Athena

Assembly Info

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Athena")]
[assembly: AssemblyProduct("Athena")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyCopyright("Copyright ©  2014")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("ef85f802-4875-4a87-92d1-e00ec9452bcc")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Levels

```
?xml version="1.0" encoding="utf-8" ?>
<level tileset="pokemon" tilesize="25" rowlength="3">
  <tile>
    <sprite>paving.cells</sprite>
  </tile>
  <tile>
    <sprite>paving.cells</sprite>
  </tile>
  <tile>
    <sprite>black</sprite>
  </tile>
  <tile>
    <sprite>black</sprite>
  </tile>
</level>
```

UI Font

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file contains an xml description of a font, and will be read by the XNA
Framework Content Pipeline. Follow the comments to customize the appearance
of the font in your game, and to change the characters which are available to draw
with.
-->
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">

    <!--
    Modify this string to change the font that will be imported.
    -->
    <FontName>Minecraftia</FontName>

    <!--
    Size is a float value, measured in points. Modify this value to change
    the size of the font.
    -->
    <Size>18</Size>

    <!--
    Spacing is a float value, measured in pixels. Modify this value to change
    the amount of spacing in between characters.
    -->
    <Spacing>3</Spacing>

    <!--
    UseKerning controls the layout of the font. If this value is true, kerning
information
    will be used when placing characters.
    -->
    <UseKerning>false</UseKerning>

    <!--
    Style controls the style of the font. Valid entries are "Regular", "Bold",
"Italic",
    and "Bold, Italic", and are case sensitive.
    -->
    <Style>Regular</Style>

    <!--
    If you uncomment this line, the default character will be substituted if you draw
    or measure text that contains characters which were not included in the font.
    -->
    <!-- <DefaultCharacter>*</DefaultCharacter> -->

    <!--
    CharacterRegions control what letters are available in the font. Every
    character from Start to End will be built and made available for drawing. The
    default range is from 32, (ASCII space), to 126, ('~'), covering the basic Latin
    character set. The characters are ordered according to the Unicode standard.
    See the documentation for more information.
    -->
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

Tileset

```
<?xml version="1.0" encoding="utf-8" ?>
<tileset tilesize="25">
  <sprite>
    <name>black</name>
    <x>0</x>
    <y>0</y>
    <collides>false</collides>
  </sprite>
  <sprite>
    <name>tile.diagonal</name>
    <x>1</x>
    <y>0</y>
    <collides>false</collides>
  </sprite>
  <sprite>
    <name>carpet.horizontal</name>
    <x>2</x>
    <y>0</y>
    <collides>false</collides>
  </sprite>
  <sprite>
    <name>paving.cells</name>
    <x>2</x>
    <y>0</y>
    <collides>false</collides>
  </sprite>
</tileset>
```

Debug

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Athena
{
    public static class Debug
    {
        public static Modes Mode;
        public enum Modes { GAME, UI };
    }
}
```

Game1

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework;
using AthenaEngine.Framework.Gameplay;
using AthenaEngine.Framework.Interfaces;
using AthenaEngine.Framework.Primatives;
using AthenaEngine.Framework.Systems;
using AthenaEngine.Framework.UI;

namespace Athena
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        // TODO: Remove warning suppression once it's used.
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        ResourceManager<Texture2D> textureManager;
        ResourceManager<SpriteFont> fontManager;
        Level test;
        Camera2D Camera;
        Character Player;
        bool FirstRun = true;
        KeyboardState OldState;


        /// <summary>
        /// The actual game class constructor.
        /// </summary>
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting
to run.
        /// This is where it can query for any required services and load any non-
graphic
        /// related content.  Calling base.Initialize will enumerate through any
components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }
        public SpriteFont font;

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);
            textureManager = new ResourceManager<Texture2D>(this);
            fontManager = new ResourceManager<SpriteFont>(this);

            this.Camera = new Camera2D(this);

            textureManager.Add("blank", this.Content.Load<Texture2D>("blank"));
            textureManager.Add("pokemon", this.Content.Load<Texture2D>("pokemon-
tiles"));
            textureManager.Add("Boards", this.Content.Load<Texture2D>("Boards"));
            textureManager.Add("guy", this.Content.Load<Texture2D>("guy"));
            fontManager.Add("SpriteFont1",
this.Content.Load<SpriteFont>("SpriteFonts/UIFont"));
            test = new Level("level1", spriteBatch, textureManager);
            this.Player = new Character(new Vector2(75, 75), new Vector2(25, 25),
spriteBatch, textureManager.Get("guy"), test);
            // Player.SpriteColor = Color.Red;
            Camera.Focus = Player;
            Camera.Initialize();
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            KeyboardState State = Keyboard.GetState();

            if (FirstRun)
            {
                if (State.IsKeyDown(Keys.W))
                {
                    Player.Move(Directions.UP);
                }
                if (State.IsKeyDown(Keys.A))
                {
                    Player.Move(Directions.LEFT);
                }
                if (State.IsKeyDown(Keys.S))
                {
                    Player.Move(Directions.DOWN);
                }
                if (State.IsKeyDown(Keys.D))
                {
                    Player.Move(Directions.RIGHT);
                }

                OldState = State;
                FirstRun = false;
            }
            else if (OldState != State)
            {
                if (State.IsKeyDown(Keys.W))
                {
                    Player.Move(Directions.UP);
                }
                if (State.IsKeyDown(Keys.A))
                {
                    Player.Move(Directions.LEFT);
                }
                if (State.IsKeyDown(Keys.S))
                {
                    Player.Move(Directions.DOWN);
                }
                if (State.IsKeyDown(Keys.D))
                {
                    Player.Move(Directions.RIGHT);
                }

                OldState = State;
            }
            else
            {
                // Do nothing.
            }

            Camera.Update(gameTime);
            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
```

```
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            Debug.Mode = Debug.Modes.GAME;

            GraphicsDevice.Clear(Microsoft.Xna.Framework.Color.CornflowerBlue);

            if (Debug.Mode == Debug.Modes.GAME)
            {
                Camera.Focus = Player;

                spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend,
null, null, null, null, Camera.Transform);

                test.Draw();
                Player.Draw();
                spriteBatch.End();
            }

            else if (Debug.Mode == Debug.Modes.UI)
            {
//                UIButton test = new UIButton(new Vector2(0, 0), new Vector2(50, 50),
spriteBatch, textureManager.Get("blank"));
                UI Test = new UI(spriteBatch, textureManager, fontManager);

                Test.AddButton(new Vector2(5, 5), "HI");
                Test.AddButton(new Vector2(50, 50), "ASK!");
                spriteBatch.Begin();
                // Test.Draw();
                // int x = 6;
                // int y = 105;

                // spriteBatch.Draw(textureManager.Get("pokemon"), new Rectangle(0, 0,
25, 25), new Rectangle((x - 1) * 25, (y - 1 )* 25, 25, 25), Color.White);
                Player.Draw();
                test.Draw();


                spriteBatch.End();


            }

            base.Draw(gameTime);
        }
    }
}
```

Program

```
#region Using Statements
using System;
using System.Collections.Generic;
using System.Linq;
#endregion

namespace Athena
{
#if WINDOWS || LINUX
    /// <summary>
    /// The main class.
    /// </summary>
    public static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            using (var game = new Game1())
                game.Run();
        }
    }
#endif
}
```

Athena Engine

RPG

Inventory

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace AthenaEngine.Framework.Gameplay.RPG
{
       /// <summary>
       /// Inventory.
       /// </summary>
    class Inventory
    {
        private List<ItemInstance> ItemList = new List<ItemInstance>();
        private int ItemCount;
```

```
            /// <summary>
            /// Add the specified item and quantity.
            /// </summary>
            /// <param name='item'>
            /// Item.
            /// </param>
            /// <param name='quantity'>
            /// Quantity.
            /// </param>
        public void Add(Item item, int quantity)
        {
            this.ItemList.Add(new ItemInstance(item, quantity));
        }

    }
}
```

Item

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace AthenaEngine.Framework.Gameplay.RPG
{
    /// <summary>
    /// A gameplay item can be held by a character.
    /// </summary>
    public class Item
    {
        /// <summary>
        /// The name of the item
        /// </summary>
        public string Name {
            get { return this.Name; }
            private set { this.Name = value; }
        }

        /// <summary>
        /// Constructor for the item class
        /// </summary>
        /// <param name="name">Name of the item</param>
        public Item(string name)
        {
            this.Name = name;
        }
    }
}
```

Item Instance

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AthenaEngine.Framework.Gameplay.RPG
{
    /// <summary>
    /// An item instance is a particular instance of an item.
    /// </summary>
    public class ItemInstance : Item
    {
        private int Quantity;
        /// <summary>
        /// Constructor for the ItemInstance class.
        /// </summary>
        /// <param name="item">The actual item of which this is an instance</param>
        /// <param name="quantity">How many of that item are in this particular 
instance?</param>
        public ItemInstance(Item item, int quantity) : base(item.Name)
        {
            this.Quantity = quantity;
        }
    }
}
```

Character

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;


using AthenaEngine.Framework.Primatives;
using AthenaEngine.Framework.Interfaces;

namespace AthenaEngine.Framework.Gameplay
{
    /// <summary>
    /// A character class holds important detail about each character such as their 
items, level, experience, skills, etc.
    /// </summary>
    public class Character : DrawableEntity, IFocusable
    {
        private int Level;
        private int Experience;

        /// <summary>
        /// Constructor for the Character class
        /// </summary>
        /// <param name="position">The coordinates of the character</param>
        /// <param name="size">The size of the character</param>
        /// <param name="spriteBatch">Which spritebatch will draw the
character</param>
        /// <param name="texture">What is the texture for the character</param>
        public Character(Vector2 position, Vector2 size, SpriteBatch spriteBatch,
Texture2D texture, Level level) : base(position, size, new Rectangle(0, 0, 25, 25),
spriteBatch, texture, level)
        {

        }


    }
}
```

Level

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Systems;

namespace AthenaEngine.Framework.Gameplay
{
    /// <summary>
    /// A Level object holds all details required to handle level drawing.
    /// </summary>
    public class Level
    {
        public List<Tile> TileList;

        /// <summary>
        /// Object constructor for the Level class.
        /// </summary>
        /// <param name="levelName">The name of the level to load.</param>
        /// <param name="spriteBatch">The spritebatch to draw the level with.</param>
        /// <param name="resourceManager">The resourceManager handling the games'
textures.</param>
        public Level(string levelName, SpriteBatch spriteBatch,
ResourceManager<Texture2D> resourceManager)
        {
            this.TileList = LevelLoaderXml.Load(levelName, this);

            foreach (Tile tile in TileList)
            {
                tile.MakeDrawable(spriteBatch, resourceManager);
            }
        }

        /// <summary>
        /// Draw the level.
        /// </summary>
        public void Draw()
        {
            foreach (Tile tile in TileList)
            {
                tile.Draw();
            }
        }
    }
}
```

Tile

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using System.Reflection;

using AthenaEngine.Framework.Primatives;
using AthenaEngine.Framework.Interfaces;
using AthenaEngine.Framework.Systems;

namespace AthenaEngine.Framework.Gameplay
{
    /// <summary>
    /// A tile is used to draw levels.
    /// </summary>
    public class Tile : CollidableEntity
    {
        private bool Drawable;
        public bool Collides = false;
        private Texture2D Texture;
        public DrawableEntity Sprite;
        private string TileSet;
        private Rectangle SourceRectangle;
        private MethodInfo Trigger;
        public bool HasTrigger = false;

        /// <summary>
        /// Class constructor for a new Tile object.
        /// </summary>
        /// <param name="x">The X coordinate of the new tile.</param>
        /// <param name="y">The Y coordinate of the new tile.</param>
        public Tile(int x, int y, string TileSet, int xOffset, int yOffset, Level
level) : base(new Vector2(x, y), new Vector2(25, 25), level)
        {
            this.TileSet = TileSet;
            this.SourceRectangle = new Rectangle(xOffset, yOffset, 25, 25);
        }

        /// <summary>
        /// Make a tile drawable by giving it a texture and a spriteBatch.
        /// </summary>
        /// <param name="spriteBatch">The SpriteBatch to draw the tile with.</param>
        /// <param name="texture">The texture to draw the tile with.</param>
        public void MakeDrawable(SpriteBatch spriteBatch, ResourceManager<Texture2D>
resoureManager)
```

```
        {
            this.Drawable = true;

            this.Texture = resoureManager.Get(TileSet);

            this.Sprite = new DrawableEntity(Position, Size, SourceRectangle,
spriteBatch, this.Texture, this.Level);
        }

        /// <summary>
        /// Draw the tile.
        /// </summary>
        public void Draw()
        {
            if (this.Drawable)
            {
                this.Sprite.Draw();
            }

        }

              /// <summary>
              /// Fires the trigger.
              /// </summary>
              /// <param name='args'>
              /// Arguments.
              /// </param>
        public void FireTrigger (object[] args)
        {
            if (this.HasTrigger)
            {
                Trigger.Invoke(null, args);
            }

        }

              /// <summary>
              /// Adds the trigger.
              /// </summary>
              /// <param name='triggerName'>
              /// Trigger name.
              /// </param>
        public void AddTrigger(string triggerName)
        {
            Type type = typeof(Triggers);
            MethodInfo method = type.GetMethod(triggerName);
            this.Trigger = method;
            this.HasTrigger = true;
        }
    }
}
```

I Collidable

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Primatives;

namespace AthenaEngine.Framework.Interfaces
{
       /// <summary>
       /// I collidable.
       /// </summary>
    interface ICollidable<T>
    {
        bool CollidesWith(T type);
    }
}
```

I Drawable

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace AthenaEngine.Framework.Interfaces
{
       /// <summary>
       /// I drawable.
       /// </summary>
    interface IDrawable
    {
              /// <summary>
              /// Draw this instance.
              /// </summary>
        void Draw();
    }
}
```

I Focusable

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;


namespace AthenaEngine.Framework.Interfaces
{
        /// <summary>
        /// I focusable.
        /// </summary>
        public interface IFocusable
        {
                /// <summary>
                /// Gets the position.
                /// </summary>
                /// <value>
                /// The position.
                /// </value>
                Vector2 Position { get; }
        }
}
```

I Moveable

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AthenaEngine.Framework.Interfaces
{
    interface IMoveable
    {
        bool Move(string direction);
        bool CanMove(string direction);

    }
}
```

Bounding Box 2D

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace AthenaEngine.Framework.Primatives
{
    /// <summary>
    /// BoundingBox2D is used for bounding boxes on 2D objects.
    /// </summary>
    public class BoundingBox2D : Entity, IEquatable<BoundingBox2D>
    {
        protected Rectangle Bounds
        {
            get
            {
                return this.Rectangle;
            }
            set
            {
                this.Rectangle = value;
            }
        }
        /// <summary>
        /// Constructor for BoundingBox2D
        /// </summary>
        /// <param name="min">x/y coordinates for the bounding box</param>
        /// <param name="max">width/height for the bounding box</param>
        public BoundingBox2D(Vector2 Position, Vector2 Size)
        {
            base.Position = Position;
            base.Size = Size;
        }

        /// <summary>
        /// Check if the BoundingBox2D is equal to another BoundingBox2D
        /// </summary>
        /// <param name="otherBounds">The other BoundingBox2D to check
against.</param>
        /// <returns></returns>
        public bool Equals (BoundingBox2D otherBounds)
        {
            if (this.Bounds.Equals(otherBounds.Bounds))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        /// <summary>
        /// Check to see if this BoundingBox2D collides with an other BoundingBox.
        /// </summary>
        /// <param name="otherBounds">The other BoundingBox2D to compare with.</param>
        /// <returns></returns>
        public bool CollidesWith(BoundingBox2D other)
        {

            bool Colliding = false;

            if (this.Equals(other))
            {
                Colliding = true;
            }
            if ((this.Bounds.Bottom < other.Bounds.Bottom) && (this.Bounds.Bottom >
other.Bounds.Top))
            {
                Colliding = true;
            }
            if ((this.Bounds.Top > other.Bounds.Top) && (this.Bounds.Top <
other.Bounds.Bottom))
            {
                Colliding = true;
            }
            if ((this.Bounds.Left > other.Bounds.Left) && (this.Bounds.Left <
other.Bounds.Right))
            {
                Colliding = true;
            }
            if ((this.Bounds.Right < other.Bounds.Right) && (this.Bounds.Right >
other.Bounds.Left))
            {
                Colliding = true;
            }

            return Colliding;
        }
    }
}
```

<u>Collidable Entity</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Interfaces;
using AthenaEngine.Framework.Gameplay;

namespace AthenaEngine.Framework.Primatives
{
    /// <summary>
    /// The Entity class is used to store objects that have positions.
    /// </summary>
    public abstract class CollidableEntity : Entity
    {
        protected BoundingBox2D Bounds
        {
            get
            {
                return new BoundingBox2D(Position, Size);
            }
            set
            {
                if (this.Size.X < 0 || this.Size.Y < 0)
                {
                    throw new ArithmeticException("Object size is invalid, less than
0.");
                }
                this.Position = new Vector2(value.X, value.Y);
                this.Size = new Vector2(value.Width, value.Height);
            }
        }

        public Level Level;

        /// <summary>
        /// Constructor for the CollidableEntity class.
        /// </summary>
        /// <param name="position">x/y coordinates of the entity.</param>
        /// <param name="size">width/height of the entity.</param>
        public CollidableEntity(Vector2 position, Vector2 size, Level level)
        {
            this.Position = position;
            this.Size = size;
            this.Bounds = new BoundingBox2D(position, size);
            this.Level = level;
        }

        /// <summary>
        /// Check if the CollidableEntity collides with another CollidableEntity.
        /// </summary>
        /// <param name="entity">The entity to check against</param>
        /// <returns>Returns true if it does collide, otherwise false.</returns>
        public bool CollidesWith (CollidableEntity entity)
        {
            if (this.Bounds.CollidesWith(entity.Bounds))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

              /// <summary>
              /// Determines whether this instance can move the specified direction.
              /// </summary>
              /// <returns>
              /// <c>true</c> if this instance can move the specified direction;
otherwise, <c>false</c>.
              /// </returns>
              /// <param name='direction'>
              /// If set to <c>true</c> direction.
              /// </param>
        public bool CanMove(int direction)
        {
            BoundingBox2D NewPosition = new BoundingBox2D(new Vector2(this.Bounds.X,
this.Bounds.Y), new Vector2(this.Bounds.Width, this.Bounds.Height));

            switch (direction)
            {
                case Directions.UP:
                    NewPosition.Y -= 25;
                    break;
                case Directions.DOWN:
                    NewPosition.Y += 25;
                    break;
                case Directions.LEFT:
                    NewPosition.X -= 25;
                    break;
                case Directions.RIGHT:
                    NewPosition.X += 25;
                    break;
            }
            foreach (Tile t in Level.TileList)
            {
                if (t.Collides)
                {
                    if (NewPosition.CollidesWith(t.Bounds))
                    {
                        return false;
                    }
                    else
                    {
                        // There wasn't.
                    }
                }
            }

            return true;

        }

              /// <summary>
              /// Move the specified direction.
              /// </summary>
              /// <param name='direction'>
              /// If set to <c>true</c> direction.
              /// </param>
        public bool Move(int direction)
        {
            if (CanMove(direction))
            {
                base.Move(direction);
                foreach (Tile t in Level.TileList)
                {
                    if ((t.X == this.X) && (t.Y == this.Y))
                    {
                        t.FireTrigger(null);
                    }
                }
                return true;
            }
            else
            {
                System.Console.WriteLine("Collision detected.");
                return false;
            }
        }
    }
}
```

Directions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AthenaEngine.Framework.Primatives
{
    public abstract class Directions
    {
        public const int LEFT = 0;
        public const int UP = 1;
        public const int RIGHT = 2;
        public const int DOWN = 3;
    }
}
```

Drawable Entity

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Gameplay;

namespace AthenaEngine.Framework.Primatives
{
    /// <summary>
    /// This is an entity which can be drawn.
    /// </summary>
    public class DrawableEntity : CollidableEntity, Interfaces.IDrawable,
Interfaces.IFocusable
    {
        public Color SpriteColor;

        protected SpriteBatch SpriteController;
        protected Texture2D SpriteSheet;
        protected Rectangle SpriteSource;
        public Level level;

        /// <summary>
        /// This is the constructor for the DrawableEntity class.
        /// </summary>
        /// <param name="position">Where the DrawableEntity will start</param>
        /// <param name="size">The size in pixels of the DrawableEntity</param>
        /// <param name="spriteBatch">The SpriteBatch responsible for drawing the
entity</param>
        /// <param name="texture">The texture used to draw the entity</param>
        public DrawableEntity(Vector2 position, Vector2 size, Rectangle SpriteSource,
SpriteBatch spriteBatch, Texture2D tileset, Level level) : base(position, size, level)
        {
            this.SpriteSource = SpriteSource;
            this.SpriteColor = Color.White;
            this.SpriteSheet = tileset;
            this.Position = position;
            this.Size = size;
            this.SpriteController = spriteBatch;
        }

        /// <summary>
        /// This draws the DrawableEntity.
        /// </summary>
        public void Draw()
        {
            this.SpriteController.Draw(this.SpriteSheet, this.Rectangle,
this.SpriteSource, this.SpriteColor);
        }
        public Vector2 Position
        {
            get
            {
                return base.Position;
            }
            set
            {
                base.Position = value;
            }
        }

    }
}
```

Entity

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Primatives;


namespace AthenaEngine.Framework.Primatives
{
    /// <summary>
    /// The Entity class is the superclass for anything.
    /// </summary>
    public abstract class Entity
    {
        protected Vector2 Position;
        protected Vector2 Size;

              /// <summary>
              /// Gets or sets the rectangle.
              /// </summary>
              /// <value>
              /// The rectangle.
              /// </value>
        protected Rectangle Rectangle
        {
            get
            {
                return new Rectangle(X, Y, Width, Height);
            }
            set
            {
                this.Position = new Vector2(value.X, value.Y);
                this.Size = new Vector2(value.Width, value.Height);
            }
        }

        public int X
        {
            get
            {
                return (int) Position.X;
            }
            set
            {
                Position.X = value;
            }
        }
        public int Y
        {
            get
            {
                return (int)Position.Y;
            }
            set
            {
                Position.Y = value;
            }
        }
        public int Width
        {
            get
            {
                return (int)Size.X;
            }
            set
            {
                Size.X = value;
            }
        }
        public int Height
        {
            get
            {
                return (int)Size.Y;
            }
            set
            {
                Size.Y = value;
            }
        }

        protected void Move(int direction)
        {
            switch (direction)
            {
                case Directions.UP:
                    this.Y -= 25;
                    break;
                case Directions.DOWN:
                    this.Y += 25;
                    break;
                case Directions.LEFT:
                    this.X -= 25;
                    break;
                case Directions.RIGHT:
                    this.X += 25;
                    break;
                default:
                    throw new InvalidOperationException("Invalid direction to move
in");
            }
        }

    }
}
```

Level Loader XML

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Xml.Linq;
using System.Reflection;

using AthenaEngine.Framework.Gameplay;

namespace AthenaEngine.Framework.Systems
{
    /// <summary>
    /// the LevelLoader is used to load levels.
    /// </summary>
    public static class LevelLoaderXml
    {
        public class Sprite
        {
            public int X;
            public int Y;
            public string Name;
            public bool Collides;

            public Sprite(int x, int y, string name, bool collides)
            {
                this.X = x;
                this.Y = y;
                this.Name = name;
                this.Collides = collides;
            }
        }

              /// <summary>
              /// Load the specified levelName and level.
              /// </summary>
              /// <param name='levelName'>
              /// Level name.
              /// </param>
              /// <param name='level'>
              /// Level.
              /// </param>
        public static List<Tile> Load (string levelName, Level level)
        {
            List<Tile> LevelList = new List<Tile>();
            XDocument Document = XDocument.Load("Content/Levels/" + levelName +
"/tiles.xml");
            XElement Level = Document.Element("level");
            string Tileset = Level.Attribute("tileset").Value;
            int RowLength = Int32.Parse(Level.Attribute("rowlength").Value);

            Dictionary<string, Sprite> SpriteList = LoadSpriteSet(Tileset);

            System.Console.WriteLine(Tileset);
            int TileSize = Int32.Parse(Level.Attribute("tilesize").Value);

            IEnumerable<XElement> Tiles = Level.Elements("tile");
            int Y = 0;
            int X = 0;
            foreach (XElement Tile in Tiles)
            {
                string SpriteName = Tile.Element("sprite").Value;
                Sprite sprite = SpriteList[SpriteName];


                Tile newTile = new Tile(X * TileSize, Y * TileSize, Tileset, sprite.X,
sprite.Y, level);

                var Te = Tile.Elements();
                foreach(var Tee in Te)
                {
                    if (Tee.Name == "trigger")
                    {
                        newTile.AddTrigger(Tee.Value);
                    }
                }
                newTile.Collides = sprite.Collides;

                LevelList.Add(newTile);
                X++;
                if (X >= RowLength)
                {
                    Y++;
                    X = 0;
                }

            }

            return LevelList;
        }

              /// <summary>
              /// Loads the sprite set.
              /// </summary>
              /// <returns>
              /// The sprite set.
              /// </returns>
              /// <param name='spriteSet'>
              /// Sprite set.
              /// </param>
        public static Dictionary<string, Sprite> LoadSpriteSet (string spriteSet)
        {
            Dictionary<string, Sprite> SpriteList = new Dictionary<string, Sprite>();

            XDocument TileSet = XDocument.Load("Content/Tilesets/" + spriteSet +
".xml");
            XElement TileSetDeclaration = TileSet.Element("tileset");
            int TileSize =
Int32.Parse(TileSetDeclaration.Attribute("tilesize").Value);

            IEnumerable<XElement> Sprites = TileSetDeclaration.Elements("sprite");
            foreach (XElement Sprite in Sprites)
            {
                int x = Int32.Parse(Sprite.Element("x").Value) * TileSize;
                int y = Int32.Parse(Sprite.Element("y").Value) * TileSize;

                string name = Sprite.Element("name").Value;
                bool collides = Boolean.Parse(Sprite.Element("collides").Value);

                Sprite newSprite = new Sprite(x, y, name, collides);
                SpriteList.Add(name, newSprite);
            }

            return SpriteList;
        }
    }
}
```

Resource Manager

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace AthenaEngine.Framework.Systems
{
    /// <summary>
    /// The ResourceManager class manages resources on behalf of the game.
    /// </summary>
    /// <typeparam name="T">The type of resource to manage</typeparam>
       public class ResourceManager<T>
       {
        private Game game;
        private Dictionary<String, T> Resources = new Dictionary<String, T>();

        /// <summary>
        /// Creates a new ResourceManager to manage resources for a game.
        /// </summary>
        /// <param name="game">The game to manage resources for.</param>
        public ResourceManager(Game game)
        {
            this.game = game;
        }

        /// <summary>
        /// Add a new resource to the resources list maintained by the
ResourceManager.
        /// </summary>
        /// <param name="key">The key to associate the resource with</param>
        /// <param name="resource">The resource associated with the key</param>
        /// <returns></returns>
        public T Add(string key, T resource)
        {
            this.Resources.Add(key, resource);
            return resource;
        }

        /// <summary>
        /// Gets the resource stored associated with a key.
        /// </summary>
        /// <param name="key">The key to find the resource associated with.</param>
        /// <returns></returns>
        public T Get(string key)
        {
            return this.Resources[key];
        }
       }
}
```

Triggers

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AthenaEngine.Framework.Systems
{
       /// <summary>
       /// Triggers.
       /// </summary>
    class Triggers
    {
              /// <summary>
              /// This was used to test
              /// </summary>
        public static void test ()
        {
            System.Console.WriteLine("You stepped on a tile.");
        }

              /// <summary>
              /// Random encounter test
              /// </summary>
        public static void encounter()
        {

            Random Rng = new Random();
            int encounter = Rng.Next(1,20);

            if((encounter %4 == 0))
            {
                Console.WriteLine("Random Encounter!");
            }
        }
    }
}
```

UI

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework;
using AthenaEngine.Framework.Gameplay;
using AthenaEngine.Framework.Interfaces;
using AthenaEngine.Framework.Primatives;
using AthenaEngine.Framework.Systems;

namespace AthenaEngine.Framework.UI
{
       /// <summary>
       /// UI
       /// </summary>
    public class UI
    {
        SpriteBatch SpriteBatch;
        ResourceManager<Texture2D> TextureManager;
        ResourceManager<SpriteFont> FontManager;
        public Level Level;
        /*
        public UIButton(Vector2 position, Vector2 size, SpriteBatch spriteBatch,
Texture2D texture) : base(position, size, spriteBatch, texture)
        {
            this.SpriteColor = Color.White;
            this.SpriteTexture = texture;
            this.SpriteRectangle = new Rectangle((int)position.X, (int)position.Y,
(int)size.X, (int)size.Y);
            this.SpriteController = spriteBatch;
        }
         */

              /// <summary>
              /// Initializes a new instance of the <see
cref="AthenaEngine.Framework.UI.UI"/> class.
              /// </summary>
              /// <param name='spriteBatch'>
              /// Sprite batch.
              /// </param>
              /// <param name='textureManager'>
              /// Texture manager.
              /// </param>
              /// <param name='fontManager'>
              /// Font manager.
              /// </param>
        public UI (SpriteBatch spriteBatch, ResourceManager<Texture2D> textureManager,
ResourceManager<SpriteFont> fontManager)
        {
            this.SpriteBatch = spriteBatch;
            this.TextureManager = textureManager;
            this.FontManager = fontManager;
        }
```

```
              /// <summary>
              /// The buttons.
              /// </summary>
        private List<UIButton> Buttons = new List<UIButton>();

              /// <summary>
              /// Adds the button.
              /// </summary>
              /// <param name='position'>
              /// Position.
              /// </param>
              /// <param name='label'>
              /// Label.
              /// </param>
        public void AddButton(Vector2 position, string label)
        {
            // Buttons.Add(new UIButton(position, new Vector2(100, 40), SpriteBatch,
TextureManager.Get("blank")));
            Buttons.Add(new UIButton(new Rectangle((int)position.X, (int)position.Y,
100, 40), this.SpriteBatch, TextureManager.Get("blank"), Color.Orange, label,
FontManager.Get("SpriteFont1")));
        }

              /// <summary>
              /// Draw this instance.
              /// </summary>
        public void Draw ()
        {
            foreach (UIButton btn in Buttons)
            {
                btn.Draw();
            }
        }
    }
}
```

<u>UI Button</u>

```
using System;
using System.Collections.Generic;
using System.Linq;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework;
using AthenaEngine.Framework.Gameplay;
using AthenaEngine.Framework.Interfaces;
using AthenaEngine.Framework.Primatives;
using AthenaEngine.Framework.Systems;

namespace AthenaEngine.Framework.UI
{
        /// <summary>
        /// User interface button.
        /// </summary>
    public class UIButton
    {
        private SpriteBatch Batch;
        private Texture2D Texture;
        private SpriteFont Font;
        private Color Color;
        private string Label;
        public Level Level;

        private Rectangle SpriteRect;

              /// <summary>
              /// Initializes a new instance of the <see
cref="AthenaEngine.Framework.UI.UIButton"/> class.
              /// </summary>
              /// <param name='rectangle'>
              /// Rectangle.
              /// </param>
              /// <param name='spriteBatch'>
              /// Sprite batch.
              /// </param>
              /// <param name='texture'>
              /// Texture.
              /// </param>
              /// <param name='color'>
              /// Color.
              /// </param>
              /// <param name='label'>
              /// Label.
              /// </param>
              /// <param name='font'>
              /// Font.
              /// </param>
        public UIButton (Rectangle rectangle, SpriteBatch spriteBatch, Texture2D
texture, Color color, string label, SpriteFont font)
        {
            this.SpriteRect = rectangle;
            this.Batch = spriteBatch;
            this.Texture = texture;
            this.Color = color;
            this.Label = label;
            this.Font = font;
        }

              /// <summary>
              /// Draw this instance.
              /// </summary>
        public void Draw ()
        {
            // DrawableEntity Base = new DrawableEntity(new Vector2(SpriteRect.X,
SpriteRect.Y), new Vector2(SpriteRect.Width, SpriteRect.Height), Batch, Texture,
Level);
            // DrawableEntity Stroke = new DrawableEntity(new Vector2(SpriteRect.X -
2, SpriteRect.Y - 2), new Vector2(SpriteRect.Width + 4, SpriteRect.Height + 4), Batch,
Texture, Level);
            // Stroke.SpriteColor = Color.Black;
            // Base.SpriteColor = this.Color;
            // Stroke.Draw();
            // Base.Draw();
            // int x = (int)SpriteRect.X + Font.MeasureString(Label).X / 2

            int X = SpriteRect.X + (SpriteRect.Width - (int)
Font.MeasureString(Label).X) / 2;
            int Y = SpriteRect.Y;
            Batch.DrawString(Font, Label, new Vector2(X, Y), Color.Black);
        }
    }
}
```

Camera 2D

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Interfaces;

namespace AthenaEngine.Framework
{
	/// <summary>
	/// Camera2D
	/// </summary>
	public class Camera2D : GameComponent
	{
		private Vector2 _position;
		protected float _viewportHeight;
		protected float _viewportWidth;

		/// <summary>
		/// Initializes a new instance of the <see
cref="AthenaEngine.Framework.Camera2D"/> class.
		/// </summary>
		/// <param name='game'>
		/// Game.
		/// </param>
		public Camera2D (Game game) : base (game)
		{

		}

		/// <summary>
		/// Gets or sets the position.
		/// </summary>
		/// <value>
		/// The position.
		/// </value>
		public Vector2 Position {
			get { return _position; }
			set { _position = value; }
		}

		public float Rotation { get; set; }
```

```
            public Vector2 Origin { get; set; }
            public float Scale { get; set; }
            public Vector2 ScreenCenter { get; protected set; }
            public Matrix Transform { get; set; }
            public IFocusable Focus { get; set; }
            public float MoveSpeed { get; set; }

       /// <summary>
    /// Called when the GameComponent needs to be initialized.
    /// </summary>
    ///
    public override void Initialize()
    {
        _viewportWidth = Game.GraphicsDevice.Viewport.Width;
        _viewportHeight = Game.GraphicsDevice.Viewport.Height;

        ScreenCenter = new Vector2(_viewportWidth/2, _viewportHeight/2);
        Scale = 1;
        MoveSpeed = 1.25f;

        base.Initialize();
    }

            /// <summary>
            /// Update the specified gameTime.
            /// </summary>
            /// <param name='gameTime'>
            /// Game time.
            /// </param>
    public override void Update(GameTime gameTime)
    {
        // Create the Transform used by any
        // spritebatch process
        Transform = Matrix.Identity*
                    Matrix.CreateTranslation(-Position.X, -Position.Y, 0)*
                    Matrix.CreateRotationZ(Rotation)*
                    Matrix.CreateTranslation(Origin.X, Origin.Y, 0)*
                    Matrix.CreateScale(new Vector3(Scale, Scale, Scale));

        Origin = ScreenCenter / Scale;

        // Move the Camera to the position that it needs to go
        var delta = (float) gameTime.ElapsedGameTime.TotalSeconds;

        _position.X += (Focus.Position.X - Position.X) * MoveSpeed * delta;
        _position.Y += (Focus.Position.Y - Position.Y) * MoveSpeed * delta;

        base.Update(gameTime);
    }

    /// <summary>
    /// Determines whether the target is in view given the specified position.
    /// This can be used to increase performance by not drawing objects
    /// directly in the viewport
    /// </summary>
    /// <param name="position">The position.</param>
    /// <param name="texture">The texture.</param>
    /// <returns>
    ///     <c>true</c> if [is in view] [the specified position]; otherwise,
<c>false</c>.
    /// </returns>
    public bool IsInView(Vector2 position, Texture2D texture)
    {
        // If the object is not within the horizontal bounds of the screen

        if ( (position.X + texture.Width) < (Position.X - Origin.X) || (position.X) >
(Position.X + Origin.X) )
            return false;

        // If the object is not within the vertical bounds of the screen
        if ((position.Y + texture.Height) < (Position.Y - Origin.Y) || (position.Y) >
(Position.Y + Origin.Y))
            return false;

        // In View
        return true;
    }
       }
}
```

<u>Bounding Box 2D Test</u>

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

using AthenaEngine.Framework.Primatives;

namespace AthenaTest.Engine_Tests.Primatives_Testing
{
    [TestClass]
    public class BoundingBox2DTest
    {
        [TestMethod]
        public void Equality_WithSameRectangle_IsEqual()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = FirstBoundingBox2D;

            bool BoxesEqual = FirstBoundingBox2D.Equals(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesEqual, true);
        }

        [TestMethod]
        public void Equality_WithIdenticalRectangle_IsEqual()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);
            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);

            bool BoxesEqual = FirstBoundingBox2D.Equals(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesEqual, true);
        }

        [TestMethod]
        public void Equality_WithDifferentRectangle_IsNotEqual()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);

            Vector2 SecondBoundingBox2D_XY = new Vector2(32, 15);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(81, 542);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesEqual = FirstBoundingBox2D.Equals(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesEqual, false);
        }

        [TestMethod]
        public void Equality_WithItself_IsEqual()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);

            bool BoxesEqual = FirstBoundingBox2D.Equals(FirstBoundingBox2D);

            Assert.AreEqual<bool>(BoxesEqual, true);
        }

        [TestMethod]
        public void Collisions_WithItself_IsTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(FirstBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, true);
        }

        [TestMethod]
        public void Collisions_WithTotallyDifferentRectangle_IsNotTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);
            Vector2 SecondBoundingBox2D_XY = new Vector2(151, 214);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(81, 542);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, false);
        }

        [TestMethod]
        public void Collisions_WithEnvelopedRectangle_IsTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(0, 0);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(100, 100);

            Vector2 SecondBoundingBox2D_XY = new Vector2(25, 25);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(25, 25);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, true);
        }

        [TestMethod]
        public void Collisions_WithRectangleOnLeft_IsFalse()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(0, 10);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, false);
        }

        [TestMethod]
        public void Collisions_WithRectangleOnRight_IsFalse()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(20, 10);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);
            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, false);
        }

        [TestMethod]
        public void Collisions_WithRectangleOnTop_IsFalse()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(10, 0);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, false);
        }

        [TestMethod]
        public void Collisions_WithRectangleOnBottom_IsFalse()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(10, 20);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, false);
        }

        [TestMethod]
        public void Collisions_WithRectangleCollidingLeft_IsTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(1, 10);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);
```

```
            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, true);
        }

        [TestMethod]
        public void Collisions_WithRectangleCollidingRight_IsTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(19, 10);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, true);
        }

        [TestMethod]
        public void Collisions_WithRectangleCollidingTop_IsTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(10, 1);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, true);
        }

        [TestMethod]
        public void Collisions_WithRectangleCollidingBottom_IsTrue()
        {
            Vector2 FirstBoundingBox2D_XY = new Vector2(10, 10);
            Vector2 FirstBoundingBox2D_WidthHeight = new Vector2(10, 10);

            Vector2 SecondBoundingBox2D_XY = new Vector2(10, 19);
            Vector2 SecondBoundingBox2D_WidthHeight = new Vector2(10, 10);

            BoundingBox2D FirstBoundingBox2D = new
BoundingBox2D(FirstBoundingBox2D_XY, FirstBoundingBox2D_WidthHeight);
            BoundingBox2D SecondBoundingBox2D = new
BoundingBox2D(SecondBoundingBox2D_XY, SecondBoundingBox2D_WidthHeight);

            bool BoxesCollide = FirstBoundingBox2D.CollidesWith(SecondBoundingBox2D);

            Assert.AreEqual<bool>(BoxesCollide, true);
        }
    }
    }
```