimationState
41          {
42              get
43              {
44                  return (int)(CurrentAnimationTime.TotalMilliseconds / Parent.Model.DurationPerFrame % Parent.Model.Fra-
    mes);
45              }
```

```
46          }
47
48          public Animator(GameObject Parent)
49              : base(Parent)
50          {
51              this.CurrentAnimationTime = new TimeSpan(0);
52              this.Animations = new Dictionary<string, AnimationNode>();
53
54              this.Name = string.Format("{0}_{1}", Parent.Name, nameof(Animator));
55          }
56
57          /// <summary>
58          /// Adds the animation.
59          /// </summary>
60          /// <param name="Name">The animation node name.</param>
61          /// <param name="Source">The source material for animation node.</param>
62          public void AddAnimation(string Name, Material Source)
63          {
64              Animations.Add(Name, new AnimationNode(Name, Source));
65              if (Animations.Count == 1)
66              {
67                  Play(Animations.ElementAt(0).Key);
68              }
69          }
70
71          /// <summary>
72          /// Adds the animation.
73          /// </summary>
74          /// <param name="Name">The animation node name.</param>
75          /// <param name="MaterialKey">The material key to search for material.</param>
76          public void AddAnimation(string Name, string MaterialKey)
77          {
78              Animations.Add(Name, new AnimationNode(Name, Database.GetGameObjectMaterial(MaterialKey)));
79              if (Animations.Count == 1)
80              {
81                  Play(Animations.ElementAt(0).Key);
82              }
83          }
84
85          /// <summary>
86          /// Plays the specified animation name.
87          /// </summary>
88          /// <param name="AnimationName">Name of the animation.</param>
89          public void Play(string AnimationName)
90          {
91              if (AnimationName != Current?.Name)
92              {
93                  AnimationNode Result;
94
95                  try
96                  {
97                      Result = Animations[AnimationName];
98                  }
99                  catch (Exception e)
100                 {
101                     Debug.WriteLine("Animation \"{0}\"not found\n{1}", AnimationName, e);
102                     return;
103                 }
104
105                 Current = Result;
106             }
107         }
108
109         protected internal override void Update()
110         {
111             if (Parent?.Model?.Frames > 1)
112             {
113                 CurrentAnimationTime = CurrentAnimationTime.Add(new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime *
    1000)));
114
115                 if (CurrentAnimationTime.TotalMilliseconds > Parent.Model.Duration)
116                 {
117                     CurrentAnimationTime = CurrentAnimationTime.Subtract(new TimeSpan(0, 0, 0, 0, Parent.Model.Duration));
118                     NumberOfPlays++;
119                 }
120             }
121         }
```

```
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
```

```
15      /// </summary>
16      /// <seealso cref="DKEngine.Core.Components.Component" />
17      public sealed class Camera : Component
18      {
19          /// <summary>
20          /// Sets the canvas background color
21          /// </summary>
22          public Color BackGround = Color.Black;
23
24          /// <summary>
25          /// The offset position of camera
26          /// </summary>
27          public Vector3 Position;
28
29          internal float X { get { return RenderingGUI ? 0 : Parent != null ? Parent.Transform.Position.X + Position.X : Position.X; } }
30          internal float Y { get { return RenderingGUI ? 0 : Parent != null ? Parent.Transform.Position.Y + Position.Y : Position.Y; } }
31
32          private bool RenderingGUI = false;
33
34          public Camera()
35              : base(null)
36          {
37              this.Position = new Vector3(0, 0, 0);
38              Engine.LoadingScene.BaseCamera = this;
39
40              this.Name = string.Format("{0}", nameof(Camera));
41          }
42
43          public Camera(GameObject Parent)
44              : base(Parent)
45          {
46              Engine.LoadingScene.BaseCamera = this;
47
48              this.Name = string.Format("{0}_{1}", Parent.Name, nameof(Camera));
49          }
50
51          internal void BufferImage(List<GameObject> GameObjectsInView)
52          {
53              BackGroundInit();
54              RenderingGUI = true;
55              List<GameObject> GUI = GameObjectsInView.Where(item => item.IsGUI).ToList();
56              int GUICount = GUI.Count;
57              for (int i = 0; i < GUICount; i++)
58                  GameObjectsInView.Remove(GUI[i]);
59
60              while (GUICount > 0)
61              {
62                  float tempHeight = GUI.FindMaxZ();
63                  GameObject[] toRender = GUI.Where(item => item.Transform.Position.Z == tempHeight).ToArray();
64
65                  int toRenderCount = toRender.Length;
66                  for (int i = toRenderCount - 1; i >= 0; i--)
67                  {
68                      toRender[i].Render();
69                      GUI.Remove(toRender[i]);
70                      GUICount--;
71                  }
72              }
73
74              RenderingGUI = false;
75              int TempCount = GameObjectsInView.Count;
76
77              while (TempCount > 0)
78              {
79                  float tempHeight = GameObjectsInView.FindMaxZ();
80                  GameObject[] toRender = GameObjectsInView.Where(item => item.Transform.Position.Z == tempHeight).ToArray();
81
82                  int toRenderCount = toRender.Length;
83                  for (int i = toRenderCount - 1; i >= 0; i--)
84                  {
85                      toRender[i].Render();
86                      GameObjectsInView.Remove(toRender[i]);
87                      TempCount--;
88                  }
```

```
89              }
90          }
91
92          private void BackGroundInit()
93          {
94              byte R = BackGround.R;
95              byte G = BackGround.G;
96              byte B = BackGround.B;
97
98              int imageBufferLenght = Engine.Render.imageBuffer.Length;
99              for (int i = 0; i < imageBufferLenght; i += 3)
100             {
101                 Engine.Render.imageBuffer[i + 2] = R;
102                 Engine.Render.imageBuffer[i + 1] = G;
103                 Engine.Render.imageBuffer[i] = B;
104             }
105
106             Array.Clear(Engine.Render.imageBufferKey, 0, Engine.Render.imageBufferKey.Length);
107         }
108
109         public sealed override void Destroy()
110         {
111             if (Engine.BaseCam == this)
112                 Engine.BaseCam = null;
113
114             try
115             {
116                 Engine.LoadingScene.AllComponents.Remove(this.Name);
117             }
118             catch
119             { }
120
121             Parent = null;
122         }
123     }
124 }
```

2.2.1.6 Core/Components/Collider.cs

```
1   /*
2   * (C) 2017 David Knieradl
3   */
4
5   using System;
6   using System.Collections.Generic;
7   using System.Diagnostics;
8   using System.Drawing;
9   using static DKEngine.Core.Components.Transform;
10
11  namespace DKEngine.Core.Components
12  {
13      /// <summary>
14      /// Collider used for GameObjects
15      /// </summary>
16      /// <seealso cref="DKEngine.Core.Components.Component" />
17      public class Collider : Component
18      {
19          internal event CollisionEnterHandler CollisionEvent;
20
21          internal delegate void CollisionEnterHandler(Collider m);
22
23          /// <summary>
24          /// Determines size and position of collider
25          /// </summary>
26          public RectangleF Area = new RectangleF();
27
28          /// <summary>
29          /// If is TRUE => Triggers OnColliderEnter once another GameObject enter this collider
30          /// </summary>
31          public bool IsTrigger = false;
32
33          /// <summary>
34          /// Collider is enabled or disabled
35          /// </summary>
36          public bool Enabled = true;
37
38          private float X { get { return Parent.Transform.Position.X + Area.X; } }
```

```
39          private float Y { get { return Parent.Transform.Position.Y + Area.Y; } }
40          private float Width { get { return Parent.Transform.Scale.X * Area.Width; } }
41          private float Height { get { return Parent.Transform.Scale.Y * Area.Height; } }
42
43          private bool _Right;
44          private bool _Left;
45          private bool _Top;
46          private bool _Bottom;
47
48          /// <summary>
49          /// Creates new Instance of Collider class
50          /// </summary>
51          /// <param name="Parent">Parent of collider (determines size of collider)</param>
52          internal Collider(GameObject Parent)
53              : base(Parent)
54          {
55              this.Area = new RectangleF(0, 0, Parent.Transform.Dimensions.X, Parent.Transform.Dimensions.Y);
56
57              this.Name = string.Format("{0}_{1}", Parent.Name, nameof(Collider));
58          }
59
60      #if DEBUG
61
62          /// <summary>
63          /// Returns string containing <b>bool</b> value for each of the directions of this object.
64          /// </summary>
65          /// <returns></returns>
66          public string DebugTestCollision()
67          {
68              return string.Format("Left {0}\nRight {1}\nTop {2}\nDown {3}", Collision(Direction.Left), Collision(Direction.Right),
    Collision(Direction.Up), Collision(Direction.Down));
69          }
70
71      #endif
72
73          /// <summary>
74          /// Collision check in specified direction.
75          /// </summary>
76          /// <param name="direction"></param>
77          /// <returns></returns>
78          public bool Collision(Direction direction)
79          {
80              if (this.IsTrigger || !this.Enabled)
81                  return false;
82
83              if (LastUpdated != Engine.LastUpdated)
84              {
85                  _Right = false;
86                  _Left = false;
87                  _Bottom = false;
88                  _Top = false;
89
90                  int count = Engine.CurrentScene.AllGameObjectsColliders.Count;
91                  for (int i = 0; i < count; i++)
92                  {
93                      Collider tmp = Engine.CurrentScene.AllGameObjectsColliders[i];
94
95                      bool _L = false;
96                      bool _R = false;
97                      bool _T = false;
98                      bool _B = false;
99
100                     float _LeftSpan = float.MaxValue;
101                     float _RightSpan = float.MaxValue;
102                     float _BottomSpan = float.MaxValue;
103                     float _TopSpan = float.MaxValue;
104
105                     if (_L = Left(tmp))
106                     {
107                         _LeftSpan = LeftSpan(tmp);
108                     }
109
110                     if (_R = Right(tmp))
111                     {
112                         _RightSpan = RightSpan(tmp);
113                     }
114
```

```
                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;
            }
```

```
192             }
193         }
194
195         private float LeftSpan(Collider obj)
196         {
197             return obj.X + obj.Width - this.X;
198         }
199
200         private float TopSpan(Collider obj)
201         {
202             return obj.Y + obj.Height - this.Y;
203         }
204
205         private float RightSpan(Collider obj)
206         {
207             return this.X + this.Width - obj.X;
208         }
209
210         private float BottomSpan(Collider obj)
211         {
212             return this.Y + this.Height - obj.Y;
213         }
214
215         private bool Left(Collider obj)
216         {
217             try
218             {
219                 if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
220                     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);
221             }
222             catch { }
223
224             return false;
225         }
226
227         private bool Right(Collider obj)
228         {
229             try
230             {
231                 if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
232                     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);
233             }
234             catch { }
235
236             return false;
237         }
238
239         private bool Up(Collider obj)
240         {
241             try
242             {
243                 if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
244                     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);
245             }
246             catch { }
247
248             return false;
249         }
250
251         private bool Down(Collider obj)
252         {
253             try
254             {
255                 if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
256                     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);
257             }
258             catch { }
259
260             return false;
```

```
261         }
262
263         private bool Collided(Collider obj)
264         {
265             try
266             {
267                 if (!this.Equals(obj) && !obj.IsTrigger && obj.Enabled)
268                     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);
269             }
270             catch { }
271
272             return false;
273         }
274
275         public override void Destroy()
276         {
277             try
278             {
279                 Engine.CurrentScene.AllGameObjectsColliders.Remove(this);
280             }
281             catch { }
282
283             try
284             {
285                 Engine.CurrentScene.AllComponents.Remove(this.Name);
286             }
287             catch
288             { }
289
290             if (Parent.Collider == this)
291                 Parent.Collider = null;
292         }
293
294         public void SetCollisionManually(Direction direction)
295         {
296             switch (direction)
297             {
298                 case Direction.Up:
299                     _Top = true;
300                     break;
301
302                 case Direction.Left:
303                     _Left = true;
304                     break;
305
306                 case Direction.Down:
307                     _Bottom = true;
308                     break;
309
310                 case Direction.Right:
311                     _Right = true;
312                     break;
313
314                 default:
315                     break;
316             }
317         }
318
319         internal sealed override void Init()
320         {
321             try
322             {
323                 Engine.LoadingScene.AllGameObjectsColliders.Add(this);
324             }
325             catch (Exception e)
326             {
327                 Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
328             }
329         }
330     }
331 }
```

2.2.1.7 Core/Components/Component.cs

```
1   using DKEngine.Core.Ext;
2   using DKEngine.Core.UI;
```

```
3   using System;
4   using System.Diagnostics;
5
6   namespace DKEngine.Core.Components
7   {
8       /// <summary>
9       /// Base class for all objects using DKEngine library
10      /// </summary>
11      public abstract class Component
12      {
13          private TimeSpan _lastUpdated;
14
15          internal TimeSpan LastUpdated
16          {
17              get
18              {
19                  TimeSpan tmp = _lastUpdated;
20                  _lastUpdated = Engine.LastUpdated;
21                  return tmp;
22              }
23          }
24
25          /// <summary>
26          /// The parent object of this instance
27          /// </summary>
28          public GameObject Parent = null;
29
30          /// <summary>
31          /// The name of this instance
32          /// </summary>
33          public string Name = "";
34
35          internal Component(GameObject Parent)
36          {
37              this.Parent = Parent;
38              _lastUpdated = Engine.LastUpdated;
39
40              try
41              {
42                  Engine.LoadingScene.NewlyGeneratedComponents.Push(this);
43              }
44              catch (Exception e)
45              {
46                  Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
47              }
48          }
49
50          internal void InitInternal()
51          {
52              Init();
53
54              try
55              {
56                  if (this.GetType() != typeof(Letter))
57                  {
58                      Engine.LoadingScene.AllComponents.AddSafe(this);
59                  }
60              }
61              catch (Exception e)
62              {
63                  Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
64              }
65          }
66
67          internal virtual void Init()
68          { }
69
70          public abstract void Destroy();
71
72          /// <summary>
73          /// Finds the specified component of specified name.
74          /// </summary>
75          /// <typeparam name="T">Determines type of desired component</typeparam>
76          /// <param name="Name">The name of desired component.</param>
77          /// <returns></returns>
78          public static T Find<T>(string Name) where T : Component
79          {
```

```
80              T retValue = null;
81
82              try
83              {
84                  retValue = (T)Engine.LoadingScene.AllComponents[Name];
85              }
86              catch (Exception ex)
87              {
88                  Debug.WriteLine("Object not found\n" + ex);
89              }
90
91              return retValue;
92          }
93      }
94  }
```

2.2.1.8 Core/Components/Material.cs

```
1   /*
2   * (C) 2017 David Knieradl
3   */
4
5   using System;
6   using System.Drawing;
7   using System.Drawing.Imaging;
8   using System.Runtime.InteropServices;
9
10  namespace DKEngine.Core.Components
11  {
12      /// <summary>
13      /// Low-Memory Material
14      /// </summary>
15      public sealed class Material
16      {
17          /// <summary>
18          /// Source image used as Texture
19          /// </summary>
20          public readonly Bitmap Texture = null;
21
22          /// <summary>
23          /// Represents scaled length of image in pixels
24          /// </summary>
25          public readonly int Width = 0;
26
27          /// <summary>
28          /// Represents scaled height of image in pixels
29          /// </summary>
30          public readonly int Height = 0;
31
32          /// <summary>
33          /// Number of frames
34          /// </summary>
35          public readonly int Frames = 1;
36
37          /// <summary>
38          /// Total duration of animated image
39          /// </summary>
40          public readonly int Duration = 1;
41
42          /// <summary>
43          /// Duration between two frames of animation
44          /// </summary>
45          public readonly int DurationPerFrame = 1;
46
47          /// <summary>
48          /// Returns true if image is animated
49          /// </summary>
50          public readonly bool IsAnimated = false;
51
52          /// <summary>
53          /// Returns true if image is looped
54          /// </summary>
55          public readonly bool IsLooped = false;
56
57          private int _SelectedLayer = -1;
58          private FrameDimension _FrameDim = null;
59          private BitmapData _BitmapData = null;
```

```
60          private byte[] _Data = null;
61          private byte _BytesPerPixel = 0;
62
63          /// <summary>
64          /// Loads image and creates new material
65          /// </summary>
66          /// <param name="source">Source image</param>
67          public Material(Image source)
68          {
69              if (source != null)
70              {
71                  _FrameDim = new FrameDimension(source.FrameDimensionsList[0]);
72                  Frames = source.GetFrameCount(_FrameDim);
73
74                  Texture = (Bitmap)source;
75
76                  Width = source.Width;
77                  Height = source.Height;
78
79                  if (ImageAnimator.CanAnimate(source))
80                  {
81                      int delay = 0;
82                      int this_delay = 0;
83                      int index = 0;
84
85                      for (int i = 0; i < Frames; i++)
86                      {
87                          this_delay = BitConverter.ToInt32(source.GetPropertyItem(20736).Value, index) * 10;
88                          delay += (this_delay < 1 ? 33 : this_delay);
89                          index += 4;
90                      }
91
92                      Duration = delay;
93                      DurationPerFrame = Duration / Frames;
94                      IsAnimated = true;
95                      IsLooped = BitConverter.ToInt16(source.GetPropertyItem(20737).Value, 0) != 1;
96                  }
97
98                  switch (source.PixelFormat)
99                  {
100                     case PixelFormat.Format32bppArgb:
101                         _BytesPerPixel = 4;
102                         break;
103
104                     case PixelFormat.Format24bppRgb:
105                         _BytesPerPixel = 3;
106                         break;
107
108                     default:
109                         throw new Exception("Unsupported");
110                 }
111
112                 _Data = new byte[Width * Height * _BytesPerPixel];
113             }
114         }
115
116         /// <summary>
117         /// Creates new material with given color and scales it by parent's given scales
118         /// </summary>
119         /// <param name="clr">Source color</param>
120         /// <param name="Size">Vector3 used for material size</param>
121         public Material(Color clr, Vector3 Size)
122         {
123             this.Width = (int)(Size.X < 1 ? 1 : Size.X);
124             this.Height = (int)(Size.Y < 1 ? 1 : Size.Y);
125
126             Frames = 1;
127
128             _BytesPerPixel = 4;
129
130             int size = Width * Height * _BytesPerPixel;
131             _Data = new byte[size];
132
133             for (int index = 0; index < size; index += _BytesPerPixel)
134             {
135                 _Data[index + 3] = clr.A;
136                 _Data[index + 2] = clr.R;
```

```
137                 _Data[index + 1] = clr.G;
138                 _Data[index] = clr.B;
139             }
140
141             unsafe
142             {
143                 fixed (byte* data = _Data)
144                 {
145                     using (Bitmap tmp = new Bitmap(Width,
146                                     Height,
147                                     Width * _BytesPerPixel,
148                                     PixelFormat.Format32bppArgb,
149                                     new IntPtr(data)))
150                     {
151                         Texture = new Bitmap(tmp);
152                     }
153                 }
154             }
155
156             _FrameDim = new FrameDimension(Texture.FrameDimensionsList[0]);
157         }
158
159         /// <summary>
160         /// Creates new material with given color and scales it by parent's given scales
161         /// </summary>
162         /// <param name="clr">Source color</param>
163         /// <param name="Parent">GameObject used for material size</param>
164         public Material(Color clr, GameObject Parent)
165             : this(clr, Parent.Transform.Dimensions)
166         { }
167
168         /// <summary>
169         /// Render material into engine image buffer
170         /// </summary>
171         /// <param name="Parent">I3Dimensional for coordiantions</param>
172         public void Render(GameObject Parent, Color? ReColor = null)
173         {
174             int AnimationState = Parent.Animator != null ? Parent.Animator.AnimationState : 0;
175             bool HasShadow = Parent.HasShadow;
176
177             if (_SelectedLayer != AnimationState)
178             {
179                 if (_BitmapData != null)
180                     Texture.UnlockBits(_BitmapData);
181                 Texture.SelectActiveFrame(_FrameDim, AnimationState);
182                 _BitmapData = Texture.LockBits(new Rectangle(0, 0, Width, Height), ImageLockMode.ReadOnly, Texture.Pi-
    xelFormat);
183                 Marshal.Copy(_BitmapData.Scan0, _Data, 0, _Data.Length);
184                 _SelectedLayer = AnimationState;
185             }
186
187             float CamX = Engine.BaseCam != null ? Engine.BaseCam.X : 0;
188             float CamY = Engine.BaseCam != null ? Engine.BaseCam.Y : 0;
189
190             int x = (int)(Parent.Transform.Position.X - CamX);
191             int y = (int)(Parent.Transform.Position.Y - CamY);
192
193             float RasteredHeight = this.Height * Parent.Transform.Scale.Y;
194             float RasteredWidth = this.Width * Parent.Transform.Scale.X;
195
196             float NonRasteredWidthRatio = 1 / Parent.Transform.Scale.X;
197             float NonRasteredHeightRatio = 1 / Parent.Transform.Scale.Y;
198
199             float NonRasteredHeight = 0;
200             float NonRasteredWidth = 0;
201             if (ReColor == null)
202             {
203                 for (int row = 0; row < RasteredHeight; row++)
204                 {
205                     NonRasteredWidth = 0;
206
207                     if (y + row >= Engine.Render.RenderHeight)
208                         break;
209
210                     for (int column = 0; column < RasteredWidth; column++)
211                     {
212                         if (x + column >= Engine.Render.RenderWidth)
```

```
213                         break;
214
215                     if (IsOnScreen(x + column, y + row))
216                     {
217                         int offset = (int)(3 * ((y + row) * Engine.Render.RenderWidth + (x + column)));
218                         int keyOffset = (int)((y + row) * Engine.Render.RenderWidth + (x + column));
219
220                         int tempColumn = (int)NonRasteredWidth;
221                         int tempRow = (int)NonRasteredHeight;
222
223                         int index = _BytesPerPixel * (tempRow * Width + tempColumn);
224
225                         if (Engine.Render.imageBufferKey[keyOffset] != 255 && _BytesPerPixel == 4 ? _Data[index + 3] != 0 :
    true)
226                         {
227                             Color temp = MixPixel(Engine.Render.imageBufferKey[keyOffset], Engine.Render.imageBuffer[offset
    + 2], Engine.Render.imageBuffer[offset + 1], Engine.Render.imageBuffer[offset],
228                                                   _BytesPerPixel == 4 ? _Data[index + 3] : (byte)255, _Data[index + 2], _Data[index +
    1], _Data[index]);
229
230                             Engine.Render.imageBufferKey[keyOffset] = temp.A;
231
232                             Engine.Render.imageBuffer[offset] = temp.B;
233                             Engine.Render.imageBuffer[offset + 1] = temp.G;
234                             Engine.Render.imageBuffer[offset + 2] = temp.R;
235                         }
236                     }
237
238                     NonRasteredWidth += NonRasteredWidthRatio;
239                 }
240
241                 NonRasteredHeight += NonRasteredHeightRatio;
242             }
243         }
244         else
245         {
246             Color tempColor = (Color)ReColor;
247
248             for (int row = 0; row < RasteredHeight; row++)
249             {
250                 NonRasteredWidth = 0;
251
252                 if (y + row >= Engine.Render.RenderHeight)
253                     break;
254
255                 for (int column = 0; column < RasteredWidth; column++)
256                 {
257                     if (x + column >= Engine.Render.RenderWidth)
258                         break;
259
260                     if (IsOnScreen(x + column, y + row))
261                     {
262                         int offset = (int)(3 * ((y + row) * Engine.Render.RenderWidth + (x + column)));
263                         int keyOffset = (int)((y + row) * Engine.Render.RenderWidth + (x + column));
264
265                         int tempColumn = (int)NonRasteredWidth;
266                         int tempRow = (int)NonRasteredHeight;
267
268                         int index = _BytesPerPixel * (tempRow * Width + tempColumn);
269
270                         if (Engine.Render.imageBufferKey[keyOffset] != 255 && _BytesPerPixel == 4 ? _Data[index + 3] != 0 :
    true)
271                         {
272                             Color temp = MixPixel(Color.FromArgb(Engine.Render.imageBufferKey[keyOffset], Engine.Ren-
    der.imageBuffer[offset + 2], Engine.Render.imageBuffer[offset + 1], Engine.Render.imageBuffer[offset]),
273                                                   tempColor);
274
275                             Engine.Render.imageBufferKey[keyOffset] = temp.A;
276
277                             Engine.Render.imageBuffer[offset] = temp.B;
278                             Engine.Render.imageBuffer[offset + 1] = temp.G;
279                             Engine.Render.imageBuffer[offset + 2] = temp.R;
280                         }
281                     }
282
283                     NonRasteredWidth += NonRasteredWidthRatio;
284                 }
```

```
285
286                 NonRasteredHeight += NonRasteredHeightRatio;
287             }
288         }
289
290         if (HasShadow)
291         {
292             NonRasteredHeight = 0;
293             NonRasteredWidth = 0;
294
295             x++;
296             y++;
297
298             for (int row = 0; row < RasteredHeight; row++)
299             {
300                 NonRasteredWidth = 0;
301
302                 if (y + row >= Engine.Render.RenderHeight)
303                     break;
304
305                 for (int column = 0; column < RasteredWidth; column++)
306                 {
307                     if (x + column >= Engine.Render.RenderWidth)
308                         break;
309
310                     if (IsOnScreen(x + column, y + row))
311                     {
312                         int offset = (int)(3 * ((y + row) * Engine.Render.RenderWidth + (x + column)));
313                         int keyOffset = (int)((y + row) * Engine.Render.RenderWidth + (x + column));
314
315                         int tempColumn = (int)NonRasteredWidth;
316                         int tempRow = (int)NonRasteredHeight;
317
318                         int index = _BytesPerPixel * (tempRow * Width + tempColumn);
319
320                         if (Engine.Render.imageBufferKey[keyOffset] != 255 && _BytesPerPixel == 4 ? _Data[index + 3] != 0 :
    true)
321                         {
322                             Color temp = MixPixel(Engine.Render.imageBufferKey[keyOffset], Engine.Render.imageBuffer[offset
    + 2], Engine.Render.imageBuffer[offset + 1], Engine.Render.imageBuffer[offset],
323                                                   (byte)192, (byte)0, (byte)0, (byte)0);
324
325                             Engine.Render.imageBufferKey[keyOffset] = temp.A;
326
327                             Engine.Render.imageBuffer[offset] = temp.B;
328                             Engine.Render.imageBuffer[offset + 1] = temp.G;
329                             Engine.Render.imageBuffer[offset + 2] = temp.R;
330                         }
331                     }
332
333                     NonRasteredWidth += NonRasteredWidthRatio;
334                 }
335
336                 NonRasteredHeight += NonRasteredHeightRatio;
337             }
338         }
339     }
340
341     public Color MixPixel(byte topA, byte topR, byte topG, byte topB, byte bottomA, byte bottomR, byte bottomG, byte
    bottomB)
342     {
343         if (topA == 0)
344             return Color.FromArgb(bottomA, bottomR, bottomG, bottomB);
345
346         if (bottomA == 0)
347             return Color.FromArgb(topA, topR, topG, topB);
348
349         float opacityTop = (float)topA / 255;
350
351         byte newA = (byte)(topA + bottomA >= 255 ? 255 : topA + bottomA);
352         byte A = (byte)(newA - topA);
353
354         float opacityBottom = (float)A / 255;
355
356         byte R = (byte)(topR * opacityTop + bottomR * opacityBottom);
357         byte G = (byte)(topG * opacityTop + bottomG * opacityBottom);
358         byte B = (byte)(topB * opacityTop + bottomB * opacityBottom);
```

```
359
360             return Color.FromArgb(newA, R, G, B);
361         }
362
363         public Color MixPixel(Color top, Color bottom)
364         {
365             if (top.A == 0)
366                 return bottom;
367
368             if (bottom.A == 0)
369                 return top;
370
371             float opacityTop = (float)top.A / 255;
372
373             byte newA = (byte)(top.A + bottom.A >= 255 ? 255 : top.A + bottom.A);
374             byte A = (byte)(newA - top.A);
375
376             float opacityBottom = (float)A / 255;
377
378             byte R = (byte)(top.R * opacityTop + bottom.R * opacityBottom);
379             byte G = (byte)(top.G * opacityTop + bottom.G * opacityBottom);
380             byte B = (byte)(top.B * opacityTop + bottom.B * opacityBottom);
381
382             return Color.FromArgb(newA, R, G, B);
383         }
384
385         private bool IsOnScreen(float x, float y)
386         {
387             return x >= 0 && x < Engine.Render.RenderWidth && y >= 0 && y < Engine.Render.RenderHeight;
388         }
389     }
390 }
```

2.2.1.9 Core/Components/Parabola.cs

```
1   using System;
2
3   namespace DKEngine.Core.Components
4   {
5       /*
6        * ------------------
7        * DOES NOT WORK YET
8        * ------------------
9        */
10
11      [Obsolete]
12      public class Parabola : Behavior
13      {
14          public TimeSpan Time;
15          public float Y;
16
17          private float _accumulated = 0f;
18          public float Accumulated { get { return _accumulated; } }
19
20          public bool Enabled = false;
21
22          private float Elapsed = 0f;
23
24          private float[] ValuesInTime;
25          private int NumberOfSamples;
26          private const float SamplesInSecodnd = 1000;
27          private const float X1 = 0f;
28          public float X2 { get; private set; }
29
30          public Parabola(GameObject Parent)
31              : base(Parent)
32          {
33              Name = string.Format("{0}_{1}", Parent.Name, nameof(Parabola));
34          }
35
36          public override void Destroy()
37          { }
38
39          protected internal override void Start()
40          {
41              NumberOfSamples = (int)Time.TotalMilliseconds;
42              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);
```

```
109                 }
110                 catch
111                 {
112                     IsAvailable = false;
113                 }
114             }
115         }
116     }
117
118     public void StopSound(Sound sound)
119     {
120         if (Engine.Sound.IsSoundEnabled)
121         {
122             if (IsAvailable)
123             {
124                 try
125                 {
126                     Engine.Sound.Instance.StopSound(sound);
127                 }
128                 catch
129                 { }
130             }
131         }
132     }
133
134     public override void Destroy()
135     {
136         try
137         {
138             Engine.LoadingScene.AllComponents.Remove(this.Name);
139         }
140         catch
141         { }
142
143         this.Parent = null;
144     }
145 }
146
147 internal class CachedSoundSampleProvider : ISampleProvider
148 {
149     public Sound cachedSound;
150     private long position;
151
152     public CachedSoundSampleProvider(Sound cachedSound)
153     {
154         this.cachedSound = cachedSound;
155     }
156
157     public int Read(float[] buffer, int offset, int count)
158     {
159         var availableSamples = cachedSound.AudioData.Length - position;
160         var samplesToCopy = Math.Min(availableSamples, count);
161         Array.Copy(cachedSound.AudioData, position, buffer, offset, samplesToCopy);
162         position += samplesToCopy;
163
164         return (int)samplesToCopy;
165     }
166
167     public WaveFormat WaveFormat { get { return cachedSound.WaveFormat; } }
168 }
169
170 /// <summary>
171 /// Class holding specified audio file
172 /// </summary>
173 public class Sound
174 {
175     public float[] AudioData { get; private set; }
176     public WaveFormat WaveFormat { get; private set; }
177     public AudioFileReader FileReader { get; private set; }
178     internal ISampleProvider _SampleProvider { get; set; }
179
180     public Sound(string audioFileName)
181     {
182         using (FileReader = new AudioFileReader(audioFileName))
183         {
184             FileReader.Volume = Engine.Sound.SoundVolume;
185             // TODO: could add resampling in here if required
```

```
186                 WaveFormat = FileReader.WaveFormat;
187                 var wholeFile = new List<float>((int)(FileReader.Length / 4));
188                 var readBuffer = new float[FileReader.WaveFormat.SampleRate * FileReader.WaveFormat.Channels];
189                 int samplesRead;
190                 while ((samplesRead = FileReader.Read(readBuffer, 0, readBuffer.Length)) > 0)
191                 {
192                     wholeFile.AddRange(readBuffer.Take(samplesRead));
193                 }
194                 AudioData = wholeFile.ToArray();
195             }
196         }
197     }
198 }
```

2.2.1.11 Core/Components/Transform.cs

```
1   /*
2   * (C) 2017 David Knieradl
3   */
4
5   namespace DKEngine.Core.Components
6   {
7       /// <summary>
8       /// Transformation class holding information about position, scale and sizes of GameObject
9       /// </summary>
10      /// <seealso cref="DKEngine.Core.Components.Component" />
11      public sealed class Transform : Component
12      {
13          private Vector3 _Dimensions;
14          private Vector3 _Position;
15          private Vector3 _Scale;
16
17          public Vector3 Dimensions
18          {
19              get { return _Dimensions; }
20              set
21              {
22                  Vector3 tmp = value - _Dimensions;
23                  _Dimensions = value;
24                  _ScaledDimensions = _Dimensions * _Scale;
25
26                  int childCount = Parent.Child.Count;
27                  for (int i = 0; i < childCount; i++)
28                      Parent.Child[i].Transform.Dimensions += tmp;
29              }
30          }
31
32          public Vector3 Position
33          {
34              get { return _Position; }
35              set
36              {
37                  Vector3 tmp = value - _Position;
38                  _Position = value;
39
40                  int childCount = Parent.Child.Count;
41                  for (int i = 0; i < childCount; i++)
42                      Parent.Child[i].Transform.Position += tmp;
43              }
44          }
45
46          public Vector3 Scale
47          {
48              get { return _Scale; }
49              set
50              {
51                  Vector3 tmp = value / _Scale;
52                  _Scale = value;
53                  _ScaledDimensions = _Dimensions * _Scale;
54
55                  int childCount = Parent.Child.Count;
56                  for (int i = 0; i < childCount; i++)
57                      Parent.Child[i].Transform.Scale *= tmp;
58              }
59          }
60
61          internal Vector3 _ScaledDimensions;
```

```
62
63          public Transform(GameObject Parent)
64              : base(Parent)
65          {
66              _Position = new Vector3();
67              _Dimensions = new Vector3();
68              _Scale = new Vector3();
69              _ScaledDimensions = new Vector3();
70          }
71
72          public override void Destroy()
73          {
74              try
75              {
76                  Engine.LoadingScene.AllComponents.Remove(this.Name);
77              }
78              catch { }
79          }
80
81          /// <summary>
82          /// Possible directions
83          /// </summary>
84          public enum Direction
85          {
86              Up,
87              Left,
88              Down,
89              Right
90          }
91      }
92  }
```

2.2.1.12 Core/Components/Vector3.cs

```
1   namespace DKEngine.Core.Components
2   {
3   #pragma warning disable CS0660 // Type defines operator == or operator != but does not override Object.Equals(object
    o)
4   #pragma warning disable CS0661 // Type defines operator == or operator != but does not override Object.GetHa-
    shCode()
5
6       /// <summary>
7       /// Three-dimensional vector
8       /// </summary>
9       public struct Vector3
10  #pragma warning restore CS0661 // Type defines operator == or operator != but does not override Object.GetHa-
    shCode()
11  #pragma warning restore CS0660 // Type defines operator == or operator != but does not override Object.Equals(object
    o)
12      {
13          /// <summary>
14          /// The X vector
15          /// </summary>
16          public float X;
17
18          /// <summary>
19          /// The Y vector
20          /// </summary>
21          public float Y;
22
23          /// <summary>
24          /// The Z vector
25          /// </summary>
26          public float Z;
27
28          public Vector3(float X, float Y, float Z)
29          {
30              this.X = X;
31              this.Y = Y;
32              this.Z = Z;
33          }
34
35          public static Vector3 operator -(Vector3 left, Vector3 right)
36          {
37              return new Vector3(left.X - right.X, left.Y - right.Y, left.Z - right.Z);
38          }
39
```

```
40          public static Vector3 operator -(Vector3 left, float right)
41          {
42              return new Vector3(left.X - right, left.Y - right, left.Z - right);
43          }
44
45          public static Vector3 operator +(Vector3 left, Vector3 right)
46          {
47              return new Vector3(left.X + right.X, left.Y + right.Y, left.Z + right.Z);
48          }
49
50          public static Vector3 operator +(Vector3 left, float right)
51          {
52              return new Vector3(left.X + right, left.Y + right, left.Z + right);
53          }
54
55          public static Vector3 operator *(Vector3 left, Vector3 right)
56          {
57              return new Vector3(left.X * right.X, left.Y * right.Y, left.Z * right.Z);
58          }
59
60          public static Vector3 operator *(Vector3 left, float right)
61          {
62              return new Vector3(left.X * right, left.Y * right, left.Z * right);
63          }
64
65          public static Vector3 operator /(Vector3 left, Vector3 right)
66          {
67              return new Vector3(left.X / right.X, left.Y / right.Y, left.Z / right.Z);
68          }
69
70          public static Vector3 operator /(Vector3 left, float right)
71          {
72              return new Vector3(left.X / right, left.Y / right, left.Z / right);
73          }
74
75          public static bool operator ==(Vector3 left, Vector3 right)
76          {
77              return left.X == right.X && left.Y == right.Y && left.Z == right.Z;
78          }
79
80          public static bool operator !=(Vector3 left, Vector3 right)
81          {
82              return left.X != right.X || left.Y != right.Y || left.Z != right.Z;
83          }
84
85          public Vector3 Add(Vector3 Value)
86          {
87              return this + Value;
88          }
89
90          public Vector3 Add(float X, float Y, float Z)
91          {
92              return this + new Vector3(X, Y, Z);
93          }
94
95          public Vector3 Add(float Value)
96          {
97              return this + Value;
98          }
99
100         public Vector3 Decrease(Vector3 Value)
101         {
102             return this - Value;
103         }
104
105         public Vector3 Decrease(float X, float Y, float Z)
106         {
107             return this - new Vector3(X, Y, Z);
108         }
109
110         public Vector3 Decrease(float Value)
111         {
112             return this - Value;
113         }
114
115         public Vector3 Multiply(Vector3 Value)
116         {
```

```
            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;
```

```
115             }
116
117             for (int i = 0; i < textAlignedCount; i++)
118             {
119                 float maxWidth = 0;
120                 int textAlignedRowCount = textAligned[i].Count;
121
122                 if (textAlignedRowCount > 0)
123                     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;
124
125                 if (maxWidth != 0)
126                 {
127                     float startX = 0;
128
129                     switch (_Parent._THA)
130                     {
131                         case HorizontalAlignment.Left:
132                             startX = 0;
133                             break;
134
135                         case HorizontalAlignment.Center:
136                             startX = (_Parent.Transform.Dimensions.X * _Parent.Transform.Scale.X - maxWidth) / 2;
137                             break;
138
139                         case HorizontalAlignment.Right:
140                             startX = _Parent.Transform.Dimensions.X * _Parent.Transform.Scale.X - maxWidth;
141                             break;
142                     }
143
144                     for (int j = 0; j < textAlignedRowCount; j++)//foreach (Letter letter in row)
145                     {
146                         if (startX != 0 || startY != 0)
147                             textAligned[i][j].Transform.Position += new Vector3(startX, startY, 0);
148
149                         retValue.Add(textAligned[i][j]);
150                     }
151                 }
152             }
153
154             _Parent._text = retValue;
155
156             _Parent._changed = false;
157         }
158
159         protected internal override void OnColliderEnter(Collider e)
160         { }
161     }
162 }
```

2.2.1.14 Core/SystemExt/Extensions.cs

```
1   /*
2   * (C) 2017 David Knieradl
3   */
4
5   using DKEngine.Core.Components;
6   using System.Collections.Generic;
7   using System.Linq;
8
9   namespace DKEngine.Core.Ext
10  {
11      public static class Extensions
12      {
13          public static void AddSafe<DataValue>(this Dictionary<string, DataValue> Destination, Component Target)
14              where DataValue : Component
15          {
16              string Key = Target.Name;
17
18              if (Engine.LoadingScene.ComponentCount.ContainsKey(Target.Name))
19              {
20                  Target.Name = string.Format("{0}_(Copy {1})", Key, Engine.LoadingScene.ComponentCount[Target.Name]++);
21                  Key = Target.Name;
22              }
23              else
24              {
25                  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);

```

```
50              Console.BufferHeight = Console.LargestWindowHeight;
51              Console.BufferWidth = Console.LargestWindowWidth;
52
53              Console.ForegroundColor = ConsoleColor.White;
54              Console.BackgroundColor = ConsoleColor.Black;
55
56              Console.Clear();
57
58              WindowSizeChecker(null, null);
59
60              Timer windowChecker = new Timer()
61              {
62                  AutoReset = true,
63                  Enabled = true,
64                  Interval = 1000f
65              };
66              windowChecker.Elapsed += WindowSizeChecker;
67
68              windowChecker.Start();
69          }
70
71          private static void WindowSizeChecker(object sender, ElapsedEventArgs e)
72          {
73              if (Console.WindowHeight != Console.LargestWindowHeight || Console.WindowWidth != Console.LargestWin-
    dowWidth)
74              {
75                  if (ConsoleStateChangeAvailable)
76                  {
77                      if (!SetConsoleDisplayMode(hConsole, 1, out xy))
78                      {
79                          ConsoleStateChangeAvailable = false;
80                      }
81                  }
82
83                  Console.CursorVisible = false;
84              }
85          }
86      }
87  }
```

2.2.1.16 Core/UI/Letter.cs

```
1   /*
2   * (C) 2017 David Knieradl
3   */
4
5   namespace DKEngine.Core.UI
6   {
7       internal sealed class Letter : GameObject
8       {
9           private Letter()
10          { }
11
12          public Letter(TextBlock Parent)
13              : base(Parent)
14          { }
15
16          public override void Destroy()
17          {
18              try
19              {
20                  if (Engine.LoadingScene.NewlyGeneratedComponents.Contains(this))
21                  {
22                      Engine.LoadingScene.NewlyGeneratedComponents.Pop();
23                  }
24              }
25              catch
26              { }
27
28              try
29              {
30                  Engine.RenderObjects.Remove(this);
31              }
32              catch { }
33
34              Parent?.Child.Remove(this);
35
```

```
            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;
```

```
37
38                  if (value != null)
39                      Model = new Material((Color)value, this);
40              }
41          }
42
43          public float FontSize
44          {
45              get { return _FontSize; }
46              set
47              {
48                  if (value <= 0)
49                  {
50                      _FontSize = 0.01f;
51                      _changed = true;
52                  }
53                  else
54                  {
55                      _FontSize = value;
56                      _changed = true;
57                  }
58              }
59          }
60
61          public HorizontalAlignment HAlignment
62          {
63              set
64              {
65                  _HA = value;
66
67                  //if (_IsGUI)
68                  //{
69                  this.Transform.Position -= new Vector3(horiOffset, 0, 0);
70
71                  switch (value)
72                  {
73                      case HorizontalAlignment.Left:
74                          horiOffset = 0;
75                          break;
76
77                      case HorizontalAlignment.Center:
78                          horiOffset = (Engine.Render.RenderWidth - this.Transform._ScaledDimensions.X) / 2;
79                          break;
80
81                      case HorizontalAlignment.Right:
82                          horiOffset = Engine.Render.RenderWidth - this.Transform._ScaledDimensions.X;
83                          break;
84                  }
85
86                  this.Transform.Position += new Vector3(horiOffset, 0, 0);
87                  //}
88
89                  //_changed = true;
90              }
91          }
92
93          public VerticalAlignment VAlignment
94          {
95              set
96              {
97                  _VA = value;
98
99                  //if (_IsGUI)
100                 //{
101                 this.Transform.Position -= new Vector3(0, vertOffset, 0);
102
103                 switch (value)
104                 {
105                     case VerticalAlignment.Top:
106                         vertOffset = 0;
107                         break;
108
109                     case VerticalAlignment.Center:
110                         vertOffset = (Engine.Render.RenderHeight - this.Transform._ScaledDimensions.Y) / 2;
111                         break;
112
113                     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;
```

```
6   using System.Diagnostics;
7   using System.Drawing;
8   using System.IO;
9   using System.Linq;
10  using System.Resources;
11
12  namespace DKEngine.Core
13  {
14      /// <summary>
15      /// DKEngine library database holding all loaded materials, scenes, etc.
16      /// </summary>
17      public static class Database
18      {
19          private enum Font
20          {
21              Num0,
22              Num1,
23              Num2,
24              Num3,
25              Num4,
26              Num5,
27              Num6,
28              Num7,
29              Num8,
30              Num9,
31              A,
32              AngleBracketLeft,
33              AngleBracketRight,
34              ArrowToLeft,
35              ArrowToRight,
36              ArrowToTop,
37              B,
38              Backslash,
39              BraceLeft,
40              BraceRight,
41              BracketLeft,
42              BracketRight,
43              C,
44              Colon,
45              Comma,
46              D,
47              Dot,
48              E,
49              Equals,
50              ExclamationMark,
51              F,
52              G,
53              H,
54              Hashtag,
55              I,
56              J,
57              K,
58              L,
59              M,
60              Minus,
61              N,
62              O,
63              P,
64              Percent,
65              Q,
66              QuestionMark,
67              QuotationMarks,
68              R,
69              S,
70              Semicolon,
71              Slash,
72              StarLarge,
73              StarSmall,
74              T,
75              U,
76              Underscore,
77              V,
78              W,
79              X,
80              Y,
81              Z,
82              NumberOfTypes
```

```
83          };
84
85          private static Dictionary<char, Font> font = new Dictionary<char, Font>()
86          {
87              { '0' , Font.Num0 },
88              { '1' , Font.Num1 },
89              { '2' , Font.Num2 },
90              { '3' , Font.Num3 },
91              { '4' , Font.Num4 },
92              { '5' , Font.Num5 },
93              { '6' , Font.Num6 },
94              { '7' , Font.Num7 },
95              { '8' , Font.Num8 },
96              { '9' , Font.Num9 },
97              { 'A' , Font.A },
98              { 'B' , Font.B },
99              { 'C' , Font.C },
100             { 'D' , Font.D },
101             { 'E' , Font.E },
102             { 'F' , Font.F },
103             { 'G' , Font.G },
104             { 'H' , Font.H },
105             { 'I' , Font.I },
106             { 'J' , Font.J },
107             { 'K' , Font.K },
108             { 'L' , Font.L },
109             { 'M' , Font.M },
110             { 'N' , Font.N },
111             { 'O' , Font.O },
112             { 'P' , Font.P },
113             { 'Q' , Font.Q },
114             { 'R' , Font.R },
115             { 'S' , Font.S },
116             { 'T' , Font.T },
117             { 'U' , Font.U },
118             { 'V' , Font.V },
119             { 'W' , Font.W },
120             { 'X' , Font.X },
121             { 'Y' , Font.Y },
122             { 'Z' , Font.Z },
123             { '-' , Font.Minus },
124             { '?' , Font.QuestionMark },
125             { '!' , Font.ExclamationMark },
126             { '.' , Font.Dot },
127             { ':' , Font.Colon },
128             { ',' , Font.Comma },
129             { '[' , Font.AngleBracketLeft },
130             { ']' , Font.AngleBracketRight },
131             { '>' , Font.ArrowToRight },
132             { '<' , Font.ArrowToLeft },
133             { '^' , Font.ArrowToTop },
134             { '{' , Font.BraceLeft },
135             { '}' , Font.BraceRight },
136             { '(' , Font.BracketLeft },
137             { ')' , Font.BracketRight },
138             { '=' , Font.Equals },
139             { '#' , Font.Hashtag },
140             { '%' , Font.Percent },
141             { '"' , Font.QuotationMarks },
142             { ';' , Font.Semicolon },
143             { '☆' , Font.StarLarge },
144             { '*' , Font.StarSmall },
145             { '_' , Font.Underscore },
146             { '/' , Font.Slash },
147             { '\\' , Font.Backslash }
148         };
149
150         private static List<Material> letterMaterial = new List<Material>();
151
152         private static void CreateLetterReferences()
153         {
154             using (BinaryReader br = new BinaryReader(new MemoryStream(Resources.FontFile)))
155             {
156                 int lenght = br.ReadInt32();
157
158                 for (int index = 0; index < lenght; index++)
159                 {
```

```
160                 byte[] byteArray = br.ReadBytes(br.ReadInt32());
161
162                 using (MemoryStream ms = new MemoryStream(byteArray))
163                 {
164                     letterMaterial.Add(new Material((Bitmap)Image.FromStream(ms)));
165                 }
166             }
167         }
168     }
169
170     private static Dictionary<string, Material> CachedMaterials = new Dictionary<string, Material>();
171     private static Dictionary<string, Scene> CachedScenes = new Dictionary<string, Scene>();
172
173     internal static void InitDatabase()
174     {
175         AddNewGameObjectMaterial("border", new Material(Resources.border));
176         AddNewGameObjectMaterial("splashScreen", new Material(Resources.DKEngine_splash2));
177
178         CreateLetterReferences();
179     }
180
181     internal static Scene GetScene(string Key)
182     {
183         Scene retValue = null;
184
185         try
186         {
187             retValue = CachedScenes[Key];
188         }
189         catch
190         { }
191
192         return retValue;
193     }
194
195     public static Material GetLetter(this char ch)
196     {
197         Material retValue = null;
198
199         try
200         {
201             retValue = letterMaterial[(int)font[Char.ToUpper(ch)]];
202         }
203         catch
204         {
205             retValue = letterMaterial[(int)font['?']];
206         }
207
208         return retValue;
209     }
210
211     public static void AddNewGameObjectMaterial(string ObjectName, Material Object)
212     {
213         try
214         {
215             if (Object != null)
216             {
217                 CachedMaterials.Add(ObjectName, Object);
218             }
219             else
220                 throw new Exception("Material is null\n" + Object.ToString());
221         }
222         catch (Exception e)
223         {
224             Debug.WriteLine("Object not found\n" + e);
225         }
226     }
227
228     public static Material GetGameObjectMaterial(string Key)
229     {
230         Material retValue = null;
231
232         try
233         {
234             retValue = CachedMaterials[Key];
235         }
236         catch (Exception e)
```

```
237             {
238                 Debug.WriteLine("Object not found\n" + e);
239             }
240 
241             return retValue;
242         }
243 
244         public static Material GetGameObjectMaterial(int Position)
245         {
246             Material retValue = null;
247 
248             try
249             {
250                 retValue = CachedMaterials.ElementAtOrDefault(Position).Value;
251             }
252             catch (Exception e)
253             {
254                 Debug.WriteLine("Object not found\n" + e);
255             }
256 
257             return retValue;
258         }
259 
260         public static string GetMaterialDatabaseKey(int Position)
261         {
262             return CachedMaterials.ElementAtOrDefault(Position).Key; //.FirstOrDefault(x => x.Value == Position).Key;
263         }
264 
265         public static void LoadResources(ResourceSet source)
266         {
267             foreach (DictionaryEntry entry in source)
268             {
269                 if (entry.Value is Image)
270                 {
271                     AddNewGameObjectMaterial((string)entry.Key, new Material((Bitmap)entry.Value));
272                 }
273             }
274         }
275 
276         internal static void RewriteWorld(string Name, object[] argsPreLoad = null)
277         {
278             try
279             {
280                 Engine.LoadingScene = CachedScenes[Name];
281 
282                 object[] preArgs = argsPreLoad ?? Engine.LoadingScene.argsPreLoad;
283                 object[] postArgs = Engine.LoadingScene.argsPostLoad;
284 
285                 var list = Engine.LoadingScene.AllComponents.ToList();
286                 for (int i = 0; i < list.Count; i++)
287                 {
288                     list[0].Value.Destroy();
289                     list.RemoveAt(0);
290                     list = Engine.LoadingScene.AllComponents.ToList();
291                 }
292 
293                 for (int i = 0; i < Engine.LoadingScene.AllBehaviors.Count; i++)
294                 {
295                     Engine.LoadingScene.AllBehaviors[0].Destroy();
296                 }
297 
298                 Engine.LoadingScene = (Scene)Activator.CreateInstance(Engine.LoadingScene.GetType());
299 
300                 Engine.LoadingScene.argsPreLoad = preArgs;
301                 Engine.LoadingScene.argsPostLoad = postArgs;
302 
303                 Engine.LoadingScene.Set(Engine.LoadingScene.argsPreLoad);
304                 Engine.LoadingScene.Init();
305                 CachedScenes[Name] = Engine.LoadingScene;
306             }
307             catch
308             { }
309         }
310 
311         internal static void AddScene(Scene Source)
312         {
313             try
```

```
314             {
315                 CachedScenes.Add(Source.Name, Source);
316             }
317             catch { }
318         }
319     }
320 }
```

2.2.1.20 Core/GameObject.cs

```
1   using DKEngine.Core.Components;
2   using System;
3   using System.Collections.Generic;
4   using System.Diagnostics;
5   using System.Drawing;
6   using System.Reflection;
7
8   namespace DKEngine.Core
9   {
10      /// <summary>
11      /// Primitive type for all renderable objects
12      /// </summary>
13      /// <seealso cref="DKEngine.Core.Components.Component" />
14      public class GameObject : Component
15      {
16          /// <summary>
17          /// The GameObject has shadow
18          /// </summary>
19          public bool HasShadow = false;
20
21          /// <summary>
22          /// Gets a value indicating whether this instance is in view.
23          /// </summary>
24          /// <value>
25          ///   <c>true</c> if this instance is in view; otherwise, <c>false</c>.
26          /// </value>
27          public bool IsInView
28          {
29              get
30              {
31                  float X = this.IsGUI ? 0 : Engine.BaseCam != null ? Engine.BaseCam.X : 0;
32                  float Y = this.IsGUI ? 0 : Engine.BaseCam != null ? Engine.BaseCam.Y : 0;
33
34                  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);
35              }
36          }
37
38          /// <summary>
39          /// Gets or sets a value indicating whether this instance is GUI.
40          /// </summary>
41          /// <value>
42          ///   <c>true</c> if this instance is GUI; otherwise, <c>false</c>.
43          /// </value>
44          public bool IsGUI
45          {
46              get { return Parent != null ? Parent.IsGUI : _IsGUI; }
47              set { _IsGUI = value; }
48          }
49
50          /// <summary>
51          /// Gets or sets the name of the type.
52          /// </summary>
53          /// <value>
54          /// The name of the type.
55          /// </value>
56          public string TypeName
57          {
58              get { return _typeName; }
59              set
60              {
61                  _typeName = value;
62                  this.Model = Database.GetGameObjectMaterial(value);
63              }
64          }
65
```

```
66          /// <summary>
67          /// Gets or sets the model.
68          /// </summary>
69          /// <value>
70          /// The model.
71          /// </value>
72          public Material Model
73          {
74              get { return _Model; }
75              set
76              {
77                  if (value != _Model && value != null)
78                  {
79                      _Model = value;
80                      this.Transform.Dimensions = new Vector3(value.Width, value.Height, 1);
81
82                      if (Animator?.Animations.Count == 0)
83                      {
84                          Animator.AddAnimation("default", _Model);
85                          Animator.Play("default");
86                      }
87                  }
88              }
89          }
90
91          /// <summary>
92          /// Gets or sets the collider.
93          /// </summary>
94          /// <value>
95          /// The collider.
96          /// </value>
97          public Collider Collider
98          {
99              get { return _collider; }
100             set
101             {
102                 if (_collider != value)
103                 {
104                     if (_collider != null)
105                     {
106                         foreach (Script scr in this.Scripts)
107                         {
108                             _collider.CollisionEvent -= scr.CollisionHandler;
109                             scr.CollisionHandler = null;
110                         }
111                     }
112
113                     if (value != null)
114                     {
115                         foreach (Script scr in this.Scripts)
116                         {
117                             scr.CollisionHandler = new Collider.CollisionEnterHandler(scr.OnColliderEnter);
118                             value.CollisionEvent += scr.CollisionHandler;
119                         }
120                     }
121
122                     _collider = value;
123                 }
124             }
125         }
126
127         /// <summary>
128         /// Gets or sets the animator.
129         /// </summary>
130         /// <value>
131         /// The animator.
132         /// </value>
133         public Animator Animator { get; set; }
134
135         /// <summary>
136         /// Gets or sets the sound source.
137         /// </summary>
138         /// <value>
139         /// The sound source.
140         /// </value>
141         public SoundSource SoundSource { get; set; }
142
```

```
143         /// <summary>
144         /// Gets or sets the foreground.
145         /// </summary>
146         /// <value>
147         /// The foreground.
148         /// </value>
149         public Color? Foreground { get; set; }
150
151         /// <summary>
152         /// Gets the transform.
153         /// </summary>
154         /// <value>
155         /// The transform.
156         /// </value>
157         public Transform Transform { get; }
158
159         /// <summary>
160         /// Gets the list of childs.
161         /// </summary>
162         /// <value>
163         /// The child.
164         /// </value>
165         public List<GameObject> Child { get; }
166
167         internal List<Script> Scripts { get; }
168         internal bool _IsGUI = false;
169         internal string _typeName = "";
170         internal Material _Model = null;
171         internal Collider _collider = null;
172
173         public GameObject()
174             : base(null)
175         {
176             this.Child = new List<GameObject>();
177             this.Scripts = new List<Script>();
178             this.Transform = new Transform(this)
179             {
180                 Dimensions = new Vector3(1, 1, 1),
181                 Scale = new Vector3(1, 1, 1),
182                 Position = new Vector3(0, 0, 0)
183             };
184         }
185
186         public GameObject(GameObject Parent)
187             : base(Parent)
188         {
189             this.Child = new List<GameObject>();
190             this.Scripts = new List<Script>();
191             this.Transform = new Transform(this)
192             {
193                 Dimensions = new Vector3(1, 1, 1),
194                 Scale = new Vector3(1, 1, 1),
195                 Position = new Vector3(0, 0, 0)
196             };
197
198             if (Parent != null)
199             {
200                 this.Parent = Parent;
201
202                 Parent.Child.Add(this);
203                 this.Transform.Position = Parent.Transform.Position;
204                 this.Transform.Scale = Parent.Transform.Scale;
205             }
206         }
207
208         internal override void Init()
209         {
210             Initialize();
211
212             try
213             {
214                 if (Parent == null)
215                     Engine.LoadingScene.Model.Add(this);
216
217                 Engine.LoadingScene.GameObjectsToAddToRender.Push(this);
218             }
219             catch (Exception e)
```

```
220             {
221                 Debug.WriteLine("Loading scene is NULL\n\n{0}", e);
222             }
223         }
224 
225         protected virtual void Initialize()
226         { }
227 
228         /// <summary>
229         /// Initializes the new script.
230         /// </summary>
231         /// <typeparam name="T">Scirpt</typeparam>
232         public void InitNewScript<T>() where T : Script
233         {
234             this.Scripts.Add((T)Activator.CreateInstance(typeof(T), this));
235         }
236 
237         /// <summary>
238         /// Initializes the new component.
239         /// </summary>
240         /// <typeparam name="T">Component</typeparam>
241         public void InitNewComponent<T>() where T : Component
242         {
243             if (typeof(T) == typeof(Animator) || typeof(T).IsSubclassOf(typeof(Animator)))
244             {
245                 if (this.Animator == null)
246                 {
247                     this.Animator = new Animator(this);
248                 }
249 
250                 return;
251             }
252 
253             if (typeof(T) == typeof(Collider) || typeof(T).IsSubclassOf(typeof(Collider)))
254             {
255                 if (this.Collider == null)
256                 {
257                     Type t = typeof(T);
258                     this.Collider = (Collider)t.Assembly.CreateInstance(t.FullName, false, BindingFlags.Instance | Bin-
dingFlags.NonPublic | BindingFlags.Public, null, new object[] { this }, null, null);
259                 }
260 
261                 return;
262             }
263 
264             if (typeof(T) == typeof(SoundSource) || typeof(T).IsSubclassOf(typeof(SoundSource)))
265             {
266                 if (this.SoundSource == null)
267                 {
268                     Type t = typeof(T);
269                     this.SoundSource = (SoundSource)t.Assembly.CreateInstance(t.FullName, false, BindingFlags.Instance |
BindingFlags.NonPublic | BindingFlags.Public, null, new object[] { this }, null, null);
270                 }
271 
272                 return;
273             }
274         }
275 
276         public override void Destroy()
277         {
278             try
279             {
280                 if (Engine.LoadingScene.NewlyGeneratedComponents.Contains(this))
281                 {
282                     Engine.LoadingScene.DestroyObjectAwaitList.Add(this);
283                     return;
284                 }
285             }
286             catch { }
287 
288             try
289             {
290                 Engine.LoadingScene.AllComponents.Remove(this.Name);
291             }
292             catch { }
293 
294             try
```

```
295             {
296                 Engine.RenderObjects.Remove(this);
297             }
298             catch { }
299 
300             try
301             {
302                 Engine.LoadingScene.Model.Remove(this);
303             }
304             catch { }
305 
306             int ScriptCount = this.Scripts.Count;
307             for (int i = 0; i < ScriptCount; i++)
308                 Scripts[0].Destroy();
309 
310             int ChildCount = this.Child.Count;
311             for (int i = 0; i < ChildCount; i++)
312                 Child[0].Destroy();
313 
314             this.Animator?.Destroy();
315             this.Animator = null;
316 
317             this.Collider?.Destroy();
318             this.Collider = null;
319 
320             this.Parent = null;
321         }
322 
323         internal virtual void Render()
324         { Model?.Render(this, Foreground); }
325 
326         /// <summary>
327         /// Finds the specified GameObject of desired name.
328         /// </summary>
329         /// <typeparam name="T">Type</typeparam>
330         /// <param name="Name">Desired name</param>
331         /// <returns></returns>
332         public static new T Find<T>(string Name) where T : GameObject
333         {
334             T retValue = null;
335 
336             try
337             {
338                 retValue = (T)Engine.LoadingScene.AllComponents[Name];
339             }
340             catch (Exception ex)
341             {
342                 Debug.WriteLine("Object not found\n" + ex);
343             }
344 
345             return retValue;
346         }
347 
348         /// <summary>
349         /// Finds the specified GameObject of desired name.
350         /// </summary>
351         /// <param name="Name">Desired name</param>
352         /// <returns></returns>
353         public static GameObject Find(string Name)
354         {
355             GameObject retValue = null;
356 
357             try
358             {
359                 retValue = (GameObject)Engine.LoadingScene.AllComponents[Name];
360             }
361             catch (Exception ex)
362             {
363                 Debug.WriteLine("Object not found\n" + ex);
364             }
365 
366             return retValue;
367         }
368 
369         /// <summary>
370         /// Instantiates GameObject.
371         /// </summary>
```

```
372         /// <typeparam name="T">Type</typeparam>
373         /// <param name="Position">The position</param>
374         /// <param name="Dimensions">The dimensions</param>
375         /// <param name="Scale">The scale</param>
376         /// <returns></returns>
377         public static T Instantiate<T>(Vector3 Position, Vector3 Dimensions, Vector3 Scale)
378             where T : GameObject, new()
379         {
380             T retValue = new T();
381
382             retValue.Transform.Position = Position;
383             retValue.Transform.Dimensions = Dimensions;
384             retValue.Transform.Scale = Scale;
385
386             return retValue;
387         }
388
389         /// <summary>
390         /// Instantiates GameObject.
391         /// </summary>
392         /// <typeparam name="T">Type</typeparam>
393         /// <param name="Transform">The transform</param>
394         /// <returns></returns>
395         public static T Instantiate<T>(Transform @Transform)
396             where T : GameObject, new()
397         {
398             return Instantiate<T>(@Transform.Position, @Transform.Dimensions, @Transform.Scale);
399         }
400     }
401 }
```

2.2.1.21 Core/Scene.cs

```
1   using DKEngine.Core.Components;
2   using System.Collections.Generic;
3
4   namespace DKEngine.Core
5   {
6       /// <summary>
7       /// DKEngine library scene
8       /// </summary>
9       /// <seealso cref="DKEngine.IPage" />
10      public abstract class Scene : IPage
11      {
12          public string Name = "";
13
14          internal Camera BaseCamera;
15
16          internal readonly Dictionary<string, Component> AllComponents;
17          internal readonly Dictionary<string, int> ComponentCount;
18          //internal readonly Dictionary<string, GameObject> AllGameObjects;
19
20          internal readonly List<GameObject> Model;
21          internal readonly List<Behavior> AllBehaviors;
22          internal readonly List<Collider> AllGameObjectsColliders;
23
24          internal readonly Stack<Component> NewlyGeneratedComponents;
25          internal readonly Stack<Behavior> NewlyGeneratedBehaviors;
26
27          internal readonly Stack<GameObject> GameObjectsToAddToRender;
28          internal readonly Stack<GameObject> GameObjectsAddedToRender;
29
30          internal readonly List<GameObject> DestroyObjectAwaitList;
31
32          internal object[] argsPreLoad;
33          internal object[] argsPostLoad;
34
35          public Scene()
36          {
37              AllComponents = new Dictionary<string, Component>(0xFFFF);
38              ComponentCount = new Dictionary<string, int>(0xFFFF);
39
40              AllBehaviors = new List<Behavior>(0xFFFF);
41              Model = new List<GameObject>(0xFFFF);
42              AllGameObjectsColliders = new List<Collider>(0xFFFF);
43
44              NewlyGeneratedComponents = new Stack<Component>(0xFFFF);
```

```
45              NewlyGeneratedBehaviors = new Stack<Behavior>(0xFFFF);
46
47              GameObjectsToAddToRender = new Stack<GameObject>(0xFFFF);
48              GameObjectsAddedToRender = new Stack<GameObject>(0xFFFF);
49
50              DestroyObjectAwaitList = new List<GameObject>(0xFFFF);
51          }
52
53          /// <summary>
54          /// Initializes model of Scene.
55          /// </summary>
56          public abstract void Init();
57
58          /// <summary>
59          /// Sets the specified arguments.
60          /// </summary>
61          /// <param name="args">The arguments</param>
62          public virtual void Set(params object[] args)
63          { }
64
65          /// <summary>
66          /// Unloads this instance.
67          /// </summary>
68          public abstract void Unload();
69
70          public static T Find<T>(string name)
71              where T : Scene
72          {
73              return (T)Database.GetScene(name);
74          }
75      }
76  }
```

2.2.1.22 Core/Script.cs

```
1   using DKEngine.Core.Components;
2
3   namespace DKEngine.Core
4   {
5       /// <summary>
6       /// Script base class
7       /// </summary>
8       /// <seealso cref="DKEngine.Core.Components.Behavior" />
9       public abstract class Script : Behavior
10      {
11          internal Collider.CollisionEnterHandler CollisionHandler;
12
13          public Script(GameObject Parent)
14              : base(Parent)
15          {
16              if (Parent.Collider != null)
17              {
18                  CollisionHandler = new Collider.CollisionEnterHandler(OnColliderEnter);
19                  Parent.Collider.CollisionEvent += CollisionHandler;
20              }
21          }
22
23          protected internal abstract void OnColliderEnter(Collider e);
24
25          public override void Destroy()
26          {
27              try
28              {
29                  Engine.LoadingScene.AllComponents.Remove(this.Name);
30              }
31              catch { }
32
33              try
34              {
35                  Engine.LoadingScene.AllBehaviors.Remove(this);
36              }
37              catch { }
38
39              if (UpdateHandle != null)
40                  Engine.UpdateEvent -= UpdateHandle;
41
42              if (CollisionHandler != null)
```

```
43                  Parent.Collider.CollisionEvent -= CollisionHandler;
44
45              Parent.Scripts.Remove(this);
46              Parent = null;
47              UpdateHandle = null;
48          }
49      }
50  }
```

2.2.1.23 Data/SplashScreen.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3
4   namespace DKEngine
5   {
6       internal sealed class SplashScreen : GameObject
7       {
8           public SplashScreen()
9           {
10              this.TypeName = "splashScreen";
11              this.InitNewComponent<Animator>();
12          }
13
14          public SplashScreen(GameObject Parent)
15              : base(Parent)
16          {
17              this.TypeName = "splashScreen";
18              this.InitNewComponent<Animator>();
19          }
20
21          protected override void Initialize()
22          { }
23      }
24  }
```

2.2.1.24 Data/SplashScreenScene.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3
4   namespace DKEngine.Data
5   {
6       internal class SplashScreenScene : Scene
7       {
8           internal SplashScreen Splash;
9
10          public override void Init()
11          {
12              Splash = new SplashScreen();
13              Splash.Transform.Position = new Vector3(-32, 0, 0);
14              Splash.Transform.Scale = new Vector3(0.5f, 0.5f, 0);
15              Camera splashScreenCam = new Camera();
16          }
17
18          public override void Unload()
19          { }
20      }
21  }
```

2.2.2 MarIO

2.2.2.1 Program.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using MarIO.Assets.Scenes;
4   using MarIO.Assets.Sprites;
5   using System.Globalization;
6
7   namespace MarIO
8   {
9       public class Program
10      {
11          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;
```

```
58
59              public const int GOOMBA_POINTS = 100;
60              public const int COIN_SCORE = 100;
61              public const int MUSHROOM_SCORE = 200;
62              public const int FLOWER_SCORE = 300;
63              public const int STAR_SCORE = 500;
64          }
65
66          public static class AnimatedWorldReferences
67          {
68              public static List<Block> BlocksToUpdate = new List<Block>();
69              public static List<float> BlocksStartPositions = new List<float>();
70
71              public static List<TextBlock> FloatingTexts = new List<TextBlock>();
72              public static List<float> FloatingTextStartPosition = new List<float>();
73
74              public static Stack<Block> SpecialActions = new Stack<Block>();
75
76              public static List<Coin> FloatingCoins = new List<Coin>();
77              public static List<float> FloatingCoinsStartPosition = new List<float>();
78          }
79
80          public static class Assets
81          {
82              public static class Sounds
83              {
84                  public const string OVERWORLD_THEME = @".\Assets\Sounds\Overworld_Theme.mp3";
85                  public const string MARIO_JUMP_FX = @".\Assets\Sounds\smb_jump-small.mp3";
86                  public const string PIPE_ENTER_FX = @".\Assets\Sounds\smb_pipe.mp3";
87                  public const string COIN_GET_FX = @".\Assets\Sounds\smb_coin.mp3";
88                  public const string UP_1_FX = @".\Assets\Sounds\smb_1-up.mp3";
89                  public const string BREAK_BLOCK_FX = @".\Assets\Sounds\smb_breakblock.mp3";
90                  public const string MARIO_DIE_FX = @".\Assets\Sounds\smb_mariodie.mp3";
91                  public const string POWER_UP_FX = @".\Assets\Sounds\smb_powerup.mp3";
92                  public const string STOMP_FX = @".\Assets\Sounds\smb_stomp.mp3";
93
94                  public static readonly Sound OVERWORLD_THEME_SOUND = new Sound(OVERWORLD_THEME);
95                  public static readonly Sound FX_MARIO_JUMP_SOUND = new Sound(MARIO_JUMP_FX);
96                  public static readonly Sound FX_PIPE_ENTER_SOUND = new Sound(PIPE_ENTER_FX);
97                  public static readonly Sound FX_1_UP_SOUND = new Sound(UP_1_FX);
98                  public static readonly Sound FX_BREAK_BLOCK_SOUND = new Sound(BREAK_BLOCK_FX);
99                  public static readonly Sound FX_MARIO_DIE_SOUND = new Sound(MARIO_DIE_FX);
100                 public static readonly Sound FX_POWER_UP_SOUND = new Sound(POWER_UP_FX);
101                 public static readonly Sound FX_STOMP_SOUND = new Sound(STOMP_FX);
102             }
103
104             public static class Animations
105             {
106                 #region Mario
107
108                 private const string POWERUP_LEFT = "powerup_left";
109                 private const string POWERUP_LEFT_MAT = "mario_powerup_left";
110
111                 private const string POWERUP_RIGHT = "powerup_right";
112                 private const string POWERUP_RIGHT_MAT = "mario_powerup_right";
113
114                 private const string CROUCHING_LEFT = "crouch_left";
115                 private const string CROUCHING_LEFT_MAT = "mario_crouch_left";
116
117                 private const string CROUCHING_RIGHT = "crouch_right";
118                 private const string CROUCHING_RIGHT_MAT = "mario_crouch_right";
119
120                 /*---------- SMALL ----------*/
121
122                 public const string MARIO_IDLE_LEFT = "idle_left";
123                 public const string MARIO_IDLE_LEFT_MAT = "mario_left";
124
125                 public const string MARIO_IDLE_RIGHT = "idle_right";
126                 public const string MARIO_IDLE_RIGHT_MAT = "mario_right";
127
128                 public const string MARIO_MOVE_LEFT = "move_left";
129                 public const string MARIO_MOVE_LEFT_MAT = "mario_move_left";
130
131                 public const string MARIO_MOVE_RIGHT = "move_right";
132                 public const string MARIO_MOVE_RIGHT_MAT = "mario_move_right";
133
134                 public const string MARIO_JUMP_LEFT = "jump_left";
```

```
135         public const string MARIO_JUMP_LEFT_MAT = "mario_jump_left";
136
137         public const string MARIO_JUMP_RIGHT = "jump_right";
138         public const string MARIO_JUMP_RIGHT_MAT = "mario_jump_right";
139
140         public const string MARIO_DEAD = "dead";
141         public const string MARIO_DEAD_MAT = "mario_dead";
142
143         public const string MARIO_CROUCHING_LEFT = CROUCHING_LEFT;
144         public const string MARIO_CROUCHING_LEFT_MAT = MARIO_IDLE_LEFT_MAT;
145
146         public const string MARIO_CROUCHING_RIGHT = CROUCHING_RIGHT;
147         public const string MARIO_CROUCHING_RIGHT_MAT = MARIO_IDLE_RIGHT_MAT;
148
149         /*---------- SUPER ----------*/
150
151         public const string MARIO_SUPER_IDLE_LEFT = "super_" + MARIO_IDLE_LEFT;
152         public const string MARIO_SUPER_IDLE_LEFT_MAT = "super_" + MARIO_IDLE_LEFT_MAT;
153
154         public const string MARIO_SUPER_IDLE_RIGHT = "super_" + MARIO_IDLE_RIGHT;
155         public const string MARIO_SUPER_IDLE_RIGHT_MAT = "super_" + MARIO_IDLE_RIGHT_MAT;
156
157         public const string MARIO_SUPER_MOVE_LEFT = "super_" + MARIO_MOVE_LEFT;
158         public const string MARIO_SUPER_MOVE_LEFT_MAT = "super_" + MARIO_MOVE_LEFT_MAT;
159
160         public const string MARIO_SUPER_MOVE_RIGHT = "super_" + MARIO_MOVE_RIGHT;
161         public const string MARIO_SUPER_MOVE_RIGHT_MAT = "super_" + MARIO_MOVE_RIGHT_MAT;
162
163         public const string MARIO_SUPER_JUMP_LEFT = "super_" + MARIO_JUMP_LEFT;
164         public const string MARIO_SUPER_JUMP_LEFT_MAT = "super_" + MARIO_JUMP_LEFT_MAT;
165
166         public const string MARIO_SUPER_JUMP_RIGHT = "super_" + MARIO_JUMP_RIGHT;
167         public const string MARIO_SUPER_JUMP_RIGHT_MAT = "super_" + MARIO_JUMP_RIGHT_MAT;
168
169         public const string MARIO_SUPER_POWERUP_LEFT = "super_" + POWERUP_LEFT;
170         public const string MARIO_SUPER_POWERUP_LEFT_MAT = "super_" + POWERUP_LEFT_MAT;
171
172         public const string MARIO_SUPER_POWERUP_RIGHT = "super_" + POWERUP_RIGHT;
173         public const string MARIO_SUPER_POWERUP_RIGHT_MAT = "super_" + POWERUP_RIGHT_MAT;
174
175         public const string MARIO_SUPER_CROUCHING_LEFT = "super_" + CROUCHING_LEFT;
176         public const string MARIO_SUPER_CROUCHING_LEFT_MAT = "super_" + CROUCHING_LEFT_MAT;
177
178         public const string MARIO_SUPER_CROUCHING_RIGHT = "super_" + CROUCHING_RIGHT;
179         public const string MARIO_SUPER_CROUCHING_RIGHT_MAT = "super_" + CROUCHING_RIGHT_MAT;
180
181         /*---------- FIRE ----------*/
182
183         public const string MARIO_FIRE_IDLE_LEFT = "fire_" + MARIO_IDLE_LEFT;
184         public const string MARIO_FIRE_IDLE_LEFT_MAT = "fire_" + MARIO_IDLE_LEFT_MAT;
185
186         public const string MARIO_FIRE_IDLE_RIGHT = "fire_" + MARIO_IDLE_RIGHT;
187         public const string MARIO_FIRE_IDLE_RIGHT_MAT = "fire_" + MARIO_IDLE_RIGHT_MAT;
188
189         public const string MARIO_FIRE_MOVE_LEFT = "fire_" + MARIO_MOVE_LEFT;
190         public const string MARIO_FIRE_MOVE_LEFT_MAT = "fire_" + MARIO_MOVE_LEFT_MAT;
191
192         public const string MARIO_FIRE_MOVE_RIGHT = "fire_" + MARIO_MOVE_RIGHT;
193         public const string MARIO_FIRE_MOVE_RIGHT_MAT = "fire_" + MARIO_MOVE_RIGHT_MAT;
194
195         public const string MARIO_FIRE_JUMP_LEFT = "fire_" + MARIO_JUMP_LEFT;
196         public const string MARIO_FIRE_JUMP_LEFT_MAT = "fire_" + MARIO_JUMP_LEFT_MAT;
197
198         public const string MARIO_FIRE_JUMP_RIGHT = "fire_" + MARIO_JUMP_RIGHT;
199         public const string MARIO_FIRE_JUMP_RIGHT_MAT = "fire_" + MARIO_JUMP_RIGHT_MAT;
200
201         public const string MARIO_FIRE_POWERUP_LEFT = "fire_" + POWERUP_LEFT;
202         public const string MARIO_FIRE_POWERUP_LEFT_MAT = "fire_" + POWERUP_LEFT_MAT;
203
204         public const string MARIO_FIRE_POWERUP_RIGHT = "fire_" + POWERUP_RIGHT;
205         public const string MARIO_FIRE_POWERUP_RIGHT_MAT = "fire_" + POWERUP_RIGHT_MAT;
206
207         public const string MARIO_FIRE_CROUCHING_LEFT = "fire_" + CROUCHING_LEFT;
208         public const string MARIO_FIRE_CROUCHING_LEFT_MAT = "fire_" + CROUCHING_LEFT_MAT;
209
210         public const string MARIO_FIRE_CROUCHING_RIGHT = "fire_" + CROUCHING_RIGHT;
211         public const string MARIO_FIRE_CROUCHING_RIGHT_MAT = "fire_" + CROUCHING_RIGHT_MAT;
```

```
212
213             /*---------- INVINCIBLE ----------*/
214
215             /*public const string MARIO_INVINCIBLE_IDLE_LEFT;
216             public const string MARIO_INVINCIBLE_IDLE_LEFT_MAT;
217
218             public const string MARIO_INVINCIBLE_IDLE_RIGHT;
219             public const string MARIO_INVINCIBLE_IDLE_RIGHT_MAT;
220
221             public const string MARIO_INVINCIBLE_MOVE_LEFT;
222             public const string MARIO_INVINCIBLE_MOVE_LEFT_MAT;
223
224             public const string MARIO_INVINCIBLE_MOVE_RIGHT;
225             public const string MARIO_INVINCIBLE_MOVE_RIGHT_MAT;
226
227             public const string MARIO_INVINCIBLE_JUMP_LEFT;
228             public const string MARIO_INVINCIBLE_JUMP_LEFT_MAT;
229
230             public const string MARIO_INVINCIBLE_JUMP_RIGHT;
231             public const string MARIO_INVINCIBLE_JUMP_RIGHT_MAT;
232
233             public const string MARIO_INVINCIBLE_DEAD;
234             public const string MARIO_INVINCIBLE_DEAD_MAT;*/
235
236             #endregion Mario
237         }
238     }
239 }
240 }
```

2.2.2.3 SystemExt.cs

```
1   using DKEngine.Core.UI;
2   using MarIO.Assets.Models;
3   using MarIO.Assets.Models.Miscellaneous;
4   using System.Drawing;
5
6   namespace MarIO
7   {
8       public static class SystemExt
9       {
10          public static void AddAsFloatingText(this TextBlock txBlock)
11          {
12              Shared.AnimatedWorldReferences.FloatingTexts.Add(txBlock);
13              Shared.AnimatedWorldReferences.FloatingTextStartPosition.Add(txBlock.Transform.Position.Y);
14          }
15
16          public static void AnimateBlockCollision(this Block block)
17          {
18              if (Shared.Mechanics.MarioCurrentState == Mario.State.Small || block.HadBonus)
19              {
20                  block.State = Block.CollisionState.Up;
21
22                  Shared.AnimatedWorldReferences.BlocksToUpdate.Add(block);
23                  Shared.AnimatedWorldReferences.BlocksStartPositions.Add(block.Transform.Position.Y);
24              }
25              else
26              {
27              }
28          }
29
30          public static void AddAsFloatingCoin(this Coin coin)
31          {
32              Shared.AnimatedWorldReferences.FloatingCoins.Add(coin);
33              Shared.AnimatedWorldReferences.FloatingCoinsStartPosition.Add(coin.Transform.Position.Y);
34          }
35
36          public static Color ToColor(this uint color)
37          {
38              byte a = (byte)(color >> 24);
39              byte r = (byte)(color >> 16);
40              byte g = (byte)(color >> 8);
41              byte b = (byte)(color >> 0);
42              return Color.FromArgb(a, r, g, b);
43          }
44      }
45  }
```

2.2.2.4 Assets/Models/Miscellaneous/Coin.cs

```
1  using DKEngine.Core;
2  using DKEngine.Core.Components;

3  namespace MarIO.Assets.Models.Miscellaneous
4  {
5  public class Coin : GameObject
6  {
7  public static Sound COIN_FX = new Sound(Shared.Assets.Sounds.COIN_GET_FX);

8  public Coin()
9  { }

10 public Coin(GameObject Parent)
11 : base(Parent)
12 { }

13 protected override void Initialize()
14 {
15 this.Name = "coin";
16 //this.TypeName = "coin";
17 this.InitNewComponent<Animator>();
18 this.Animator.AddAnimation("default", Database.GetGameObjectMaterial("coin"));
19 }
20 }
21 }
```

2.2.2.5 Assets/Models/Miscellaneous/Heart.cs

```
1  using DKEngine.Core;
2  using DKEngine.Core.Components;
3
4  namespace MarIO.Assets.Models.Miscellaneous
5  {
6      public class Heart : GameObject
7      {
8          public Heart()
9          { }
10
11         public Heart(GameObject Parent)
12             : base(Parent)
13         { }
14
15         protected override void Initialize()
16         {
17             this.Name = "heart";
18             //this.TypeName = "coin";
19             this.InitNewComponent<Animator>();
20             this.Animator.AddAnimation("default", Database.GetGameObjectMaterial("heart"));
21         }
22     }
23 }
```

2.2.2.6 Assets/Models/Miscellaneous/PowerUp.cs

```
1  using DKEngine.Core;
2  using DKEngine.Core.Components;
3  using DKEngine.Core.UI;
4  using MarIO.Assets.Scripts;
5  using System;
6  using System.Drawing;
7
8  namespace MarIO.Assets.Models.Miscellaneous
9  {
10     public class PowerUp : GameObject
11     {
12         public Mario PlayerReference;
13
14         public enum PowerUpType
15         {
16             Mushroom,
17             Flower,
18             Star
19         }
20
```

```
21          public Action OnPickedUp { get; private set; }
22          public PowerUpType Type { get; private set; }
23
24          protected override void Initialize()
25          {
26              this.Name = nameof(PowerUp);
27
28              this.InitNewComponent<Collider>();
29              this.Collider.Area = new RectangleF(0, 0, 16, 16);
30              this.Collider.Enabled = false;
31
32              this.InitNewScript<PowerUpScript>();
33
34              switch (Shared.Mechanics.MarioCurrentState)
35              {
36                  case Mario.State.Small:
37                      this.TypeName = "mushroom";
38                      Type = PowerUpType.Mushroom;
39                      OnPickedUp = () =>
40                      {
41                          Shared.Mechanics.GameScore += Shared.Mechanics.MUSHROOM_SCORE;
42                          TextBlock FloatingText = new TextBlock()
43                          {
44                              Text = string.Format("{0}", Shared.Mechanics.MUSHROOM_SCORE),
45                              TextShadow = true
46                          };
47                          FloatingText.Transform.Position = this.Transform.Position;
48                          FloatingText.Transform.Dimensions = new Vector3(20, 6, 0);
49                          FloatingText.AddAsFloatingText();
50                          PlayerReference.CurrentState = Mario.State.Super;
51
52                          OnPickedUp = null;
53
54                          this.Destroy();
55                      };
56                      break;
57
58                  case Mario.State.Super:
59                      this.TypeName = "flower";
60                      Type = PowerUpType.Flower;
61                      this.InitNewComponent<Animator>();
62                      this.Animator.AddAnimation("default", "flower");
63                      this.Animator.Play("default");
64                      OnPickedUp = () =>
65                      {
66                          Shared.Mechanics.GameScore += Shared.Mechanics.FLOWER_SCORE;
67                          TextBlock FloatingText = new TextBlock()
68                          {
69                              Text = string.Format("{0}", Shared.Mechanics.FLOWER_SCORE),
70                              TextShadow = true
71                          };
72                          FloatingText.Transform.Position = this.Transform.Position;
73                          FloatingText.Transform.Dimensions = new Vector3(20, 6, 0);
74                          FloatingText.AddAsFloatingText();
75
76                          PlayerReference.CurrentState = Mario.State.Fire;
77
78                          OnPickedUp = null;
79
80                          this.Destroy();
81                      };
82                      this.Collider.IsTrigger = true;
83                      break;
84
85                  case Mario.State.Fire:
86                  case Mario.State.Invincible:
87                      this.TypeName = "1-UP";
88                      Type = PowerUpType.Star;
89                      this.InitNewComponent<Animator>();
90                      this.Animator.AddAnimation("default", "star");
91                      this.Animator.Play("default");
92                      OnPickedUp = () =>
93                      {
94                          Shared.Mechanics.GameScore += Shared.Mechanics.STAR_SCORE;
95                          TextBlock FloatingText = new TextBlock()
96                          {
97                              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
```

```
12      {
13          public enum BlockType
14          {
15              Ground1,
16              Ground2,
17              Ground3,
18              Ground4,
19              Bridge,
20              Bush1,
21              Bush2,
22              Bush3,
23              BushSmall,
24              CastleBig,
25              CastleSmall,
26              Cloud1,
27              Cloud2,
28              Cloud3,
29              Fence,
30              Finish,
31              Flag,
32              FlagPole,
33              Mountain,
34              NoCoin,
35              Sky,
36              Water1,
37              Water2,
38              Pipe1,
39              Pipe2,
40              Pipe3,
41              Pipe4,
42              Pipe5,
43              UnderGround1,
44              UnderGround2,
45              UnderGround3,
46              UnderGround4,
47              UnderGroundBackground1,
48              UnderGroundBackground2,
49              NumberOfObjects
50          }
51
52          public static Dictionary<BlockType, string> BlockTypeNames = new Dictionary<BlockType, string>()
53          {
54              { BlockType.Bridge, "bridge" },
55              { BlockType.Bush1, "bush_01" },
56              { BlockType.Bush2, "bush_02" },
57              { BlockType.Bush3, "bush_03" },
58              { BlockType.BushSmall, "bush_small" },
59              { BlockType.CastleBig, "castle_big" },
60              { BlockType.CastleSmall, "castle_small" },
61              { BlockType.Cloud1, "cloud_01" },
62              { BlockType.Cloud2, "cloud_02" },
63              { BlockType.Cloud3, "cloud_03" },
64              { BlockType.Fence, "fence" },
65              { BlockType.Flag, "finish_flag" },
66              { BlockType.FlagPole, "flag_pole" },
67              { BlockType.Finish, "" },
68              { BlockType.Ground1, "block_1_with_coin" },
69              { BlockType.Ground2, "block_02" },
70              { BlockType.Ground3, "block_03" },
71              { BlockType.Ground4, "block_04" },
72              { BlockType.Mountain, "mountain" },
73              { BlockType.NoCoin, "block_nocoins" },
74              { BlockType.Pipe1, "pipe_01" },
75              { BlockType.Pipe2, "pipe_02" },
76              { BlockType.Pipe3, "pipe_03" },
77              { BlockType.Pipe4, "pipe_04" },
78              { BlockType.Pipe5, "pipe_05" },
79              { BlockType.Sky, "sky" },
80              { BlockType.UnderGround1, "underground_block_01" },
81              { BlockType.UnderGround2, "underground_block_02" },
82              { BlockType.UnderGround3, "underground_block_03" },
83              { BlockType.UnderGround4, "underground_block_04" },
84              { BlockType.UnderGroundBackground1, "background_01" },
85              { BlockType.UnderGroundBackground2, "background_02" },
86              { BlockType.Water1, "water_01" },
87              { BlockType.Water2, "water_02" },
88          };
```

```
89
90          public enum CollisionState
91          {
92              Stay,
93              Up,
94              Down
95          }
96
97          public BlockType Type { get; set; }
98          public bool InitCollider { get; set; }
99          public CollisionState State { get; set; }
100
101         public bool SpecialActionActivate
102         {
103             get { return _specialAction; }
104             set
105             {
106                 if (value)
107                 {
108                     Shared.AnimatedWorldReferences.SpecialActions.Push(this);
109                 }
110
111                 _specialAction = value;
112             }
113         }
114
115         public Action SpecialAction { get; set; }
116         public Direction PipeEnterDirection { get; set; }
117         public bool CoinGot { get; set; }
118
119         public bool PowerUp
120         {
121             get { return _powerUp; }
122             set
123             {
124                 _powerUp = value;
125                 if (value)
126                     _hadBonus = true;
127             }
128         }
129
130         public byte CoinCount
131         {
132             get { return _coinCount; }
133             set
134             {
135                 _coinCount = value;
136                 if (value > 0)
137                     _hadBonus = true;
138             }
139         }
140
141         public bool HadBonus
142         {
143             get { return _hadBonus; }
144         }
145
146         private bool _powerUp = false;
147         private byte _coinCount = 0;
148         private bool _hadBonus = false;
149         private bool _specialAction = false;
150         private SoundOutput FX_Player;
151
152         public Block()
153             : base()
154         { }
155
156         public Block(GameObject Parent)
157             : base(Parent)
158         { }
159
160         protected override void Initialize()
161         {
162             this.TypeName = BlockTypeNames[Type];
163             if (InitCollider)
164                 this.InitNewComponent<Collider>();
165
```

```
166             switch (Type)
167             {
168                 case BlockType.Finish:
169                     {
170                         this.Transform.Dimensions = new Vector3(32, 200, 0);
171 
172                         Block part1 = new Block(this)
173                         {
174                             Name = string.Format("{0}_Flag", this.Name),
175                             Type = BlockType.Flag
176                         };
177                         Block part2 = new Block(this)
178                         {
179                             Name = string.Format("{0}_Pole", this.Name),
180                             Type = BlockType.FlagPole
181                         };
182                         part2.Transform.Position -= new Vector3(16, 0, 0);
183                     }
184                     break;
185 
186                 case BlockType.Pipe1:
187                     {
188                         PipeEnterDirection = Direction.Right;
189                         this.InitNewComponent<Collider>();
190                         this.Collider.IsTrigger = true;
191                         this.Collider.Area = new System.Drawing.RectangleF(-1, 15, 1, 1);
192 
193                         this.InitNewScript<PipePort>();
194 
195                         Blocker block = new Blocker(this)
196                         {
197                             Name = string.Format("{0}_Blocker", this.Name)
198                         };
199                         block.InitNewComponent<Collider>();
200                         block.Collider.Area = new System.Drawing.RectangleF(0, 0, this.Transform.Dimensions.X, this.Trans-
    form.Dimensions.Y);
201                     }
202                     break;
203 
204                 case BlockType.Pipe3:
205                     {
206                         PipeEnterDirection = Direction.Down;
207                         this.InitNewComponent<Collider>();
208                         this.Collider.IsTrigger = true;
209                         this.Collider.Area = new System.Drawing.RectangleF(15, -1, 1, 1);
210 
211                         this.InitNewScript<PipePort>();
212 
213                         Blocker block = new Blocker(this)
214                         {
215                             Name = string.Format("{0}_Blocker", this.Name)
216                         };
217                         block.InitNewComponent<Collider>();
218                         block.Collider.Area = new System.Drawing.RectangleF(0, 0, this.Transform.Dimensions.X, this.Trans-
    form.Dimensions.Y);
219                     }
220                     break;
221             }
222 
223             if (CoinCount > 0 || PowerUp)
224             {
225                 this.InitNewComponent<Animator>();
226                 this.Animator.AddAnimation("default", this.TypeName);
227                 this.Animator.AddAnimation("nocoin", BlockTypeNames[BlockType.NoCoin]);
228             }
229 
230             FX_Player = GameObject.Find<SoundOutput>(nameof(SoundOutput));
231         }
232 
233         public void GetContent()
234         {
235             if (PowerUp)
236             {
237                 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));
238                 PowerUp = false;
239                 this.Animator.Play("nocoin");
```

```
240             }
241             else if (CoinCount > 0 && !CoinGot)
242             {
243                 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();
244                 CoinCount--;
245                 Shared.Mechanics.GameScore += Shared.Mechanics.COIN_SCORE;
246                 Shared.Mechanics.FXSoundSource.PlaySound(Coin.COIN_FX);
247                 CoinGot = true;
248
249                 if (CoinCount == 0)
250                 {
251                     this.Animator.Play("nocoin");
252                 }
253             }
254         }
255
256         public void DestroyAnim()
257         { }
258     }
259 }
```

2.2.2.10 Assets/Models/Blocker.cs

```
1  using DKEngine.Core;
2  using DKEngine.Core.Components;
3
4  namespace MarIO.Assets.Models
5  {
6      public class Blocker : GameObject
7      {
8          public Blocker()
9              : base()
10         { }
11
12         public Blocker(GameObject Parent)
13             : base(Parent)
14         { }
15
16         protected override void Initialize()
17         {
18             this.InitNewComponent<Collider>();
19         }
20     }
21 }
```

2.2.2.11 Assets/Models/Delayer.cs

```
1  using DKEngine.Core;
2  using MarIO.Assets.Scripts;
3  using System;
4
5  namespace MarIO.Assets.Models
6  {
7      public class Delayer : GameObject
8      {
9          public TimeSpan TimeToWait;
10         public Action CalledAction;
11
12         public Delayer()
13         {
14             Name = nameof(Delayer);
15         }
16
17         protected override void Initialize()
18         {
19             this.InitNewScript<DelayScript>();
20         }
21     }
22 }
```

2.2.2.12 Assets/Models/Enemy.cs

```
1  using DKEngine.Core;
2  using DKEngine.Core.Components;
3  using MarIO.Assets.Scripts;
```

```
4   using System.Collections.Generic;
5
6   namespace MarIO.Assets.Models
7   {
8       public abstract class Enemy : AnimatedObject
9       {
10          public enum EnemyType
11          {
12              Goomba,
13              GoombaBlue,
14              GoombaSilver,
15              KoopaTroopa,
16              KoopaParatroopa,
17              PiranhaPlant,
18              Spiny,
19              BuzzyBeatle,
20              BuzzyBeatleBlue,
21              BuzzyBeatleSilver,
22              FireBar,
23              BulletBill,
24              BillBlasterLarge,
25              BillBlasterSmall
26          }
27
28          protected static Dictionary<EnemyType, string> EnemyTypeNames = new Dictionary<EnemyType, string>()
29          {
30              { EnemyType.Goomba, "goomba" },
31              { EnemyType.GoombaBlue, "" },
32              { EnemyType.GoombaSilver, "" },
33              { EnemyType.KoopaTroopa, "" },
34              { EnemyType.KoopaParatroopa, "" },
35              { EnemyType.PiranhaPlant, "" },
36              { EnemyType.Spiny, "" },
37              { EnemyType.BuzzyBeatle, "" },
38              { EnemyType.BuzzyBeatleBlue, "" },
39              { EnemyType.BuzzyBeatleSilver, "" },
40              { EnemyType.FireBar, "" },
41              { EnemyType.BulletBill, "" },
42              { EnemyType.BillBlasterLarge, "" },
43              { EnemyType.BillBlasterSmall, "" }
44          };
45
46          public EnemyType Type { get; set; }
47
48          public Enemy()
49              : base()
50          { }
51
52          public Enemy(GameObject Parent)
53              : base(Parent)
54          { }
55      }
56
57      internal class Goomba : Enemy
58      {
59          protected override void Initialize()
60          {
61              this.Name = "Goomba";
62              this.Type = EnemyType.Goomba;
63
64              this.InitNewComponent<Collider>();
65              this.Collider.Area = new System.Drawing.RectangleF(0, 0, 16, 16);
66
67              this.InitNewScript<GoombaController>();
68              this.InitNewComponent<Animator>();
69              this.Animator.AddAnimation("default", Database.GetGameObjectMaterial(EnemyTypeNames[Type]));
70              this.Animator.AddAnimation("dead", Database.GetGameObjectMaterial(EnemyTypeNames[Type] + "_dead"));
71          }
72      }
73  }
```

2.2.2.13 Assets/Models/Group.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3
4   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;

```

```
44              Vector3 tmp = this.Transform.Position;
45
46              switch (value)
47              {
48                  case State.Dead:
49                  case State.Small:
50                      this.Collider.Area = new RectangleF(2, 0, 12, 16);
51
52                      TopTrigger.Transform.Position = tmp.Add(2.5f, -1, 0);//new Vector3(tmp.X + 2.5f, tmp.Y - 1, tmp.Z);
53                      TopTrigger.Transform.Dimensions = new Vector3(11, 1, 0);
54
55                      RightTrigger.Transform.Position = tmp.Add(14, 0, 0); //new Vector3(tmp.X + 14, tmp.Y, tmp.Z);
56                      RightTrigger.Transform.Dimensions = new Vector3(1, 14, 0);
57
58                      LeftTrigger.Transform.Position = tmp.Add(1, 0, 0); //new Vector3(tmp.X + 1, tmp.Y, tmp.Z);
59                      LeftTrigger.Transform.Dimensions = new Vector3(1, 14, 0);
60
61                      BottomTrigger.Transform.Position = tmp.Add(1, 16, 0); //new Vector3(tmp.X + 1, tmp.Y + 16, tmp.Z);
62                      BottomTrigger.Transform.Dimensions = new Vector3(14, 2, 0);
63
64                      TopTrigger.Collider.Area = new RectangleF(0, 0, 11, 1);
65                      RightTrigger.Collider.Area = new RectangleF(0, 0, 1, 14);
66                      LeftTrigger.Collider.Area = new RectangleF(0, 0, 1, 14);
67                      BottomTrigger.Collider.Area = new RectangleF(0, 0, 14, 2);
68
69                      break;
70
71                  case State.Super:
72                  case State.Fire:
73                  case State.Invincible:
74                      this.Collider.Area = new RectangleF(0, 0, 16, 32);
75
76                      TopTrigger.Transform.Position = tmp.Add(0.5f, -1, 0); //new Vector3(tmp.X + 0.5f, tmp.Y - 1, tmp.Z + 0);
77                      TopTrigger.Transform.Dimensions = new Vector3(15, 1, 0);
78
79                      RightTrigger.Transform.Position = tmp.Add(16, 0, 0); //new Vector3(tmp.X + 16, tmp.Y + 0, tmp.Z + 0);
80                      RightTrigger.Transform.Dimensions = new Vector3(1, 30, 0);
81
82                      LeftTrigger.Transform.Position = tmp.Add(-1, 0, 0); //new Vector3(tmp.X - 1, tmp.Y + 0, tmp.Z + 0);
83                      LeftTrigger.Transform.Dimensions = new Vector3(1, 30, 0);
84
85                      BottomTrigger.Transform.Position = tmp.Add(0, 32, 0); //new Vector3(tmp.X + 0, tmp.Y + 32, tmp.Z + 0);
86                      BottomTrigger.Transform.Dimensions = new Vector3(16, 2, 0);
87
88                      TopTrigger.Collider.Area = new RectangleF(0, 0, 15, 1);
89                      RightTrigger.Collider.Area = new RectangleF(0, 0, 1, 30);
90                      LeftTrigger.Collider.Area = new RectangleF(0, 0, 1, 30);
91                      BottomTrigger.Collider.Area = new RectangleF(0, 0, 16, 2);
92
93                      break;
94
95                  default:
96                      break;
97              }
98
99  #if DEBUG
100             TopTrigger.Model = new Material(Color.Black, TopTrigger);
101             RightTrigger.Model = new Material(Color.Black, RightTrigger);
102             LeftTrigger.Model = new Material(Color.Black, LeftTrigger);
103             BottomTrigger.Model = new Material(Color.Black, BottomTrigger);
104 #endif
105         }
106     }
107
108     public Movement CurrentMovement { get; set; }
109     public Direction PipeEnteredInDirection { get { return EnteredPipe.PipeEnterDirection; } }
110     public Block EnteredPipe { get; set; }
111
112     public WorldChangeManagerScript WorldManager { get; set; }
113
114     public Mario()
115     {
116         InitTriggers();
117     }
118
119     public Mario(GameObject Parent)
120         : base(Parent)
```

```
121         {
122             InitTriggers();
123         }
124 
125         public enum State
126         {
127             Dead,
128             Small,
129             Super,
130             Fire,
131             Invincible
132         }
133 
134         public enum Movement
135         {
136             Standing,
137             Crouching
138         }
139 
140         protected override void Initialize()
141         {
142             this.Name = "Player";
143 
144             this.InitNewComponent<Animator>();
145             this.Animator.AddAnimation(MARIO_IDLE_LEFT, MARIO_IDLE_LEFT_MAT);
146             this.Animator.AddAnimation(MARIO_IDLE_RIGHT, MARIO_IDLE_RIGHT_MAT);
147             this.Animator.AddAnimation(MARIO_JUMP_LEFT, MARIO_JUMP_LEFT_MAT);
148             this.Animator.AddAnimation(MARIO_JUMP_RIGHT, MARIO_JUMP_RIGHT_MAT);
149             this.Animator.AddAnimation(MARIO_MOVE_LEFT, MARIO_MOVE_LEFT_MAT);
150             this.Animator.AddAnimation(MARIO_MOVE_RIGHT, MARIO_MOVE_RIGHT_MAT);
151             this.Animator.AddAnimation(MARIO_DEAD, MARIO_DEAD_MAT);
152             this.Animator.AddAnimation(MARIO_CROUCHING_LEFT, MARIO_CROUCHING_LEFT_MAT);
153             this.Animator.AddAnimation(MARIO_CROUCHING_RIGHT, MARIO_CROUCHING_RIGHT_MAT);
154 
155             this.Animator.AddAnimation(MARIO_SUPER_IDLE_LEFT, MARIO_SUPER_IDLE_LEFT_MAT);
156             this.Animator.AddAnimation(MARIO_SUPER_IDLE_RIGHT, MARIO_SUPER_IDLE_RIGHT_MAT);
157             this.Animator.AddAnimation(MARIO_SUPER_JUMP_LEFT, MARIO_SUPER_JUMP_LEFT_MAT);
158             this.Animator.AddAnimation(MARIO_SUPER_JUMP_RIGHT, MARIO_SUPER_JUMP_RIGHT_MAT);
159             this.Animator.AddAnimation(MARIO_SUPER_MOVE_LEFT, MARIO_SUPER_MOVE_LEFT_MAT);
160             this.Animator.AddAnimation(MARIO_SUPER_MOVE_RIGHT, MARIO_SUPER_MOVE_RIGHT_MAT);
161             this.Animator.AddAnimation(MARIO_SUPER_POWERUP_LEFT, MARIO_SUPER_POWERUP_LEFT_MAT);
162             this.Animator.AddAnimation(MARIO_SUPER_POWERUP_RIGHT, MARIO_SUPER_POWERUP_RIGHT_MAT);
163             this.Animator.AddAnimation(MARIO_SUPER_CROUCHING_RIGHT, MARIO_SU-
PER_CROUCHING_RIGHT_MAT);
164             this.Animator.AddAnimation(MARIO_SUPER_CROUCHING_LEFT, MARIO_SU-
PER_CROUCHING_LEFT_MAT);
165 
166             /*this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT, Shared.Assets.Animati-
ons.MARIO_FIRE_IDLE_LEFT_MAT);
167             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_IDLE_RIGHT, Shared.Assets.Animati-
ons.MARIO_FIRE_IDLE_RIGHT_MAT);
168             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_JUMP_LEFT, Shared.Assets.Animati-
ons.MARIO_FIRE_JUMP_LEFT_MAT);
169             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_JUMP_RIGHT, Shared.Assets.Animati-
ons.MARIO_FIRE_JUMP_RIGHT_MAT);
170             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_MOVE_LEFT, Shared.Assets.Animati-
ons.MARIO_FIRE_MOVE_LEFT_MAT);
171             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT, Shared.Assets.Animati-
ons.MARIO_FIRE_MOVE_RIGHT_MAT);
172             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_POWERUP_LEFT, Shared.Assets.Anima-
tions.MARIO_FIRE_POWERUP_LEFT_MAT);
173             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_POWERUP_RIGHT, Shared.Assets.Ani-
mations.MARIO_FIRE_POWERUP_RIGHT_MAT);
174             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_CROUCHING_RIGHT, Shared.Assets.Ani-
mations.MARIO_FIRE_CROUCHING_RIGHT_MAT);
175             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_FIRE_CROUCHING_LEFT, Shared.Assets.Ani-
mations.MARIO_FIRE_CROUCHING_LEFT_MAT);*/
176 
177             /*this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_IDLE_LEFT, Shared.Assets.Animations.MA-
RIO_IDLE_LEFT_MAT);
178             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_IDLE_RIGHT, Shared.Assets.Animations.MA-
RIO_IDLE_RIGHT_MAT);
179             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_JUMP_LEFT, Shared.Assets.Animations.MA-
RIO_JUMP_LEFT_MAT);
180             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_JUMP_RIGHT, Shared.Assets.Animations.MA-
RIO_JUMP_RIGHT_MAT);
```

```
181             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_MOVE_LEFT, Shared.Assets.Animations.MA-
    RIO_MOVE_LEFT_MAT);
182             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_MOVE_RIGHT, Shared.Assets.Animations.MA-
    RIO_MOVE_RIGHT_MAT);
183             this.Animator.AddAnimation(Shared.Assets.Animations.MARIO_DEAD, Shared.Assets.Animations.MA-
    RIO_DEAD_MAT);*/
184
185             if (InitCharacterController)
186                 this.InitNewScript<CharacterController>();
187
188             if (InitCameraController)
189                 this.InitNewScript<CameraController>();
190
191             if (InitCollider)
192             {
193                 this.InitNewComponent<Collider>();
194             }
195
196             BottomTrigger.InitNewScript<BottomMarioChecker>();
197             LeftTrigger.InitNewScript<LeftMarioChecker>();
198             RightTrigger.InitNewScript<RightMarioChecker>();
199             TopTrigger.InitNewScript<TopMarioChecker>();
200
201             CurrentState = Shared.Mechanics.MarioCurrentState;
202
203             WorldManager = Behavior.Find<WorldChangeManagerScript>("worldManager");
204         }
205
206         private void InitTriggers()
207         {
208             BottomTrigger = new Trigger(this)
209             {
210                 Name = "Bottom_Trigger"
211             };
212             LeftTrigger = new Trigger(this)
213             {
214                 Name = "Left_Trigger"
215             };
216             TopTrigger = new Trigger(this)
217             {
218                 Name = "Top_Trigger"
219             };
220             RightTrigger = new Trigger(this)
221             {
222                 Name = "Right_Trigger"
223             };
224         }
225
226         public void PipeEnter(Block Pipe)
227         {
228             ChangeState = true;
229             EnteredPipe = Pipe;
230         }
231     }
232 }
```

2.2.2.16 Assets/Models/MusicPlayer.cs

```
1   using DKEngine.Core;
2   using MarIO.Assets.Scripts;
3
4   namespace MarIO.Assets.Models
5   {
6       public class MusicPlayer : GameObject
7       {
8           protected override void Initialize()
9           {
10              this.Name = "MusicPlayer";
11              this.InitNewScript<MusicScript>();
12          }
13      }
14  }
```

2.2.2.17 Assets/Models/SoundOutput.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
```

```
3
4   namespace MarIO.Assets.Models
5   {
6       public class SoundOutput : GameObject
7       {
8           protected override void Initialize()
9           {
10              this.Name = nameof(SoundOutput);
11              this.InitNewComponent<SoundSource>();
12              Shared.Mechanics.FXPlayer = this;
13          }
14      }
15  }
```

2.2.2.18 Assets/Models/Trigger.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3
4   namespace MarIO.Assets.Models
5   {
6       public class Trigger : GameObject
7       {
8           public Trigger()
9               : base()
10          { }
11
12          public Trigger(GameObject Parent)
13              : base(Parent)
14          { }
15
16          protected override void Initialize()
17          {
18              this.InitNewComponent<Collider>();
19              this.Collider.IsTrigger = true;
20          }
21      }
22  }
```

2.2.2.19 Assets/Scenes/About.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using DKEngine.Core.UI;
5   using MarIO.Assets.Models;
6   using System;
7   using System.Collections.Generic;
8   using System.Drawing;
9   using System.Linq;
10  using System.Text;
11  using System.Threading.Tasks;
12
13  namespace MarIO.Assets.Scenes
14  {
15      class About : Scene
16      {
17          public override void Init()
18          {
19              new Camera()
20              {
21                  BackGround = Shared.Mechanics.OverworldBackground.ToColor()
22              };
23
24              new Group()
25              {
26                  SizeInBlocks = new Vector3(1, 20, 0),
27                  Type = Block.BlockType.Ground2,
28                  InitCollider = true
29              }.Transform.Position = new Vector3(0, 0, 0);
30
31              new Group()
32              {
33                  SizeInBlocks = new Vector3(1, 20, 0),
34                  Type = Block.BlockType.Ground2,
35                  InitCollider = true
36              }.Transform.Position = new Vector3(48, 0, 0);
```

```
37
38              new Group()
39              {
40                  SizeInBlocks = new Vector3(2, 1, 0),
41                  Type = Block.BlockType.Ground2,
42                  InitCollider = true
43              }.Transform.Position = new Vector3(16, 224, 0);
44
45              new Block()
46              {
47                  InitCollider = true,
48                  Type = Block.BlockType.Pipe3,
49                  SpecialAction = GoBack
50              }.Transform.Position = new Vector3(16, 192, 1);
51
52              new Mario()
53              {
54                  InitCollider = true,
55                  InitCharacterController = true
56              }.Transform.Position = new Vector3(16, 80, 0);
57
58              var _Mario = new TextBlock()
59              {
60                  Foreground = Color.LawnGreen,
61                  FontSize = 6,
62                  HAlignment = Text.HorizontalAlignment.Center,
63                  IsGUI = true,
64                  Text = "MARIO",
65                  TextShadow = true,
66                  TextHAlignment = Text.HorizontalAlignment.Center
67              };
68              _Mario.Transform.Position += new Vector3(30, 20, 0);
69              _Mario.Transform.Dimensions = new Vector3(200, 30, 0);
70
71              var _author = new TextBlock()
72              {
73                  FontSize = 2,
74                  HAlignment = Text.HorizontalAlignment.Center,
75                  IsGUI = true,
76                  Text = "BY DAVID KNIERADL 2017",
77                  TextShadow = true,
78                  TextHAlignment = Text.HorizontalAlignment.Center
79              };
80              _author.Transform.Position += new Vector3(30, 80, 0);
81              _author.Transform.Dimensions = new Vector3(200, 30, 0);
82
83              var _using = new TextBlock()
84              {
85                  Foreground = Color.YellowGreen,
86                  FontSize = 3,
87                  HAlignment = Text.HorizontalAlignment.Center,
88                  IsGUI = true,
89                  Text = "Made with",
90                  TextShadow = true,
91                  TextHAlignment = Text.HorizontalAlignment.Center
92              };
93              _using.Transform.Position += new Vector3(30, 110, 0);
94              _using.Transform.Dimensions = new Vector3(200, 30, 0);
95
96              var _dkengine = new TextBlock()
97              {
98                  FontSize = 2,
99                  HAlignment = Text.HorizontalAlignment.Center,
100                 IsGUI = true,
101                 Text = "DKENGINE",
102                 TextShadow = true,
103                 TextHAlignment = Text.HorizontalAlignment.Center
104             };
105             _dkengine.Transform.Position += new Vector3(30, 140, 0);
106             _dkengine.Transform.Dimensions = new Vector3(200, 30, 0);
107
108             var _naudio = new TextBlock()
109             {
110                 FontSize = 2,
111                 HAlignment = Text.HorizontalAlignment.Center,
112                 IsGUI = true,
113                 Text = "NAUDIO",
```

```
114                 TextShadow = true,
115                 TextHAlignment = Text.HorizontalAlignment.Center
116             };
117             _naudio.Transform.Position += new Vector3(30, 155, 0);
118             _naudio.Transform.Dimensions = new Vector3(200, 30, 0);
119
120             var _ver = new TextBlock()
121             {
122                 FontSize = 1,
123                 HAlignment = Text.HorizontalAlignment.Center,
124                 IsGUI = true,
125                 Text = "version 0.0.1 alpha",
126                 TextShadow = true,
127                 TextHAlignment = Text.HorizontalAlignment.Center
128             };
129             _ver.Transform.Position += new Vector3(30, 190, 0);
130             _ver.Transform.Dimensions = new Vector3(200, 30, 0);
131
132             new MusicPlayer();
133             new SoundOutput();
134             new BackgroundWorker();
135         }
136
137         public override void Unload()
138         { }
139
140         private void GoBack()
141         {
142             Engine.ChangeScene(nameof(MainMenu), true);
143         }
144     }
145 }
```

2.2.2.20 Assets/Scenes/GameOver.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using DKEngine.Core.UI;
5   using MarIO.Assets.Models;
6   using MarIO.Assets.Models.Miscellaneous;
7   using System;
8   using System.Drawing;
9
10  namespace MarIO.Assets.Scenes
11  {
12      internal class GameOver : Scene
13      {
14          public GameOver()
15          {
16              Name = nameof(GameOver);
17          }
18
19          public override void Init()
20          {
21              TextBlock GameOver = new TextBlock()
22              {
23                  FontSize = 5,
24                  Foreground = Color.White,
25                  HAlignment = Text.HorizontalAlignment.Center,
26                  IsGUI = true,
27                  Name = "tx_GameOver",
28                  Text = "GAME OVER",
29                  TextHAlignment = Text.HorizontalAlignment.Center,
30                  VAlignment = Text.VerticalAlignment.Center,
31              };
32              GameOver.Transform.Dimensions = new Vector3(200, 30, 0);
33              GameOver.Transform.Position += new Vector3(0, -30, 0);
34
35              TextBlock Score = new TextBlock()
36              {
37                  FontSize = 2.5f,
38                  Foreground = Color.White,
39                  HAlignment = Text.HorizontalAlignment.Center,
40                  IsGUI = true,
41                  Name = "tx_Score",
42                  Text = Shared.Mechanics.GameScoreStr,
```

```
43                  TextHAlignment = Text.HorizontalAlignment.Center,
44                  VAlignment = Text.VerticalAlignment.Center
45              };
46              Score.Transform.Dimensions = new Vector3(100, 30, 0);
47              Score.Transform.Position += new Vector3(0, 30, 0);
48
49              GameObject holder = new GameObject();
50              holder.Transform.Position = new Vector3(136, 156, 0);
51
52              Coin CoinIcon = new Coin(holder)
53              {
54                  IsGUI = true,
55                  Name = "coin_icon"
56              };
57              CoinIcon.Transform.Scale = new Vector3(2f, 2f, 0);
58
59              TextBlock Coins = new TextBlock(holder)
60              {
61                  FontSize = 2.5f,
62                  IsGUI = true,
63                  TextHAlignment = Text.HorizontalAlignment.Center,
64                  Text = string.Format($"*{Shared.Mechanics.CoinsCount:00}")
65              };
66              Coins.Transform.Dimensions = new Vector3(40, 15, 0);
67              Coins.Transform.Position += new Vector3(12, 2, 0);
68
69              new Delayer()
70              {
71                  CalledAction = () => Engine.LoadScene<MainMenu>(),
72                  TimeToWait = new TimeSpan(0, 0, 5)
73              };
74
75              new Camera()
76              {
77                  BackGround = Shared.Mechanics.WorldChangeBackground.ToColor()
78              };
79
80              Shared.Mechanics.MarioCurrentState = Mario.State.Super;
81          }
82
83          public override void Unload()
84          { }
85      }
86  }
```

2.2.2.21 Assets/Scenes/Level_1_1.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3   using MarIO.Assets.Models;
4   using MarIO.Assets.Scripts;
5
6   namespace MarIO.Assets.Scenes
7   {
8       public class Level_1_1 : MapBase
9       {
10          private const int offset = 520;
11
12          public Level_1_1()
13          {
14              Name = MapBase.LevelsNames[nameof(Level_1_1)];
15              Shared.Mechanics.LastWorldType = typeof(Level_1_1);
16          }
17
18          public override void Load()
19          {
20              /*---------- BG PRESET ----------*/
21              for (int i = 0; i < 8; i++)
22              {
23                  new Block()
24                  {
25                      Name = $"cloud_1_{i}",
26                      Type = Block.BlockType.Cloud1,
27                  }.Transform.Position = new Vector3(50 + i * offset, -82, -2);
28
29                  new Block()
30                  {
```

```
                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()
```

```
108             {
109                 Name = "ground4",
110                 InitCollider = true,
111                 Type = Block.BlockType.Ground2
112             };
113             _4.SizeInBlocks = new Vector3(100, 3, 0);
114             _4.Transform.Position = new Vector3(2544, 48, 0);
115
116             #endregion GROUND
117
118             #region Platform1
119
120             new Block()
121             {
122                 Name = "bonus_1",
123                 Type = Block.BlockType.Ground1,
124                 CoinCount = 1,
125                 InitCollider = true
126             }.Transform.Position = new Vector3(320, -12, 0);
127
128             new Block()
129             {
130                 Name = "platform_1",
131                 Type = Block.BlockType.Ground4,
132                 InitCollider = true
133             }.Transform.Position = new Vector3(400, -12, 0);
134
135             new Block()
136             {
137                 Name = "platform_1",
138                 Type = Block.BlockType.Ground1,
139                 InitCollider = true,
140                 CoinCount = 0,
141                 PowerUp = true
142             }.Transform.Position = new Vector3(416, -12, 0);
143
144             new Block()
145             {
146                 Name = "platform_1",
147                 Type = Block.BlockType.Ground4,
148                 InitCollider = true
149             }.Transform.Position = new Vector3(432, -12, 0);
150
151             new Block()
152             {
153                 Name = "platform_1",
154                 Type = Block.BlockType.Ground1,
155                 InitCollider = true,
156                 CoinCount = 1
157             }.Transform.Position = new Vector3(432, -76, 0);
158
159             new Block()
160             {
161                 Name = "platform_1",
162                 CoinCount = 1,
163                 Type = Block.BlockType.Ground1,
164                 InitCollider = true
165             }.Transform.Position = new Vector3(448, -12, 0);
166
167             new Block()
168             {
169                 Name = "platform_1",
170                 Type = Block.BlockType.Ground4,
171                 InitCollider = true
172             }.Transform.Position = new Vector3(464, -12, 0);
173
174             #endregion Platform1
175
176             new Block()
177             {
178                 Name = "pipe",
179                 Type = Block.BlockType.Pipe3
180             }.Transform.Position = new Vector3(544, 16, 1);
181
182             new Goomba().Transform.Position = new Vector3(600, 18, 0);
183
184             {
```

```
185             GameObject holder = new GameObject();
186             holder.Transform.Dimensions = new Vector3(32, 64, 0);
187             holder.Transform.Position = new Vector3(700, 32, 0);
188             holder.InitNewComponent<Collider>();
189
190             new Block(holder)
191             {
192                 Name = "pipe",
193                 Type = Block.BlockType.Pipe4
194             }.Transform.Position += new Vector3(0, 0, -1);
195
196             new Block(holder)
197             {
198                 Name = "pipe",
199                 Type = Block.BlockType.Pipe3
200             }.Transform.Position += new Vector3(0, -32, 1);
201         }
202
203         new Goomba().Transform.Position = new Vector3(760, 18, 0);
204         new Goomba().Transform.Position = new Vector3(800, 18, 0);
205
206         {
207             GameObject holder = new GameObject();
208             holder.Transform.Dimensions = new Vector3(32, 64, 0);
209             holder.Transform.Position = new Vector3(860, 16, 0);
210             holder.InitNewComponent<Collider>();
211
212             new Block(holder)
213             {
214                 Name = "pipe",
215                 Type = Block.BlockType.Pipe4
216             }.Transform.Position += new Vector3(0, 0, -1);
217
218             new Block(holder)
219             {
220                 Name = "pipe",
221                 Type = Block.BlockType.Pipe3
222             }.Transform.Position += new Vector3(0, -32, 1);
223         }
224
225         new Block()
226         {
227             Type = Block.BlockType.Ground4,
228             InitCollider = true
229         }.Transform.Position = new Vector3(1184, -12, 0);
230
231         new Block()
232         {
233             Type = Block.BlockType.Ground1,
234             CoinCount = 3,
235             InitCollider = true
236         }.Transform.Position = new Vector3(1200, -12, 0);
237
238         new Block()
239         {
240             Type = Block.BlockType.Ground4,
241             InitCollider = true
242         }.Transform.Position = new Vector3(1216, -12, 0);
243
244         for (int i = 0; i < 10; i++)
245         {
246             new Block()
247             {
248                 Type = Block.BlockType.Ground4,
249                 InitCollider = true
250             }.Transform.Position = new Vector3(1232 + i * 16, -76, 0);
251         }
252
253         for (int i = 0; i < 3; i++)
254         {
255             new Block()
256             {
257                 Type = Block.BlockType.Ground4,
258                 InitCollider = true
259             }.Transform.Position = new Vector3(1440 + i * 16, -76, 0);
260         }
261
```

```
262             new Block()
263             {
264                 Type = Block.BlockType.Ground1,
265                 InitCollider = true,
266                 CoinCount = 1
267             }.Transform.Position = new Vector3(1488, -76, 0);
268
269             new Block()
270             {
271                 Type = Block.BlockType.Ground4,
272                 InitCollider = true,
273                 CoinCount = 5
274             }.Transform.Position = new Vector3(1488, -12, 0);
275
276             new Block()
277             {
278                 Type = Block.BlockType.Ground4,
279                 InitCollider = true,
280                 CoinCount = 5
281             }.Transform.Position = new Vector3(1616, -12, 0);
282
283             new Block()
284             {
285                 Type = Block.BlockType.Ground4,
286                 InitCollider = true,
287                 CoinCount = 1
288             }.Transform.Position = new Vector3(1632, -12, 0);
289
290             #region Bonus Field
291
292             new Block()
293             {
294                 Type = Block.BlockType.Ground1,
295                 InitCollider = true,
296                 CoinCount = 1
297             }.Transform.Position = new Vector3(1680, -12, 0);
298
299             new Block()
300             {
301                 Type = Block.BlockType.Ground1,
302                 InitCollider = true,
303                 CoinCount = 1
304             }.Transform.Position = new Vector3(1744, -12, 0);
305
306             new Block()
307             {
308                 Type = Block.BlockType.Ground1,
309                 InitCollider = true,
310                 PowerUp = true
311             }.Transform.Position = new Vector3(1744, -76, 0);
312
313             new Block()
314             {
315                 Type = Block.BlockType.Ground1,
316                 InitCollider = true,
317                 CoinCount = 1
318             }.Transform.Position = new Vector3(1808, -12, 0);
319
320             #endregion Bonus Field
321
322             new Block()
323             {
324                 Type = Block.BlockType.Ground4,
325                 InitCollider = true
326             }.Transform.Position = new Vector3(1968, -12, 0);
327
328             for (int i = 0; i < 3; i++)
329             {
330                 new Block()
331                 {
332                     Type = Block.BlockType.Ground4,
333                     InitCollider = true
334                 }.Transform.Position = new Vector3(2000 + i * 16, -76, 0);
335             }
336
337             new Block()
338             {
```

```
            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);

```

```
416             new Group()
417             {
418                 InitCollider = true,
419                 SizeInBlocks = new Vector3(3, 1, 0),
420                 Type = Block.BlockType.Ground3
421             }.Transform.Position = new Vector3(2288, 16, 0);
422
423             new Group()
424             {
425                 InitCollider = true,
426                 SizeInBlocks = new Vector3(2, 1, 0),
427                 Type = Block.BlockType.Ground3
428             }.Transform.Position = new Vector3(2288, 0, 0);
429
430             new Group()
431             {
432                 InitCollider = true,
433                 SizeInBlocks = new Vector3(1, 1, 0),
434                 Type = Block.BlockType.Ground3
435             }.Transform.Position = new Vector3(2288, -16, 0);
436
437             #endregion Stairs2
438
439             #region Stairs3
440
441             new Group()
442             {
443                 InitCollider = true,
444                 SizeInBlocks = new Vector3(5, 1, 0),
445                 Type = Block.BlockType.Ground3
446             }.Transform.Position = new Vector3(2432, 32, 0);
447
448             new Group()
449             {
450                 InitCollider = true,
451                 SizeInBlocks = new Vector3(4, 1, 0),
452                 Type = Block.BlockType.Ground3
453             }.Transform.Position = new Vector3(2448, 16, 0);
454
455             new Group()
456             {
457                 InitCollider = true,
458                 SizeInBlocks = new Vector3(3, 1, 0),
459                 Type = Block.BlockType.Ground3
460             }.Transform.Position = new Vector3(2464, 0, 0);
461
462             new Group()
463             {
464                 InitCollider = true,
465                 SizeInBlocks = new Vector3(2, 1, 0),
466                 Type = Block.BlockType.Ground3
467             }.Transform.Position = new Vector3(2480, -16, 0);
468
469             #endregion Stairs3
470
471             #region Stairs4
472
473             new Group()
474             {
475                 InitCollider = true,
476                 SizeInBlocks = new Vector3(4, 1, 0),
477                 Type = Block.BlockType.Ground3
478             }.Transform.Position = new Vector3(2544, 32, 0);
479
480             new Group()
481             {
482                 InitCollider = true,
483                 SizeInBlocks = new Vector3(3, 1, 0),
484                 Type = Block.BlockType.Ground3
485             }.Transform.Position = new Vector3(2544, 16, 0);
486
487             new Group()
488             {
489                 InitCollider = true,
490                 SizeInBlocks = new Vector3(2, 1, 0),
491                 Type = Block.BlockType.Ground3
492             }.Transform.Position = new Vector3(2544, 0, 0);
```

```
493
494         new Group()
495         {
496             InitCollider = true,
497             SizeInBlocks = new Vector3(1, 1, 0),
498             Type = Block.BlockType.Ground3
499         }.Transform.Position = new Vector3(2544, -16, 0);
500
501         #endregion Stairs4
502
503         new Block()
504         {
505             Type = Block.BlockType.Pipe3
506         }.Transform.Position = new Vector3(2704, 16, 1);
507
508         new Block()
509         {
510             Type = Block.BlockType.Ground4,
511             InitCollider = true
512         }.Transform.Position = new Vector3(2768, -12, 0);
513
514         new Block()
515         {
516             Type = Block.BlockType.Ground4,
517             InitCollider = true
518         }.Transform.Position = new Vector3(2784, -12, 0);
519
520         new Block()
521         {
522             Type = Block.BlockType.Ground1,
523             InitCollider = true,
524             CoinCount = 1
525         }.Transform.Position = new Vector3(2800, -12, 0);
526
527         new Block()
528         {
529             Type = Block.BlockType.Ground4,
530             InitCollider = true
531         }.Transform.Position = new Vector3(2816, -12, 0);
532
533         new Block()
534         {
535             Type = Block.BlockType.Pipe3
536         }.Transform.Position = new Vector3(2928, 16, 1);
537
538         #region Stairs5
539
540         new Group()
541         {
542             InitCollider = true,
543             SizeInBlocks = new Vector3(7, 1, 0),
544             Type = Block.BlockType.Ground3
545         }.Transform.Position = new Vector3(2960, 32, 0);
546
547         new Group()
548         {
549             InitCollider = true,
550             SizeInBlocks = new Vector3(6, 1, 0),
551             Type = Block.BlockType.Ground3
552         }.Transform.Position = new Vector3(2976, 16, 0);
553
554         new Group()
555         {
556             InitCollider = true,
557             SizeInBlocks = new Vector3(5, 1, 0),
558             Type = Block.BlockType.Ground3
559         }.Transform.Position = new Vector3(2992, 0, 0);
560
561         new Group()
562         {
563             InitCollider = true,
564             SizeInBlocks = new Vector3(4, 1, 0),
565             Type = Block.BlockType.Ground3
566         }.Transform.Position = new Vector3(3008, -16, 0);
567
568         new Group()
569         {
```

```
570                 InitCollider = true,
571                 SizeInBlocks = new Vector3(3, 1, 0),
572                 Type = Block.BlockType.Ground3
573             }.Transform.Position = new Vector3(3024, -32, 0);
574
575             new Group()
576             {
577                 InitCollider = true,
578                 SizeInBlocks = new Vector3(2, 1, 0),
579                 Type = Block.BlockType.Ground3
580             }.Transform.Position = new Vector3(3040, -48, 0);
581
582             new Group()
583             {
584                 InitCollider = true,
585                 SizeInBlocks = new Vector3(1, 1, 0),
586                 Type = Block.BlockType.Ground3
587             }.Transform.Position = new Vector3(3056, -64, 0);
588
589             #endregion Stairs5
590
591             new Block()
592             {
593                 Type = Block.BlockType.CastleBig
594             }.Transform.Position = new Vector3(3216, -152, -1);
595
596             Trigger EndOfWorld = new Trigger();
597             EndOfWorld.Transform.Position = new Vector3(3216, -40, 0);
598             EndOfWorld.Transform.Dimensions = new Vector3(200, 80, 0);
599             EndOfWorld.InitNewScript<WorldEnd>();
600
601             Mario m = new Mario()
602             {
603                 InitCameraController = true,
604                 InitCharacterController = true,
605                 InitCollider = true
606             };
607             m.Transform.Position = new Vector3(10, -10, 0);
608
609             Camera c = new Camera()
610             {
611                 BackGround = Shared.Mechanics.OverworldBackground.ToColor()
612             };
613
614             new SoundOutput();
615             new GUIUpdater();
616             new BackgroundWorker();
617             new MusicPlayer();
618
619             Trigger DeathZone = new Trigger();
620             DeathZone.InitNewScript<DeathZoneScript>();
621             DeathZone.Transform.Dimensions = new Vector3(5000, 10, 0);
622             DeathZone.Transform.Position = new Vector3(0, 100, 0);
623         }
624     }
625 }
```

2.2.2.22 Assets/Scenes/MainMenu.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using DKEngine.Core.UI;
5   using MarIO.Assets.Models;
6   using MarIO.Assets.Scripts;
7   using System;
8   using System.Drawing;
9
10  namespace MarIO.Assets.Scenes
11  {
12      public class MainMenu : Scene
13      {
14          public MainMenu()
15          {
16              Name = nameof(MainMenu);
17          }
18
```

```
19          public override void Init()
20          {
21              Group wall4 = new Group()
22              {
23                  InitCollider = true,
24                  Name = "Wall_4",
25                  SizeInBlocks = new Vector3(21, 2, 0),
26                  Type = Block.BlockType.Ground2
27              };
28              wall4.Transform.Position = new Vector3(0, 16 * 13, 0);
29
30              Group wall5 = new Group()
31              {
32                  InitCollider = true,
33                  Name = "Wall_5",
34                  SizeInBlocks = new Vector3(22, 1, 0),
35                  Type = Block.BlockType.Ground2
36              };
37              wall5.Transform.Position = new Vector3(0, 16 * 9, 0);
38
39              Block pipe1 = new Block()
40              {
41                  Name = "Pipe_1_Play",
42                  Type = Block.BlockType.Pipe3
43              };
44              pipe1.Transform.Position = new Vector3(32, 16 * 7, 1);
45              pipe1.SpecialAction = Play;
46
47              Block pipe2 = new Block()
48              {
49                  Name = "Pipe_2_About",
50                  Type = Block.BlockType.Pipe3
51              };
52              pipe2.Transform.Position = new Vector3(143, 16 * 7, 1);
53
54              Block pipe3 = new Block()
55              {
56                  Name = "Pipe_3_Exit",
57                  Type = Block.BlockType.Pipe3
58              };
59              pipe3.Transform.Position = new Vector3(256, 16 * 7, 1);
60              pipe3.SpecialAction = Exit;
61
62              Camera baseCam = new Camera()
63              {
64                  BackGround = Shared.Mechanics.OverworldBackground.ToColor()
65              };
66
67              Mario player = new Mario()
68              {
69                  InitCharacterController = true,
70                  InitCollider = true
71              };
72
73              TextBlock MainMenuHeader = new TextBlock()
74              {
75                  FontSize = 6,
76                  HAlignment = Text.HorizontalAlignment.Center,
77                  Name = "tx_MainMenuHeader",
78                  Text = "MARIO",
79                  TextHAlignment = Text.HorizontalAlignment.Center,
80                  TextShadow = true
81              };
82              MainMenuHeader.Transform.Position += new Vector3(0, 10, 0);
83              MainMenuHeader.Transform.Dimensions = new Vector3(200, 50, 0);
84
85              TextBlock PlayText = new TextBlock()
86              {
87                  Name = "tx_Play",
88                  Text = "Play",
89                  TextHAlignment = Text.HorizontalAlignment.Center,
90                  TextShadow = true
91              };
92              PlayText.Transform.Position = new Vector3(9, 96, -1);
93              PlayText.Transform.Dimensions = new Vector3(80, 20, 0);
94
95              TextBlock OptionsText = new TextBlock()
```

```
96          {
97              Name = "tx_Options",
98              Text = "About",
99              TextHAlignment = Text.HorizontalAlignment.Center,
100             TextShadow = true,
101             HAlignment = Text.HorizontalAlignment.Center
102         };
103         OptionsText.Transform.Position += new Vector3(0, 96, -1);
104         OptionsText.Transform.Dimensions = new Vector3(80, 20, 0);
105
106         TextBlock ExitText = new TextBlock()
107         {
108             Name = "tx_Exit",
109             Text = "Exit",
110             TextHAlignment = Text.HorizontalAlignment.Center,
111             TextShadow = true,
112             HAlignment = Text.HorizontalAlignment.Right
113         };
114         ExitText.Transform.Position += new Vector3(-8, 96, -1);
115         ExitText.Transform.Dimensions = new Vector3(80, 20, 0);
116
117         Block cloud1 = new Block()
118         {
119             Name = "cloud_1",
120             Type = Block.BlockType.Cloud3
121         };
122         cloud1.Transform.Position = new Vector3(-10, 20, -1);
123
124         Block cloud2 = new Block()
125         {
126             Name = "cloud_2",
127             Type = Block.BlockType.Cloud1
128         };
129         cloud2.Transform.Position = new Vector3(120, -15, -1);
130
131         Block cloud3 = new Block()
132         {
133             Name = "cloud_3",
134             Type = Block.BlockType.Cloud2
135         };
136         cloud3.Transform.Position = new Vector3(180, 34, -1);
137
138         Block mountain = new Block()
139         {
140             Name = "mountain",
141             Type = Block.BlockType.Mountain
142         };
143         mountain.Transform.Position = new Vector3(100, 152, -1);
144         mountain.Transform.Scale = new Vector3(2, 2, 0);
145
146         Block bush1 = new Block()
147         {
148             Name = "bush_1",
149             Type = Block.BlockType.Bush3
150         };
151         bush1.Transform.Position = new Vector3(180, 182, -1);
152
153         Block bush2 = new Block()
154         {
155             Name = "bush_2",
156             Type = Block.BlockType.Bush2
157         };
158         bush2.Transform.Position = new Vector3(25, 182, -1);
159
160         Block fence1 = new Block()
161         {
162             Name = "fence_1",
163             Type = Block.BlockType.Fence
164         };
165         fence1.Transform.Position = new Vector3(90, 192, -1);
166
167         Block fence2 = new Block()
168         {
169             Name = "fence_2",
170             Type = Block.BlockType.Fence
171         };
172         fence2.Transform.Position = new Vector3(106, 192, -1);
```

```
173
174             Block fence3 = new Block()
175             {
176                 Name = "fence_3",
177                 Type = Block.BlockType.Fence
178             };
179             fence3.Transform.Position = new Vector3(122, 192, -1);
180
181             Blocker leftSide = new Blocker()
182             {
183                 Name = "LeftSideBlocker"
184             };
185             leftSide.Transform.Position = new Vector3(-10, -20, 0);
186             leftSide.Transform.Dimensions = new Vector3(10, 148, 0);
187
188             Blocker rightSide = new Blocker()
189             {
190                 Name = "LeftSideBlocker"
191             };
192             rightSide.Transform.Position = new Vector3(320, -20, 0);
193             rightSide.Transform.Dimensions = new Vector3(10, 148, 0);
194
195             BackgroundWorker BW = new BackgroundWorker();
196             BW.InitNewComponent<Collider>();
197             BW.Collider.Area = new RectangleF(-10, 160, 10, 30);
198             BW.Collider.IsTrigger = true;
199             BW.InitNewScript<MainMenuSpawnScript>();
200
201             new MusicPlayer();
202             new SoundOutput();
203         }
204
205         public override void Set(params object[] Args)
206         { }
207
208         public override void Unload()
209         { }
210
211         private void Exit()
212         {
213             Environment.Exit(1);
214         }
215
216         private void Play()
217         {
218             Shared.Mechanics.MarioCurrentState = Mario.State.Small;
219             Shared.Mechanics.CoinsCount = 0;
220             Shared.Mechanics.GameScore = 0;
221             Shared.Mechanics.Lives = 3;
222             Shared.Mechanics.TimeCounter.Reset();
223
224             Engine.ChangeScene(nameof(WorldScreen), true, new object[] { (Action)(() => Engine.ChangeS-
    cene(MapBase.LevelsNames[nameof(Level_1_1)], true)), $"world:get|{nameof(Level_1_1)}" });
225         }
226     }
227 }
```

2.2.2.23 Assets/Scenes/MapBase.cs

```
1   using DKEngine.Core;
2   using System.Collections.Generic;
3
4   namespace MarIO.Assets.Scenes
5   {
6       public abstract class MapBase : Scene
7       {
8           public static Dictionary<string, string> LevelsNames = new Dictionary<string, string>()
9           {
10              { nameof(Test), "test" },
11              { nameof(Level_1_1), "1-1" }
12          };
13
14          public sealed override void Init()
15          {
16              Load();
17
18              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()
```

```
26          {
27              Name = nameof(WorldScreen);
28          }
29
30          public override void Init()
31          {
32              TextBlock _World = new TextBlock()
33              {
34                  FontSize = 5,
35                  Foreground = Color.White,
36                  HAlignment = Text.HorizontalAlignment.Center,
37                  IsGUI = true,
38                  Name = "tx_const_world",
39                  Text = "WORLD",
40                  TextHAlignment = Text.HorizontalAlignment.Center,
41                  VAlignment = Text.VerticalAlignment.Center
42              };
43              _World.Transform.Position += new Vector3(0, -40, 0);
44              _World.Transform.Dimensions = new Vector3(120, 30, 0);
45
46              World = new TextBlock()
47              {
48                  FontSize = 4,
49                  Foreground = Color.White,
50                  HAlignment = Text.HorizontalAlignment.Center,
51                  IsGUI = true,
52                  Name = "tx_world",
53                  TextHAlignment = Text.HorizontalAlignment.Center,
54                  VAlignment = Text.VerticalAlignment.Center,
55                  Text = WorldName
56              };
57              World.Transform.Position += new Vector3(0, -5, 0);
58              World.Transform.Dimensions = new Vector3(100, 30, 0);
59
60              GameObject holder = new GameObject();
61              holder.Transform.Position = new Vector3(120, 140, 0);
62
63              Heart _HeartIcon = new Heart(holder)
64              {
65                  IsGUI = true,
66                  Name = "heart_icon"
67              };
68
69              _HeartIcon.Transform.Scale = new Vector3(3, 3, 0);
70
71              Lives = new TextBlock(holder)
72              {
73                  FontSize = 3.5f,
74                  IsGUI = true,
75                  TextHAlignment = Text.HorizontalAlignment.Center,
76                  Text = RemainingLives
77              };
78              Lives.Transform.Dimensions = new Vector3(40, 15, 0);
79              Lives.Transform.Position += new Vector3(32, 8, 0);
80
81              Delayer = new Delayer()
82              {
83                  CalledAction = WorldChange,
84                  TimeToWait = Delay ?? _defautlTimeSpan
85              };
86
87              new Camera()
88              {
89                  BackGround = Shared.Mechanics.WorldChangeBackground.ToColor()
90              };
91
92              if (Shared.Mechanics.MarioCurrentState == Mario.State.Dead)
93                  Shared.Mechanics.MarioCurrentState = Mario.State.Small;
94          }
95
96          public override void Set(params object[] args)
97          {
98              if (args == null)
99                  return;
100
101             string[] stringParameters = args.Where(obj => obj is string).ToList().Cast<string>().ToArray();
102             object[] otherParameters = args.Where(obj => !(obj is string)).ToArray();
```

```
103
104             for (int i = 0; i < stringParameters.Length; i++)
105             {
106                 string[] parameters = stringParameters[i].Split(':');
107
108                 switch (parameters[0])
109                 {
110                     case "world":
111                         if (parameters[1].Split('|')[0] == "get")
112                         {
113                             WorldName = MapBase.LevelsNames[parameters[1].Split('|')[1]];
114                         }
115                         else
116                         {
117                             WorldName = parameters[1];
118                         }
119
120                         break;
121
122                     case "time":
123                         Delay = TimeSpan.Parse(parameters[1]);
124                         break;
125                 }
126             }
127
128             foreach (object item in otherParameters)
129             {
130                 if (item is Action)
131                 {
132                     WorldChange = ((Action)item);
133                 }
134             }
135
136             RemainingLives = string.Format($"*{Shared.Mechanics.Lives:00}");
137         }
138
139         public override void Unload()
140         { }
141     }
142 }
```

2.2.2.26 Assets/Scripts/BlockAnimatorScript.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using MarIO.Assets.Models;
5
6   namespace MarIO.Assets.Scripts
7   {
8       public class BlockAnimatorScript : Script
9       {
10          private float AnimationHeight = 2;
11          private float AnimationSpeed = 20;
12
13          public BlockAnimatorScript(GameObject Parent)
14              : base(Parent)
15          { }
16
17          protected override void OnColliderEnter(Collider e)
18          {
19          }
20
21          protected override void Start()
22          {
23          }
24
25          protected override void Update()
26          {
27              if (Shared.AnimatedWorldReferences.BlocksToUpdate.Count > 0)
28              {
29                  for (int i = 0; i < Shared.AnimatedWorldReferences.BlocksToUpdate.Count; i++)
30                  {
31                      float StartBlockY = Shared.AnimatedWorldReferences.BlocksStartPositions[i];
32                      Block CurrentBlock = Shared.AnimatedWorldReferences.BlocksToUpdate[i];
33
```

```
34                  if (CurrentBlock.State == Block.CollisionState.Up && StartBlockY - AnimationHeight < CurrentBlock.Trans-
    form.Position.Y)
35                  {
36                      CurrentBlock.Transform.Position -= new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);
37
38                      if (CurrentBlock.Transform.Position.Y <= StartBlockY - AnimationHeight)
39                      {
40                          CurrentBlock.State = Block.CollisionState.Down;
41                      }
42                  }
43                  else if (CurrentBlock.State == Block.CollisionState.Down && CurrentBlock.Transform.Position.Y < Start-
    BlockY)
44                  {
45                      CurrentBlock.Transform.Position += new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);
46
47                      if (CurrentBlock.Transform.Position.Y > StartBlockY)
48                      {
49                          CurrentBlock.State = Block.CollisionState.Stay;
50                          CurrentBlock.Transform.Position = new Vector3(CurrentBlock.Transform.Position.X, StartBlockY,
    CurrentBlock.Transform.Position.Z);
51
52                          Shared.AnimatedWorldReferences.BlocksStartPositions.RemoveAt(i);
53                          Shared.AnimatedWorldReferences.BlocksToUpdate.RemoveAt(i);
54
55                          CurrentBlock.CoinGot = false;
56
57                          i--;
58                      }
59                  }
60              }
61          }
62      }
63  }
64  }
```

2.2.2.27 Assets/Scripts/CameraController.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4
5   namespace MarIO.Assets.Scripts
6   {
7       public class CameraController : Script
8       {
9           private GameObject Player;
10
11          private Camera TargetCam;
12          private float PositionX;
13          private float MaxCameraDistance;
14
15          private Vector3 Offset;
16
17          public CameraController(GameObject Parent)
18              : base(Parent)
19          { }
20
21          protected override void OnColliderEnter(Collider e)
22          { }
23
24          protected override void Start()
25          {
26              MaxCameraDistance = Engine.Render.RenderWidth / 3;
27              Offset = new Vector3(20, 0, 0);
28
29              Player = GameObject.Find<GameObject>("Player");
30              TargetCam = Component.Find<Camera>("Camera");
31              TargetCam.Position = new Vector3(0, -160, 0);
32          }
33
34          protected override void Update()
35          {
36              if (Player.Transform.Position.X - TargetCam.Position.X > MaxCameraDistance)
37              {
38                  TargetCam.Position += new Vector3(Player.Transform.Position.X - PositionX, 0, 0);
39              }
40
```

```
41              if (Player.Transform.Position.X < TargetCam.Position.X)
42              {
43                  Player.Transform.Position = Player.Transform.Position.Add(TargetCam.Position.X - Player.Transform.Posi-
    tion.X, 0, 0);
44              }
45
46              PositionX = Player.Transform.Position.X;
47          }
48      }
49  }
```

2.2.2.28 Assets/Scripts/CharacterController.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using MarIO.Assets.Models;
5   using MarIO.Assets.Scenes;
6   using System;
7   using static DKEngine.Core.Components.Transform;
8
9   namespace MarIO.Assets.Scripts
10  {
11      public class CharacterController : Script
12      {
13          public bool Enabled = true;
14
15          private Animator PlayerAnimator;
16          private Mario Player;
17          //private SoundSource SoundOutput;
18
19          private float horiSpeed = 0;
20          private float vertSpeed = 0;
21
22          private const float MovementSpeed = 120f;
23          private const float FloatSpeed = 300f;
24
25          private const float Acceleration = 3.5f;
26
27          private const float DeathAnimSpeed = 120f;
28
29          private bool CanJump = true;
30          private bool IsFalling = false;
31          private bool Jumped = false;
32          private bool IsFacingLeft = false;
33          private bool EnemyKilledAnim = false;
34          private bool FirstTimeDeadAnimPlay = true;
35
36          private bool FirstTimePipeEnter = true;
37          private float PipeEnterStartPosition;
38          private float PipeEnterSpeed = 50f;
39
40          private readonly TimeSpan WorldReload = new TimeSpan(0, 0, 3);
41          private TimeSpan WorldReloadNow = new TimeSpan();
42
43          private Mario.State LastState;
44          private bool ChangingState = false;
45
46          private string _idle
47          {
48              get
49              {
50                  switch (Player.CurrentState)
51                  {
52                      case Mario.State.Dead:
53                      case Mario.State.Small:
54                          return IsFacingLeft ? Shared.Assets.Animations.MARIO_IDLE_LEFT : Shared.Assets.Animations.MA-
    RIO_IDLE_RIGHT;
55
56                      case Mario.State.Super:
57                          return IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_IDLE_LEFT : Shared.Assets.Animati-
    ons.MARIO_SUPER_IDLE_RIGHT;
58
59                      case Mario.State.Fire:
60                          return IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT : Shared.Assets.Animati-
    ons.MARIO_FIRE_IDLE_RIGHT;
61
```

```
62                  /*case Mario.State.Invincible:
63                      return IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_LEFT : Shared.Assets.Ani-
mations.MARIO_INVINCIBLE_IDLE_RIGHT;*/
64
65                  default:
66                      throw new Exception("JAK");
67              }
68          }
69      }
70
71      private string _crouch
72      {
73          get
74          {
75              switch (Player.CurrentState)
76              {
77                  case Mario.State.Small:
78                      return IsFacingLeft ? Shared.Assets.Animations.MARIO_CROUCHING_LEFT : Shared.Assets.Animati-
ons.MARIO_CROUCHING_RIGHT;
79
80                  case Mario.State.Super:
81                      return IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_CROUCHING_LEFT : Shared.As-
sets.Animations.MARIO_SUPER_CROUCHING_RIGHT;
82
83                  case Mario.State.Fire:
84                      return IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_CROUCHING_LEFT : Shared.Assets.Ani-
mations.MARIO_FIRE_CROUCHING_RIGHT;
85
86                  /*case Mario.State.Invincible:
87                      return IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_LEFT : Shared.Assets.Ani-
mations.MARIO_INVINCIBLE_IDLE_RIGHT;*/
88
89                  default:
90                      throw new Exception("JAK");
91              }
92          }
93      }
94
95      private string _superPowerUp
96      {
97          get { return IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_POWERUP_LEFT : Shared.Assets.Ani-
mations.MARIO_SUPER_POWERUP_RIGHT; }
98      }
99
100     private string _firePowerUp
101     {
102         get { return IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_POWERUP_LEFT : Shared.Assets.Animati-
ons.MARIO_FIRE_POWERUP_RIGHT; }
103     }
104
105     private string IDLE
106     {
107         get
108         {
109             return Player.CurrentMovement == Mario.Movement.Crouching ? _crouch : _idle;
110         }
111     }
112
113     private string MOVE
114     {
115         get
116         {
117             switch (Player.CurrentState)
118             {
119                 case Mario.State.Small:
120                     return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_MOVE_RIGHT : Shared.Assets.Animati-
ons.MARIO_MOVE_LEFT;
121
122                 case Mario.State.Super:
123                     return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_SUPER_MOVE_RIGHT : Shared.Assets.Ani-
mations.MARIO_SUPER_MOVE_LEFT;
124
125                 case Mario.State.Fire:
126                     return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT : Shared.Assets.Anima-
tions.MARIO_FIRE_MOVE_LEFT;
127
128                 /*case Mario.State.Invincible:
```

```
129                     return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_RIGHT : Shared.As-
    sets.Animations.MARIO_INVINCIBLE_MOVE_LEFT;*/
130
131                 default:
132                     throw new Exception("JAK");
133             }
134         }
135     }
136
137     private string JUMP
138     {
139         get
140         {
141             switch (Player.CurrentState)
142             {
143                 case Mario.State.Small:
144                     return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_JUMP_RIGHT : Shared.As-
    sets.Animations.MARIO_JUMP_LEFT)
145                                           : (IsFacingLeft ? Shared.Assets.Animations.MARIO_JUMP_LEFT : Shared.Assets.Animati-
    ons.MARIO_JUMP_RIGHT);
146
147                 case Mario.State.Super:
148                     return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_SUPER_JUMP_RIGHT :
    Shared.Assets.Animations.MARIO_SUPER_JUMP_LEFT)
149                                           : (IsFacingLeft ? Shared.Assets.Animations.MARIO_SUPER_JUMP_LEFT : Shared.As-
    sets.Animations.MARIO_SUPER_JUMP_RIGHT);
150
151                 case Mario.State.Fire:
152                     return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_FIRE_JUMP_RIGHT : Sha-
    red.Assets.Animations.MARIO_FIRE_JUMP_LEFT)
153                                           : (IsFacingLeft ? Shared.Assets.Animations.MARIO_FIRE_JUMP_LEFT : Shared.As-
    sets.Animations.MARIO_FIRE_JUMP_RIGHT);
154
155                 /*case Mario.State.Invincible:
156                     return horiSpeed != 0 ? (horiSpeed > 0 ? Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_RIGHT
    : Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_LEFT)
157                                           : (IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_JUMP_LEFT : Sha-
    red.Assets.Animations.MARIO_INVINCIBLE_JUMP_RIGHT);*/
158
159                 default:
160                     throw new Exception("JAK");
161             }
162         }
163     }
164
165     private string POWERUP
166     {
167         get
168         {
169             switch (LastState)
170             {
171                 case Mario.State.Small:
172                     return _superPowerUp;
173
174                 case Mario.State.Super:
175                     return LastState < Player.CurrentState ? _firePowerUp : _superPowerUp;
176
177                 case Mario.State.Fire:
178                     return LastState < Player.CurrentState ? "" : _firePowerUp;
179
180                 default:
181                     throw new Exception("JAK");
182             }
183         }
184     }
185
186     public CharacterController(GameObject Parent)
187         : base(Parent)
188     {
189         this.Name = nameof(CharacterController);
190         this.Parent.InitNewComponent<Collider>();
191     }
192
193     protected override void OnColliderEnter(Collider e)
194     { }
195
196     protected override void Start()
```

```
197         {
198             Player = GameObject.Find<Mario>("Player");
199             PlayerAnimator = Component.Find<Animator>("Player_Animator");
200             //SoundOutput = Component.Find<SoundSource>("Player_SoundSource");
201 
202             LastState = Player.CurrentState;
203 
204             Player.Animator.Play(Shared.Assets.Animations.MARIO_IDLE_RIGHT);
205         }
206 
207         protected override void Update()
208         {
209             if (!Enabled)
210                 return;
211 
212             if (LastState != Player.CurrentState && Player.CurrentState != Mario.State.Dead)
213             {
214                 if (!ChangingState)
215                 {
216                     PlayerAnimator.Play(POWERUP);
217                     Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_POWER_UP_SOUND);
218                     bool FromSmalltoLarge = Player.CurrentState > Mario.State.Small && LastState == Mario.State.Small;
219                     bool FromLargeToSmall = Player.CurrentState == Mario.State.Small && LastState == Mario.State.Super;
220                     float YtoAdd = FromSmalltoLarge ? -16 : (FromLargeToSmall ? 0 : 16);
221                     Player.Transform.Position += new Vector3(0, YtoAdd, 0);
222                     ChangingState = true;
223 
224                     Player.LeftTrigger.Collider.Enabled = false;
225                     Player.RightTrigger.Collider.Enabled = false;
226                     Player.TopTrigger.Collider.Enabled = false;
227                     Player.BottomTrigger.Collider.Enabled = false;
228 
229                     Player.Collider.Enabled = false;
230 
231                     return;
232                 }
233 
234                 if (PlayerAnimator.NumberOfPlays > 5)
235                 {
236                     LastState = Player.CurrentState;
237 
238                     Player.LeftTrigger.Collider.Enabled = true;
239                     Player.RightTrigger.Collider.Enabled = true;
240                     Player.TopTrigger.Collider.Enabled = true;
241                     Player.BottomTrigger.Collider.Enabled = true;
242 
243                     Player.Collider.Enabled = true;
244 
245                     ChangingState = false;
246                 }
247                 else
248                     return;
249             }
250             else if (Player.CurrentState == Mario.State.Dead)
251             {
252                 DeadAnimation();
253             }
254             else if (Player.KilledEnemy)
255             {
256                 Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_STOMP_SOUND);
257                 Player.KilledEnemy = false;
258                 EnemyKilledAnim = true;
259                 Jumped = true;
260                 IsFalling = false;
261                 vertSpeed = -FloatSpeed;
262             }
263             else if (Player.ChangeState)
264             {
265                 if (FirstTimePipeEnter)
266                 {
267                     Shared.Mechanics.FXSoundSource.StopSound(Shared.Assets.Sounds.OVERWORLD_THEME_SOUND);
268                     Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_PIPE_ENTER_SOUND);
269                     Player.Collider.Enabled = false;
270                     PipeEnterStartPosition = Player.PipeEnteredInDirection == Direction.Down ? Player.Transform.Position.Y :
    Player.Transform.Position.X;
271                     horiSpeed = 0;
272                     vertSpeed = 0;
```

```
273                 FirstTimePipeEnter = false;
274             }
275
276             if (Player.PipeEnteredInDirection == Direction.Right)
277             {
278                 if (Player.Transform.Position.X < PipeEnterStartPosition + 16)
279                 {
280                     horiSpeed = PipeEnterSpeed;
281                 }
282                 else
283                 {
284                     Player.WorldManager.CurrentlyEnteredPipeScript = Player.EnteredPipe;
285                 }
286             }
287             else if (Player.PipeEnteredInDirection == Direction.Down)
288             {
289                 if (Player.Transform.Position.Y < PipeEnterStartPosition + 16)
290                 {
291                     vertSpeed = PipeEnterSpeed;
292                 }
293                 else
294                 {
295                     Player.WorldManager.CurrentlyEnteredPipeScript = Player.EnteredPipe;
296                 }
297             }
298         }
299         else if (Player.CurrentState > Mario.State.Dead)
300         {
301             Movement();
302         }
303
304         Player.Transform.Position = Player.Transform.Position.Add(horiSpeed * Engine.DeltaTime, vertSpeed * En-
    gine.DeltaTime, 0);
305
306         AnimationControl();
307     }
308
309     private void DeadAnimation()
310     {
311         horiSpeed = 0;
312
313         if (FirstTimeDeadAnimPlay)
314         {
315             Player.Collider.Enabled = false;
316             Player.BottomTrigger.Collider.Enabled = false;
317             Player.LeftTrigger.Collider.Enabled = false;
318             Player.RightTrigger.Collider.Enabled = false;
319             Player.TopTrigger.Collider.Enabled = false;
320
321             vertSpeed = -FloatSpeed;
322
323             FirstTimeDeadAnimPlay = false;
324
325             Shared.Mechanics.FXSoundSource.StopSound(Shared.Assets.Sounds.OVERWORLD_THEME_SOUND);
326             Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_MARIO_DIE_SOUND);
327         }
328         else
329         {
330             vertSpeed += Engine.DeltaTime * DeathAnimSpeed * Acceleration;
331
332             WorldReloadNow += new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime * 1000));
333
334             if (WorldReloadNow > WorldReload)
335             {
336                 Shared.Mechanics.Lives--;
337
338                 if (Shared.Mechanics.Lives == 0)
339                     Engine.ChangeScene(nameof(GameOver), true);
340                 else
341                     Engine.ChangeScene(nameof(WorldScreen), true);
342             }
343         }
344     }
345
346     private void Movement()
347     {
348         if (Player.Collider.Collision(Direction.Down))
```

```
349             {
350                 IsFalling = false;
351                 Jumped = false;
352 
353                 vertSpeed = 0;
354             }
355 
356             if (Engine.Input.IsKeyDown(ConsoleKey.A) || horiSpeed < 0)
357             {
358                 Left();
359             }
360 
361             if (Engine.Input.IsKeyDown(ConsoleKey.W) || Jumped)
362             {
363                 Jump();
364             }
365 
366             if (Engine.Input.IsKeyDown(ConsoleKey.D) || horiSpeed > 0)
367             {
368                 Right();
369             }
370 
371             if (Engine.Input.IsKeyDown(ConsoleKey.S))
372             {
373                 if (vertSpeed == 0)
374                 {
375                     horiSpeed = 0;
376                     Player.CurrentMovement = Mario.Movement.Crouching;
377                 }
378             }
379             else
380             {
381                 Player.CurrentMovement = Mario.Movement.Standing;
382             }
383 
384             if (!Player.Collider.Collision(Direction.Down))
385             {
386                 Fall();
387             }
388         }
389 
390         private void Jump()
391         {
392             if (Engine.Input.IsKeyDown(ConsoleKey.W))
393             {
394                 Player.CurrentMovement = Mario.Movement.Standing;
395 
396                 if (CanJump)
397                 {
398                     if (EnemyKilledAnim)
399                     {
400                         vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed * 2;
401 
402                         if (vertSpeed <= 0)
403                         {
404                             IsFalling = true;
405                             EnemyKilledAnim = false;
406                         }
407                     }
408                     else if (!IsFalling)
409                     {
410                         if (vertSpeed == 0 && !Jumped)
411                         {
412                             Shared.Mechanics.FXSoundSource.PlaySound(Shared.Assets.Sounds.FX_MARIO_JUMP_SOUND);
413                             vertSpeed = -FloatSpeed * 1.5f;
414                             Jumped = true;
415                         }
416                         else if (!Player.Collider.Collision(Direction.Up) && vertSpeed < 0)
417                         {
418                             vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
419                         }
420                         else
421                         {
422                             vertSpeed = 0;
423                             IsFalling = true;
424                         }
425                     }
```

```
426             }
427         }
428         else if (Jumped)
429         {
430             if (EnemyKilledAnim)
431             {
432                 vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed * 4;
433 
434                 if (vertSpeed <= 0)
435                 {
436                     IsFalling = true;
437                     EnemyKilledAnim = false;
438                 }
439             }
440             else if (!IsFalling)
441             {
442                 vertSpeed = -vertSpeed;
443                 IsFalling = true;
444                 EnemyKilledAnim = false;
445             }
446         }
447     }
448 
449     private void Left()
450     {
451         if (Engine.Input.IsKeyDown(ConsoleKey.A))
452         {
453             Player.CurrentMovement = Mario.Movement.Standing;
454 
455             IsFacingLeft = true;
456             if (!Player.Collider.Collision(Direction.Left) && horiSpeed > -MovementSpeed)
457             {
458                 horiSpeed -= Engine.DeltaTime * Acceleration * MovementSpeed;
459             }
460             else if (Player.Collider.Collision(Direction.Left))
461             {
462                 horiSpeed = 0;
463             }
464             else
465             {
466                 horiSpeed = -MovementSpeed;
467             }
468         }
469         else if (horiSpeed < 0)
470         {
471             IsFacingLeft = true;
472             horiSpeed += Engine.DeltaTime * Acceleration * MovementSpeed * 4;
473 
474             if (horiSpeed >= 0 || Player.Collider.Collision(Direction.Left))
475             {
476                 horiSpeed = 0;
477             }
478         }
479     }
480 
481     private void Right()
482     {
483         if (Engine.Input.IsKeyDown(ConsoleKey.D))
484         {
485             Player.CurrentMovement = Mario.Movement.Standing;
486 
487             IsFacingLeft = false;
488             if (!Player.Collider.Collision(Direction.Right) && horiSpeed < MovementSpeed)
489             {
490                 horiSpeed += Engine.DeltaTime * Acceleration * MovementSpeed;
491             }
492             else if (Player.Collider.Collision(Direction.Right))
493             {
494                 horiSpeed = 0;
495             }
496             else
497             {
498                 horiSpeed = MovementSpeed;
499             }
500         }
501         else if (horiSpeed > 0)
502         {
```

```
503                 IsFacingLeft = false;
504                 horiSpeed -= Engine.DeltaTime * Acceleration * MovementSpeed * 2;
505
506                 if (horiSpeed <= 0 || Player.Collider.Collision(Direction.Right))
507                 {
508                     horiSpeed = 0;
509                 }
510             }
511         }
512
513         private void Fall()
514         {
515             if (!IsFalling && !Jumped)
516             {
517                 vertSpeed = 0;
518                 Jumped = true;
519                 IsFalling = true;
520             }
521             else if (IsFalling)
522             {
523                 if (vertSpeed < FloatSpeed)
524                 {
525                     vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
526
527                     if (vertSpeed > FloatSpeed)
528                         vertSpeed = FloatSpeed;
529                 }
530                 else
531                 {
532                     vertSpeed = FloatSpeed;
533                 }
534             }
535         }
536
537         private void AnimationControl()
538         {
539             if (Player.CurrentState > Mario.State.Dead)
540             {
541                 if (Jumped)
542                 {
543                     PlayerAnimator.Play(JUMP);
544                 }
545                 else
546                 {
547                     if (horiSpeed != 0)
548                         PlayerAnimator.Play(MOVE);
549                     else
550                         PlayerAnimator.Play(IDLE);
551                 }
552             }
553             else
554             {
555                 PlayerAnimator.Play(Shared.Assets.Animations.MARIO_DEAD);
556             }
557         }
558     }
559 }
```

2.2.2.29 Assets/Scripts/DeathZoneScript.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3   using MarIO.Assets.Models;
4   using System.Diagnostics;
5
6   namespace MarIO.Assets.Scripts
7   {
8       public class DeathZoneScript : Script
9       {
10          public DeathZoneScript(GameObject Parent) : base(Parent)
11          { }
12
13          protected override void OnColliderEnter(Collider e)
14          {
15              Debug.WriteLine($"{e.Parent}");
16
17              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;
```

```
23          private const float DeadTime = 3f;
24
25          private Enemy Target;
26
27          public GoombaController(GameObject Parent) : base(Parent)
28          {
29              Target = (Enemy)Parent;
30          }
31
32          protected override void OnColliderEnter(Collider e)
33          { }
34
35          protected override void Start()
36          {
37              CurrentSpeed = -Speed;
38          }
39
40          protected override void Update()
41          {
42              if (!Target.IsDestroyed)
43              {
44                  Movement();
45              }
46              else
47              {
48                  DeadAnimation();
49              }
50          }
51
52          private void Movement()
53          {
54              if (Target.Collider.Collision(Direction.Left))
55              {
56                  CurrentSpeed = Speed;
57              }
58
59              if (Target.Collider.Collision(Direction.Right))
60              {
61                  CurrentSpeed = -Speed;
62              }
63
64              if (!Target.Collider.Collision(Direction.Down))
65              {
66                  if (!IsFalling)
67                  {
68                      vertSpeed = 0;
69                      IsFalling = true;
70                  }
71                  else
72                  {
73                      if (vertSpeed < FloatSpeed)
74                      {
75                          vertSpeed += Engine.DeltaTime * Acceleration;
76                      }
77                      else
78                      {
79                          vertSpeed = FloatSpeed;
80                      }
81                  }
82              }
83              else if (IsFalling)
84              {
85                  vertSpeed = 0;
86                  IsFalling = false;
87              }
88
89              Target.Transform.Position += new Vector3(CurrentSpeed * Engine.DeltaTime, vertSpeed * Engine.DeltaTime, 0);
90          }
91
92          private void DeadAnimation()
93          {
94              if (firstTimeDeadAnimation)
95              {
96                  Shared.Mechanics.GameScore += Shared.Mechanics.GOOMBA_POINTS;
97                  TextBlock FloatingText = new TextBlock()
98                  {
99                      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

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using DKEngine.Core.UI;
5
6   namespace MarIO.Assets.Scripts
7   {
8       public class FloatingTextAnimatorScript : Script
9       {
10          private float AnimationHeight = 30;
11          private float AnimationSpeed = 20;
12
13          public FloatingTextAnimatorScript(GameObject Parent)
14              : base(Parent)
15          { }
16
17          protected override void OnColliderEnter(Collider e)
18          { }
19
20          protected override void Start()
21          { }
22
23          protected override void Update()
24          {
25              if (Shared.AnimatedWorldReferences.FloatingTexts.Count > 0)
26              {
27                  for (int i = 0; i < Shared.AnimatedWorldReferences.FloatingTexts.Count; i++)
28                  {
29                      float StartTextBlockY = Shared.AnimatedWorldReferences.FloatingTextStartPosition[i];
30                      TextBlock CurrentTextBlock = Shared.AnimatedWorldReferences.FloatingTexts[i];
31
32                      if (CurrentTextBlock.Transform.Position.Y > StartTextBlockY - AnimationHeight)
33                      {
34                          CurrentTextBlock.Transform.Position -= new Vector3(0, Engine.DeltaTime * AnimationSpeed, 0);
35
36                          if (CurrentTextBlock.Transform.Position.Y < StartTextBlockY - AnimationHeight)
37                          {
38                              Shared.AnimatedWorldReferences.FloatingTextStartPosition.RemoveAt(i);
39                              Shared.AnimatedWorldReferences.FloatingTexts.RemoveAt(i);
40
41                              CurrentTextBlock.Destroy();
42
43                              i--;
44                          }
45                      }
46                  }
47              }
48          }
49      }
50  }
```

2.2.2.34 Assets/Scripts/GUIUpdateScript.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using DKEngine.Core.UI;
5
6   namespace MarIO.Assets.Scripts
7   {
8       public class GUIUpdateScript : Script
9       {
10          private TextBlock Time;
11          private TextBlock Coins;
12          private TextBlock World;
13          private TextBlock Lives;
14          private TextBlock Score;
15
16          public GUIUpdateScript(GameObject Parent) : base(Parent)
17          { }
18
19          protected override void OnColliderEnter(Collider e)
20          { }
21
22          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)
{ }
```

```
13  protected override void OnColliderEnter(Collider e)
14  {
15  if (e.Parent is Enemy)
16  {
17  Enemy tmp = e.Parent as Enemy;
18  Debug.WriteLine(string.Format("Zabil jsi {0}", tmp.Name));
19  tmp.IsDestroyed = true;
20  Mario.KilledEnemy = true;
21  }
22  else if (e.Parent is PowerUp)
23  {
24  ((PowerUp)e.Parent).OnPickedUp?.Invoke();
25  }
26  }

27  protected override void Start()
28  {
29  Mario = GameObject.Find<Mario>("Player");
30  }

31  protected override void Update()
32  { }
33  }

34  internal class TopMarioChecker : Script
35  {
36  private Mario Mario;

37  public TopMarioChecker(GameObject Parent) : base(Parent)
38  { }

39  protected override void OnColliderEnter(Collider e)
40  {
41  if (e.Parent is Enemy)
42  {
43  Debug.WriteLine(string.Format("Zabilo Tě {0}", e.Parent.TypeName));
44  Mario.CurrentState--;
45  }
46  else if (e.Parent is Block)
47  {
48  Block tmp = e.Parent as Block;

49  if (tmp.State == Block.CollisionState.Stay)
50  {
        a.  tmp.AnimateBlockCollision();
51  }

52  tmp.GetContent();
53  }
54  else if (e.Parent is PowerUp)
55  {
56  ((PowerUp)e.Parent).OnPickedUp?.Invoke();
57  }
58  }

59  protected override void Start()
60  {
61  Mario = GameObject.Find<Mario>("Player");
62  }

63  protected override void Update()
64  { }
65  }

66  internal class LeftMarioChecker : Script
67  {
68  private Mario Mario;

69  public LeftMarioChecker(GameObject Parent) : base(Parent)
70  { }

71  protected override void OnColliderEnter(Collider e)
72  {
73  if (e.Parent is Enemy)
74  {
75  Debug.WriteLine(string.Format("Zabilo Tě {0}", e.Parent.TypeName));
76  Mario.CurrentState--;
```

```
77  //Mario?.Destroy();
78  }
79  else if (e.Parent is PowerUp)
80  {
81  ((PowerUp)e.Parent).OnPickedUp?.Invoke();
82  }
83  }

84  protected override void Start()
85  {
86  Mario = GameObject.Find<Mario>("Player");
87  }

88  protected override void Update()
89  { }
90  }

91  internal class RightMarioChecker : Script
92  {
93  private Mario Mario;

94  public RightMarioChecker(GameObject Parent) : base(Parent)
95  { }

96  protected override void OnColliderEnter(Collider e)
97  {
98  if (e.Parent is Enemy)
99  {
100 Debug.WriteLine(string.Format("Zabilo Tě {0}", e.Parent.TypeName));
101 Mario.CurrentState--;
102 //Mario?.Destroy();
103 }
104 else if (e.Parent is PowerUp)
105 {
106 ((PowerUp)e.Parent).OnPickedUp?.Invoke();
107 }
108 }

109 protected override void Start()
110 {
111 Mario = GameObject.Find<Mario>("Player");
112 }

113 protected override void Update()
114 { }
115 }
116 }
```

2.2.2.37 Assets/Scripts/MusicScript.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3   using System;
4   using System.Diagnostics;
5
6   namespace MarIO.Assets.Scripts
7   {
8       public class MusicScript : Script
9       {
10          private Sound Music;
11          private TimeSpan MusicLenght;
12
13          private Stopwatch Timer;
14
15          public MusicScript(GameObject Parent) : base(Parent)
16          { }
17
18          protected override void OnColliderEnter(Collider e)
19          { }
20
21          protected override void Start()
22          {
23              Music = Shared.Assets.Sounds.OVERWORLD_THEME_SOUND;
24              MusicLenght = Music.FileReader.TotalTime;
25              Shared.Mechanics.FXSoundSource.PlaySound(Music);
26              Timer = Stopwatch.StartNew();
27          }
```

```
28
29          protected override void Update()
30          {
31              if (Timer.Elapsed > MusicLenght)
32              {
33                  Shared.Mechanics.FXSoundSource.PlaySound(Music);
34
35                  Timer.Restart();
36              }
37          }
38      }
39  }
```

2.2.2.38 Assets/Scripts/PipePort.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3   using MarIO.Assets.Models;
4
5   namespace MarIO.Assets.Scripts
6   {
7       public class PipePort : Script
8       {
9           private Mario Player;
10          public Block Pipe;
11
12          public PipePort(GameObject Parent) : base(Parent)
13          { }
14
15          protected override void OnColliderEnter(Collider e)
16          {
17              if (Pipe.SpecialAction != null)
18              {
19                  if (e.Parent == Player)
20                  {
21                      switch (Pipe.PipeEnterDirection)
22                      {
23                          case Transform.Direction.Up:
24                              break;
25
26                          case Transform.Direction.Left:
27                              break;
28
29                          case Transform.Direction.Down:
30                              if (Player.CurrentMovement == Mario.Movement.Crouching)
31                              {
32                                  Player.PipeEnter(Pipe);
33                              }
34                              break;
35
36                          case Transform.Direction.Right:
37                              if (Player.CurrentMovement == Mario.Movement.Standing)
38                              {
39                                  Player.PipeEnter(Pipe);
40                              }
41                              break;
42
43                          default:
44                              break;
45                      }
46                  }
47              }
48          }
49
50          protected override void Start()
51          {
52              Player = GameObject.Find<Mario>("Player");
53              Pipe = (Block)Parent;
54          }
55
56          protected override void Update()
57          { }
58      }
59  }
```

2.2.2.39 Assets/Scripts/PowerUpScript.cs

```
1  using DKEngine;
2  using DKEngine.Core;
3  using DKEngine.Core.Components;
4  using MarIO.Assets.Models;
5  using MarIO.Assets.Models.Miscellaneous;
6  using System;
7  using static DKEngine.Core.Components.Transform;
8
9  namespace MarIO.Assets.Scripts
10 {
11     internal class PowerUpScript : Script
12     {
13         private PowerUp Target;
14         private bool CreatedForFirstTime = true;
15         private bool CreatedAnimation = true;
16         private float CreatedStartY;
17         private const float CreationAnimationSpeed = 20f;
18
19         private const float Speed = 80f;
20         private const float FloatSpeed = 250f;
21         private const float Acceleration = 3.5f;
22
23         private float CurrentSpeed = 0;
24         private float vertSpeed = 0;
25         private bool IsFalling = false;
26         private bool Jumped = false;
27
28         public PowerUpScript(GameObject Parent) : base(Parent)
29         {
30             Target = Parent as PowerUp;
31         }
32
33         protected override void OnColliderEnter(Collider e)
34         { }
35
36         protected override void Start()
37         {
38             CurrentSpeed = Speed;
39
40             Target.PlayerReference = GameObject.Find<Mario>("Player");
41         }
42
43         protected override void Update()
44         {
45             if (CreatedForFirstTime)
46             {
47                 Target.Collider.Enabled = false;
48                 CreatedStartY = Target.Transform.Position.Y;
49                 CreatedForFirstTime = false;
50                 return;
51             }
52             else if (CreatedAnimation)
53             {
54                 if (CreatedStartY < Target.Transform.Position.Y + 16)
55                 {
56                     Target.Transform.Position -= new Vector3(0, Engine.DeltaTime * CreationAnimationSpeed, 0);
57                 }
58                 else
59                 {
60                     Target.Transform.Position = new Vector3(Target.Transform.Position.X, CreatedStartY - 16, Target.Trans-
   form.Position.Z);
61                     Target.Collider.Enabled = true;
62                     CreatedAnimation = false;
63                 }
64
65                 return;
66             }
67             else
68             {
69                 switch (Target.Type)
70                 {
71                     case PowerUp.PowerUpType.Mushroom:
72                         MushroomMovement();
73                         break;
74
```

```
75                  case PowerUp.PowerUpType.Flower:
76                      CurrentSpeed = 0;
77                      break;
78
79                  case PowerUp.PowerUpType.Star:
80                      StarMovement();
81                      break;
82
83                  default:
84                      throw new Exception("JAK");
85              }
86          }
87      }
88
89      private void MushroomMovement()
90      {
91          if (Target.Collider.Collision(Direction.Left))
92          {
93              CurrentSpeed = Speed;
94          }
95
96          if (Target.Collider.Collision(Direction.Right))
97          {
98              CurrentSpeed = -Speed;
99          }
100
101         if (!Target.Collider.Collision(Direction.Down))
102         {
103             if (!IsFalling)
104             {
105                 vertSpeed = 0;
106                 IsFalling = true;
107             }
108             else
109             {
110                 if (vertSpeed < FloatSpeed)
111                 {
112                     vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
113                 }
114                 else
115                 {
116                     vertSpeed = FloatSpeed;
117                 }
118             }
119         }
120         else if (IsFalling)
121         {
122             vertSpeed = 0;
123             IsFalling = false;
124         }
125
126         Target.Transform.Position += new Vector3(CurrentSpeed * Engine.DeltaTime, vertSpeed * Engine.DeltaTime, 0);
127     }
128
129     private void StarMovement()
130     {
131         if (Target.Collider.Collision(Direction.Left))
132         {
133             CurrentSpeed = Speed;
134         }
135
136         if (Target.Collider.Collision(Direction.Right))
137         {
138             CurrentSpeed = -Speed;
139         }
140
141         if (!Target.Collider.Collision(Direction.Down))
142         {
143             if (vertSpeed == 0 && !Jumped)
144             {
145                 vertSpeed = -FloatSpeed * 1.5f;
146                 Jumped = true;
147             }
148             else if (!Target.Collider.Collision(Direction.Up) && vertSpeed < 0)
149             {
150                 vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
151             }
```

```
152                 else
153                 {
154                     vertSpeed = 0;
155                     IsFalling = true;
156                 }
157             }
158             else
159             {
160                 if (!IsFalling && !Jumped)
161                 {
162                     vertSpeed = 0;
163                     Jumped = true;
164                     IsFalling = true;
165                 }
166                 else if (IsFalling)
167                 {
168                     if (vertSpeed < FloatSpeed)
169                     {
170                         vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
171                     }
172                     else
173                     {
174                         vertSpeed = FloatSpeed;
175                     }
176                 }
177             }
178         }
179     }
180 }
```

2.2.2.40 Assets/Scripts/SpecialBlocksUpdateScript.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3
4   namespace MarIO.Assets.Scripts
5   {
6       public class SpecialBlocksUpdateScript : Script
7       {
8           public SpecialBlocksUpdateScript(GameObject Parent)
9               : base(Parent)
10          { }
11
12          protected override void OnColliderEnter(Collider e)
13          { }
14
15          protected override void Start()
16          {
17          }
18
19          protected override void Update()
20          {
21              while (Shared.AnimatedWorldReferences.SpecialActions.Count > 0)
22              {
23                  Shared.AnimatedWorldReferences.SpecialActions.Pop().SpecialAction();
24              }
25          }
26      }
27  }
```

2.2.2.41 Assets/Scripts/WorldChangeManagerScript.cs

```
1   using DKEngine.Core;
2   using DKEngine.Core.Components;
3   using MarIO.Assets.Models;
4
5   namespace MarIO.Assets.Scripts
6   {
7       public class WorldChangeManagerScript : Script
8       {
9           public Block CurrentlyEnteredPipeScript;
10
11          public WorldChangeManagerScript(GameObject Parent) : base(Parent)
12          {
13              Name = "worldManager";
14          }
15
```

```
16          protected override void OnColliderEnter(Collider e)
17          { }
18
19          protected override void Start()
20          { }
21
22          protected override void Update()
23          {
24              CurrentlyEnteredPipeScript?.SpecialAction();
25              CurrentlyEnteredPipeScript = null;
26          }
27      }
28  }
```

2.2.2.42 Assets/Scripts/WorldEnd.cs

```
1   using DKEngine;
2   using DKEngine.Core;
3   using DKEngine.Core.Components;
4   using MarIO.Assets.Models;
5   using MarIO.Assets.Scenes;
6   using System;
7   using static DKEngine.Core.Components.Transform;
8
9   namespace MarIO.Assets.Scripts
10  {
11      internal class WorldEnd : Script
12      {
13          private Mario Player;
14          private CharacterController PlayerController;
15          private Animator PlayerAnimator;
16
17          private float horiSpeed = 0;
18          private float vertSpeed = 0;
19          private float Distance = 180;
20          private float startX;
21
22          private const float MovementSpeed = 80f;
23          private const float FloatSpeed = 300f;
24
25          private const float Acceleration = 3.5f;
26
27          private readonly TimeSpan _delay = new TimeSpan(0, 0, 3);
28          private TimeSpan Delay = new TimeSpan();
29
30          private string MOVE
31          {
32              get
33              {
34                  switch (Player.CurrentState)
35                  {
36                      case Mario.State.Small:
37                          return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_MOVE_RIGHT : Shared.Assets.Animations.MARIO_MOVE_LEFT;
38
39                      case Mario.State.Super:
40                          return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_SUPER_MOVE_RIGHT : Shared.Assets.Animations.MARIO_SUPER_MOVE_LEFT;
41
42                      case Mario.State.Fire:
43                          return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_FIRE_MOVE_RIGHT : Shared.Assets.Animations.MARIO_FIRE_MOVE_LEFT;
44
45                      /*case Mario.State.Invincible:
46                          return horiSpeed >= 0 ? Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_RIGHT : Shared.Assets.Animations.MARIO_INVINCIBLE_MOVE_LEFT;*/
47
48                      default:
49                          throw new Exception("JAK");
50                  }
51              }
52          }
53
54          private string IDLE
55          {
56              get
57              {
```

```
58                  switch (Player.CurrentState)
59                  {
60                      case Mario.State.Dead:
61                      case Mario.State.Small:
62                          return horiSpeed < 0 ? Shared.Assets.Animations.MARIO_IDLE_LEFT : Shared.Assets.Animations.MA-
RIO_IDLE_RIGHT;
63
64                      case Mario.State.Super:
65                          return horiSpeed < 0 ? Shared.Assets.Animations.MARIO_SUPER_IDLE_LEFT : Shared.Assets.Animati-
ons.MARIO_SUPER_IDLE_RIGHT;
66
67                      case Mario.State.Fire:
68                          return horiSpeed < 0 ? Shared.Assets.Animations.MARIO_FIRE_IDLE_LEFT : Shared.Assets.Animati-
ons.MARIO_FIRE_IDLE_RIGHT;
69
70                      /*case Mario.State.Invincible:
71                          return IsFacingLeft ? Shared.Assets.Animations.MARIO_INVINCIBLE_IDLE_LEFT : Shared.Assets.Ani-
mations.MARIO_INVINCIBLE_IDLE_RIGHT;*/
72
73                      default:
74                          throw new Exception("JAK");
75                  }
76              }
77          }
78
79          public WorldEnd(GameObject Parent) : base(Parent)
80          {
81              startX = Parent.Transform.Position.X;
82          }
83
84          protected override void OnColliderEnter(Collider e)
85          {
86              if (e.Parent is Mario)
87                  PlayerController.Enabled = false;
88          }
89
90          protected override void Start()
91          {
92              Player = GameObject.Find<Mario>("Player");
93              PlayerAnimator = Component.Find<Animator>("Player_Animator");
94              PlayerController = Script.Find<CharacterController>(nameof(CharacterController));
95          }
96
97          protected override void Update()
98          {
99              if (Shared.Mechanics.TimeLeft.TotalSeconds <= 0)
100             {
101                 Player.CurrentState = Mario.State.Dead;
102                 Shared.Mechanics.TimeCounter.Stop();
103             }
104
105             if (!PlayerController.Enabled)
106             {
107                 PlayerAnimator.Play(MOVE);
108
109                 if (!Player.Collider.Collision(Direction.Down))
110                 {
111                     if (vertSpeed < FloatSpeed)
112                     {
113                         vertSpeed += Engine.DeltaTime * Acceleration * FloatSpeed;
114
115                         if (vertSpeed > FloatSpeed)
116                             vertSpeed = FloatSpeed;
117                     }
118                     else
119                     {
120                         vertSpeed = FloatSpeed;
121                     }
122                 }
123
124                 if (Player.Transform.Position.X > startX + Distance)
125                 {
126                     PlayerAnimator.Play(IDLE);
127                     Delay += new TimeSpan(0, 0, 0, 0, (int)(Engine.DeltaTime * 1000));
128                     if (_delay < Delay)
129                     {
```

```
130                     Shared.Mechanics.FXSoundSource.StopSound(Shared.As-
    sets.Sounds.OVERWORLD_THEME_SOUND);
131                     Engine.ChangeScene(nameof(GameOver), true);
132                 }
133             }
134             else
135             {
136                 this.Player.Transform.Position += new Vector3(MovementSpeed * Engine.DeltaTime, vertSpeed, 0);
137             }
138         }
139     }
140 }
141 }
```

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