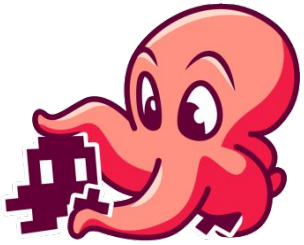


Game
Programming
Laboratory

How not to completely mess up your code

Simone Guggiari

12 Mar 19

How to write a game: demo

Dungeon Coin Collector ©

1. Structure
2. Draw sprites
3. Animation
4. Input
5. Collision
6. Sound
7. UI

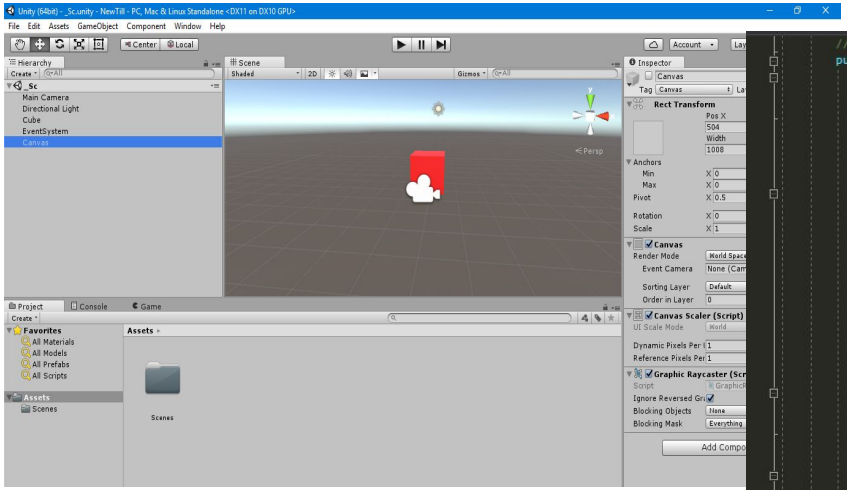
25 min, 52 loc

how many used MonoGame?

STEPS

1. setup project
2. draw player
3. input & move
4. animate
5. show dungeon
6. coins (random + anim.)
7. collect & audio
8. score & UI

Engine vs Framework



unity

UNREAL ENGINE

MONOGAME

```
    /// </summary>
    public override void Update() {
        if (!IsActive) {
            return;
        }

        _pulseTimer = (_pulseTimer + 0.14f) % 6.27f;
        _lightScale = (7f + (float)Math.Sin(_pulseTimer)) / 9f;

        for (int playerIndex = 0; playerIndex < GameManager.NumPlayers; ++playerIndex) {

            // find the vector between the center of the screen and the point light:
            Camera camera = Camera.GetCamera(playerIndex);
            Vector2 mosV = camera.WorldToScreenPoint(_sun);
            _screenCenter = new Vector2(Screen.SplitWidth / 2.0f, Screen.SplitHeight / 2.0f);
            Vector2 vec = mosV - _screenCenter;

            Vector3 cameraSun = camera.transform.Forward;

            _active[playerIndex] = Vector3.Dot(cameraSun, _sun) > 0.0f;

            if (!_active[playerIndex]) {
                continue;
            }

            // Update the flare positions along the vector and update alpha, and rotation
            for (int i = 0; i < _flares[playerIndex].Count; ++i) {
                Flare flare = _flares[playerIndex][i];
                flare.Pos = flare.Offset * vec + _screenCenter;

                // (don't rotate the main flare - looks strange)
                if (i != 2) {
                    //(calculates the angle of the vector so we can make sprites face the light)
                    flare.Rot = (float)Math.Atan2(vec.Y, vec.X);
                }

                // last flare needs angle to be corrected (should just do on sprite sheet)
                if (i + 1 == _flares[playerIndex].Count) {
                    flare.Rot += MathHelper.PiOver2;
                }

                flare.Color = _lightColor;

                // find the percentage distance the flare is from the center (approximately)
                // we divide based on approximate maximum distance we want between the opposite ends of the vector (to control fade out rate)
                Vector2 radiusVec = (i != 2) ? vec / (15 * 300) : vec / (15 * 450);

                // now scale the alpha transparency by this percent:
```

Complexity rises quickly

Code Files: (191, 14’813 sloc)

- Engine (79) today: half presented
- Setup (30)
- Scripts (82)

Content Files: (246)

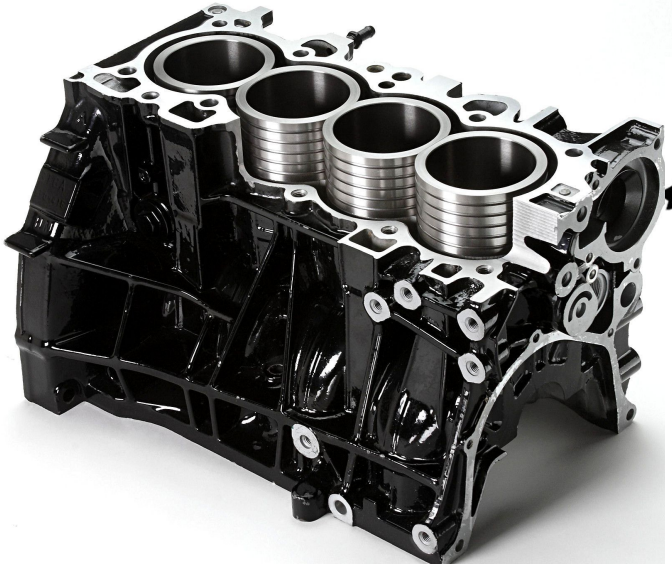
- Audio (40)
- Images (126)
- Models (56)
- Effects (24)

Idea

Many different options

Engine vs Scripts

We don't want to pollute game with accessing texture or input
Separate them



Game Engine

Game Scripts

Idea

Everything is a GameObject

They have Components

You only write components

Encapsulation 120%

That's it

(sorry unity for copying)

unity

Game Structure (static classes)

Input

Audio

Content

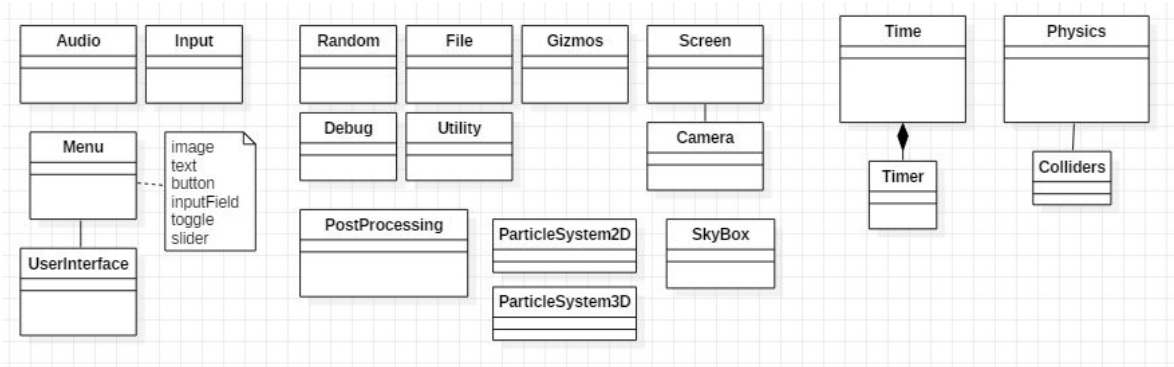
Screen + Cameras

Time

Physics

Debug

UI/Menu



Game structure (GO)

Gameobject

Component

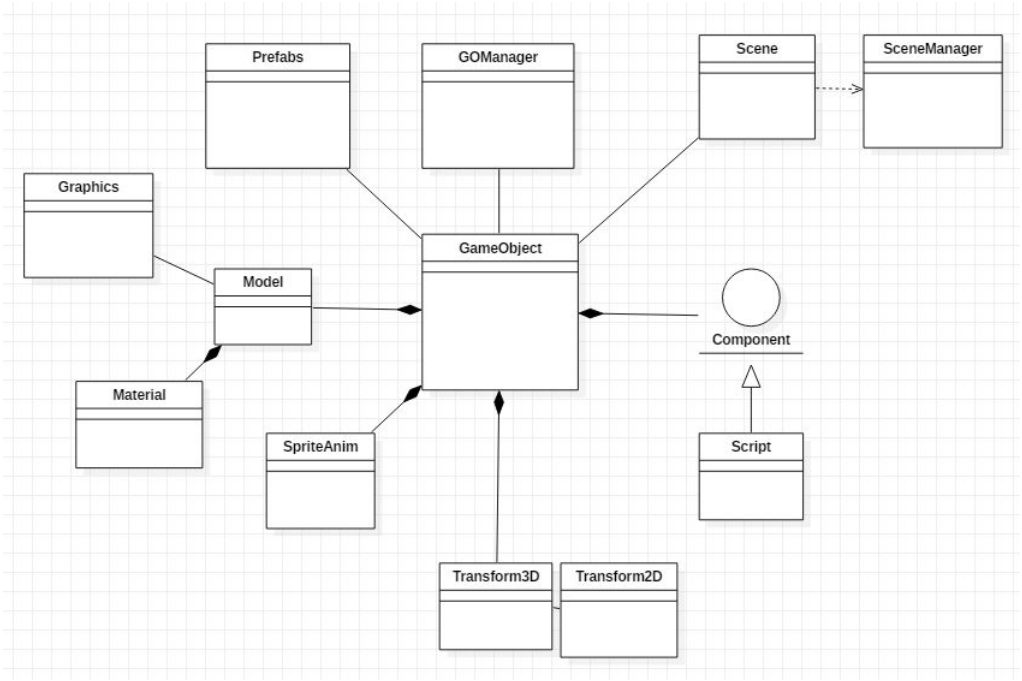
Script

Transform

Prefabs

Scene

Graphics



Input

Keyboard

Mouse

Gamepad

buttons + sticks + vibration

Up/Down/Held

Manage connect/disconnect/switch (id)

Input
+KeyboardState old new +MouseState old new +GamePadState[] olds news +enum axis horiz_vert +enum dir left_right_up_down +enum stick left_right
+bool GetAxis(dir) +float GetAxisRaw(axis) +bool GetKey(key) +bool GetKeyDown(key) +bool GetKeyUp(key) +vector2 MousePos() +vector2 MouseDelta() +void SetMousePos(vector2) +int MouseWheel() +bool GetMouseButton(index) +bool GetMouseButtonDown(index) +bool GetMouseButtonUp(index) +bool GetButton(id, button) +bool GetButtonDown(id, button) +bool GetButtonUp(id, button) +vector2 GetThumbStick(id, stick) +float GetTrigger(id) +bool GetTriggerDown(id) +bool GetTriggerUp(id) +void Vibrate(id, duration, force) +void StopVibrate(id)

Audio

Play sound/music (at position)

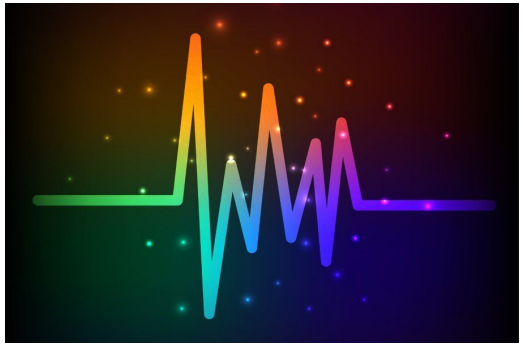
Manage dictionary<string, SoundEffect>

```
AudioM.Play(“boom”);
```

Different volume channels

music/effects/speech

Pitch



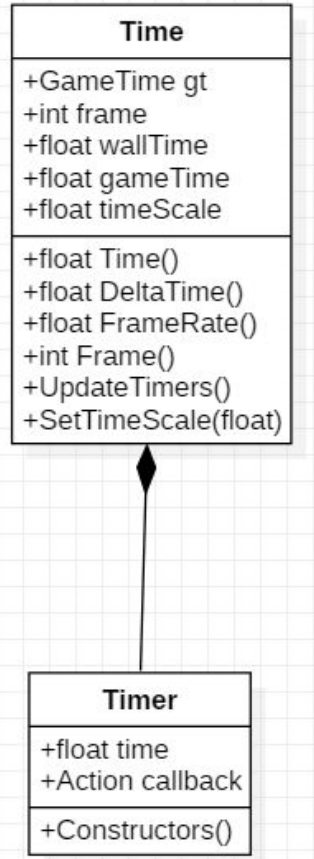
Audio
+dictionary string_soundEffect +dictionary string_song +volume +soundEffectsVolue +musicVolume +pitchRange +transform Listener
+BuildSoundLibraries() +SetVolume() +PlayEffect(name, vector3 position) +PlaySong(string name)

Time

Provide access to

- time (real/game)
- deltaTime
- timeScale (pause/make faster)
- frames

Manage and call timers



Screen / Cameras

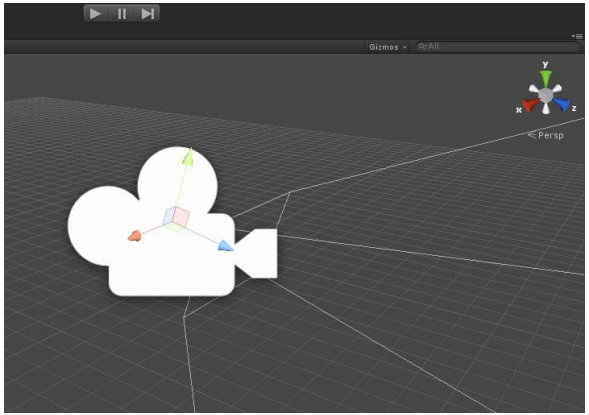
Manage screen (size, title)

Manage viewports

- 1-2-4 split screen

Manage cameras

- near/far
- fov/projection
- transform
- rays/points



Screen

+int width height
+string title
+int numSplits
+Camera[] cams
+Viewport[] views
+RasterizerState
+bool verticalSplit

+SetupWindow()
+SetupViewports()
+SetupCams()
+SetupRasterizer()
+float AspectRatio()

Camera

+float near far
+float fov
+float focusDist
+projection type
+viewport
+matrix projection view
+transform

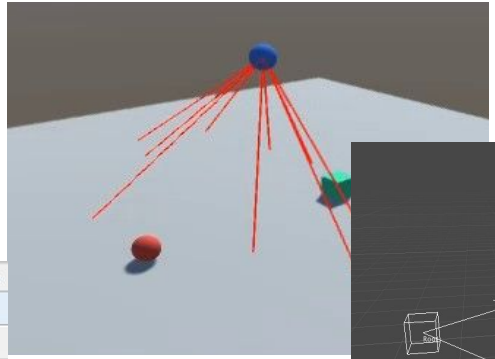
+Constructor()
+ComputeProjection()
+Ray ScreenPointToRay()
+Vector2 WorldToScreenPoint()
+static GetCamera()

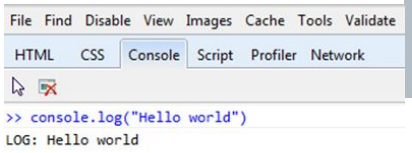
Utility

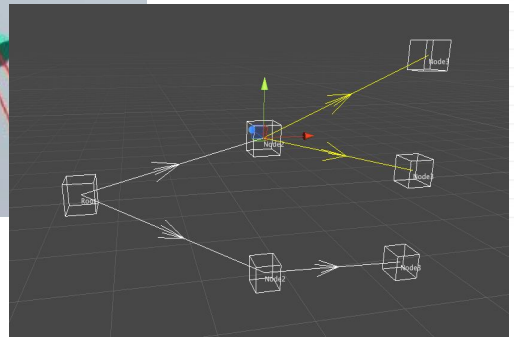
Safe resource/file access (Content)

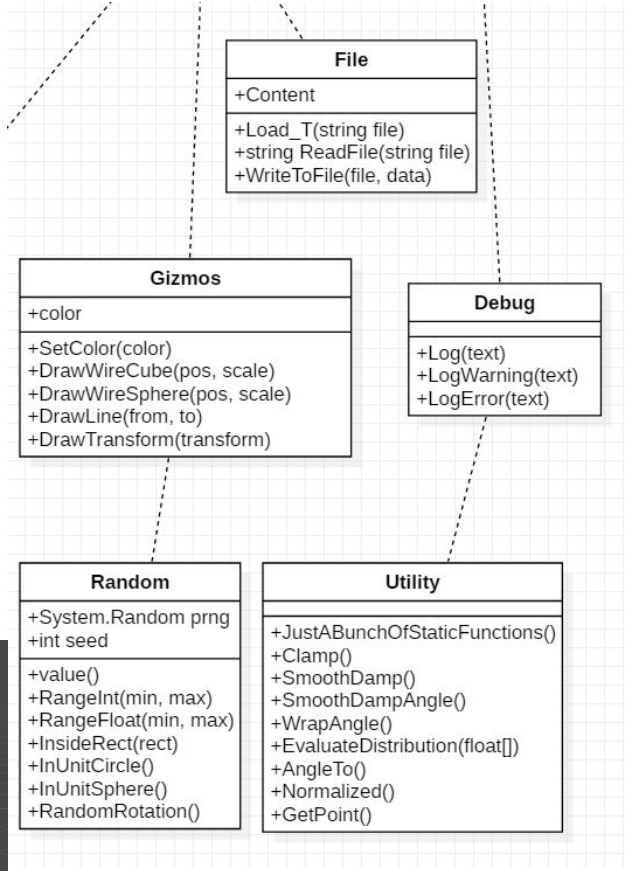
In-game debug (Log, Gizmos)

Math helpers (Random, Utility)









UI/Menu

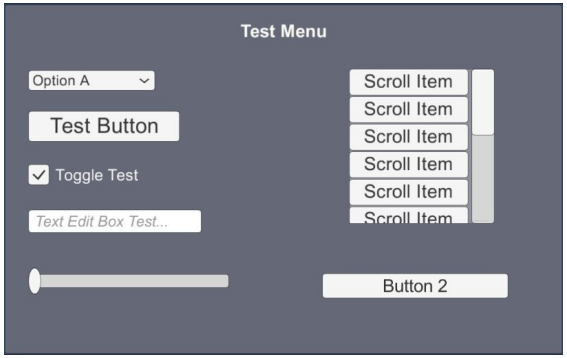
HUD (in-game info)

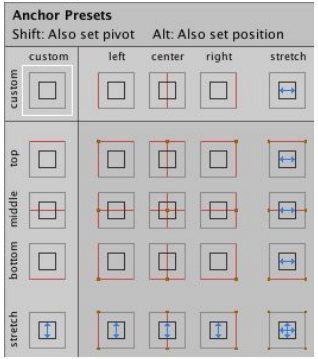
Menu screens

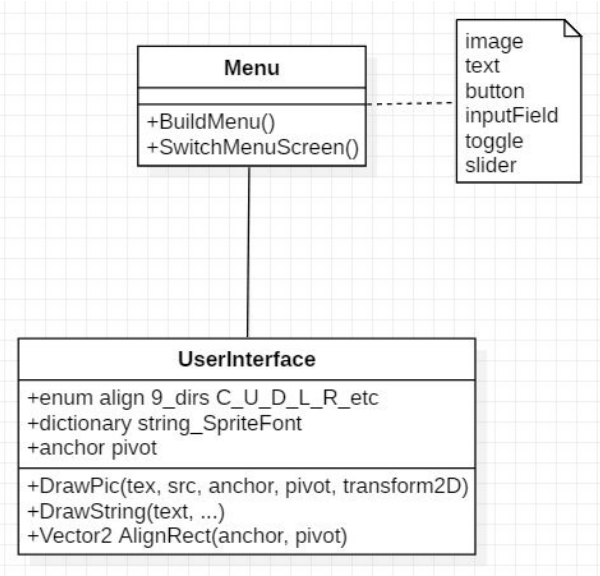
- images/text
- buttons
- sliders/toggles
- input fields

Screen size adjustment

- Anchors







Effects

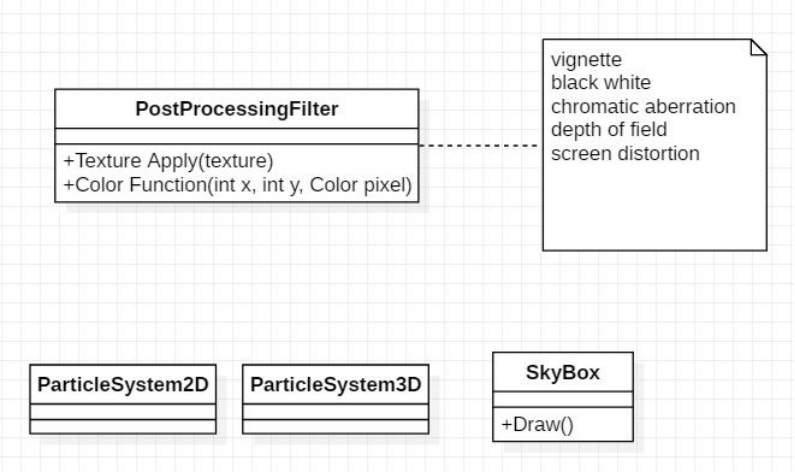
Particle systems (2D/3D)

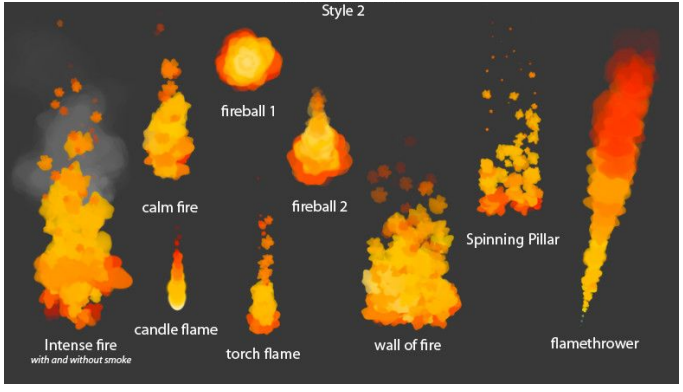
    2D: simple sprite atlas

    3D: particles moving/rotating

Filters (color/dof/...)

Skybox



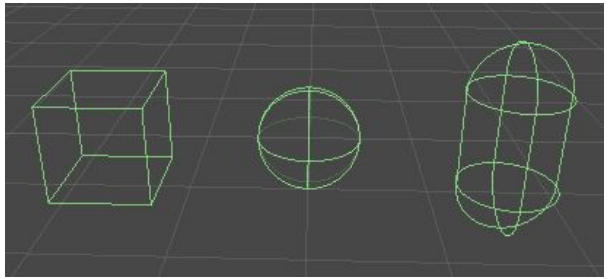


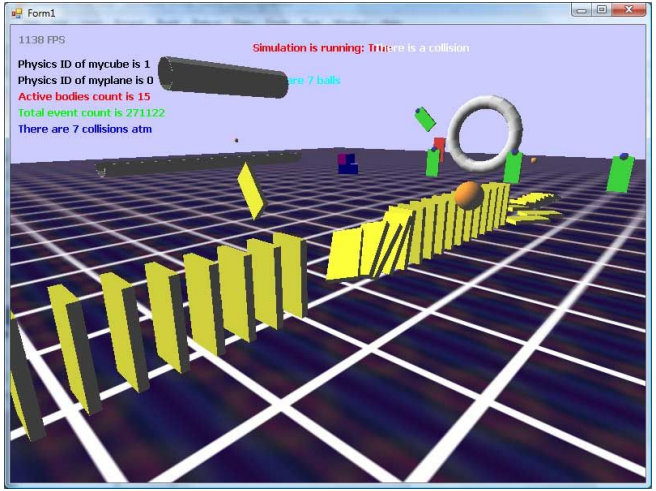
Physics

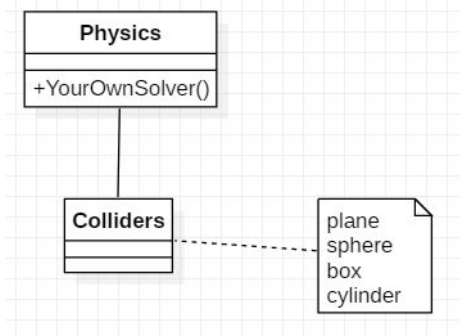
Solver

Colliders

Good luck!







Library

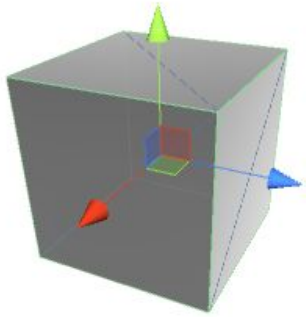
Box 2D (2D)

Bullet (3D)

Pay attention to costs!

(learn, include, make work)

GameObject, Component, Script



Every in-game entity

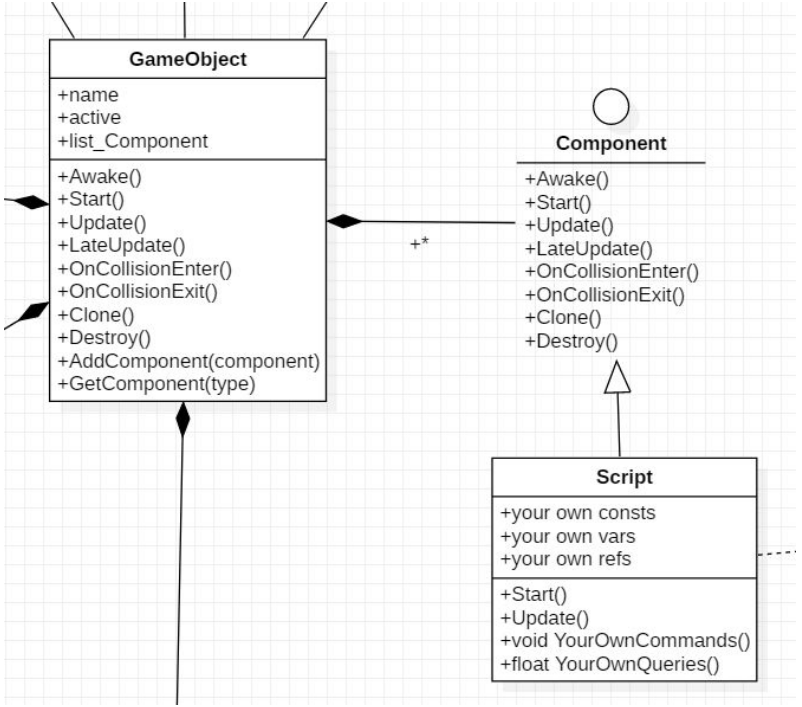
Manages many components

functionality

Has transform (pos/rot/scale)

Has graphics component

model (3D) / sprite (2D)



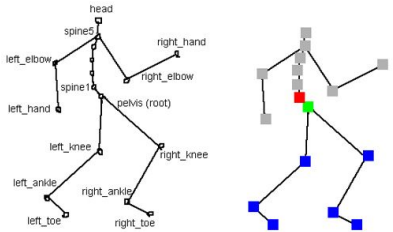
Transform

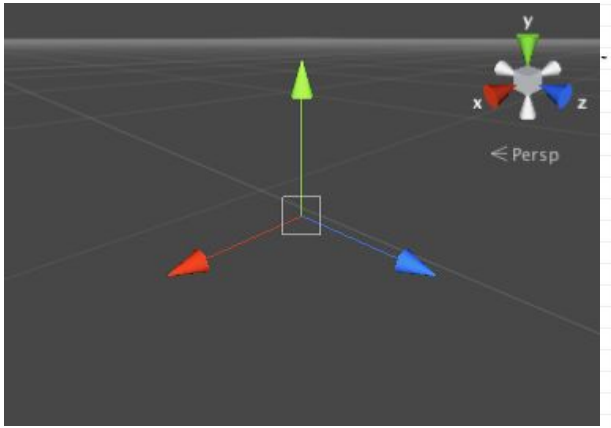
Position, rotation, scale

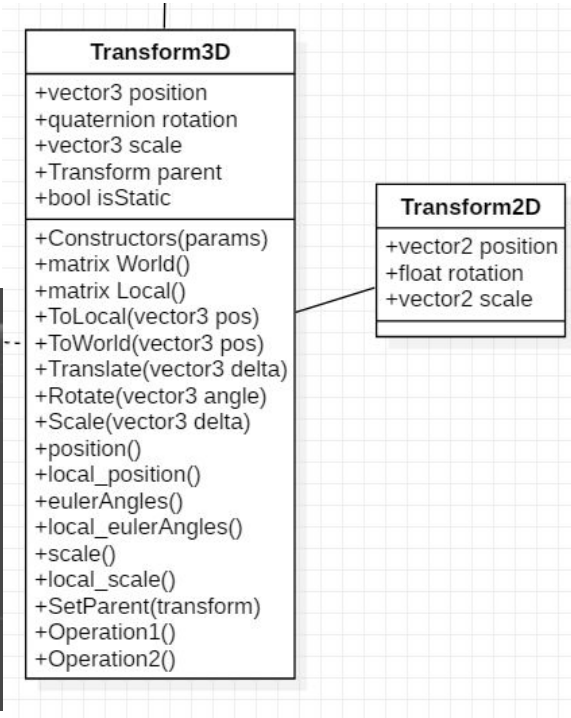
(3D vs 2D)

Math heavy

Transform chains







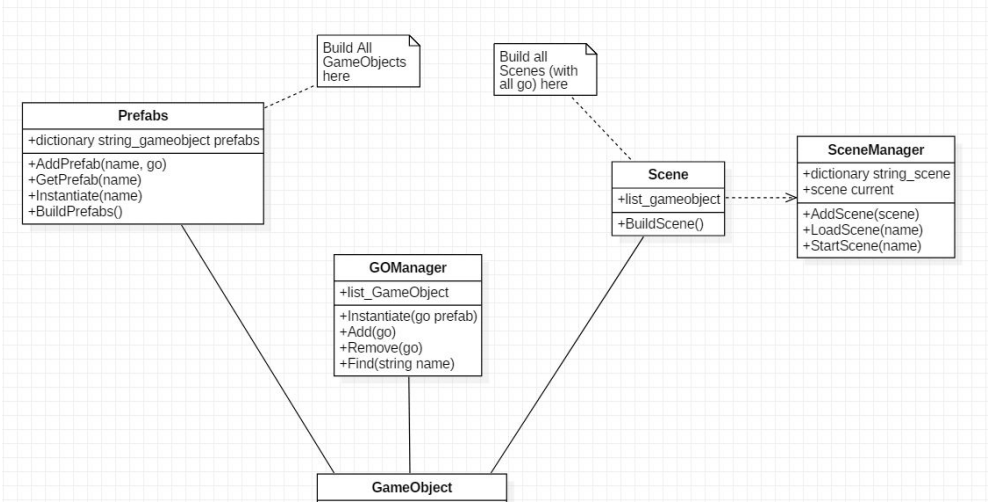
Managing Go

Build prefabs

- sample objects to instantiate
- (e.g. coin/enemy)
- factory pattern

Create scenes (levels)

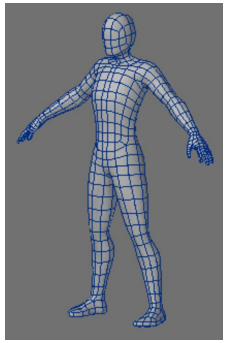
- static level



Graphics

Sprites (2D)

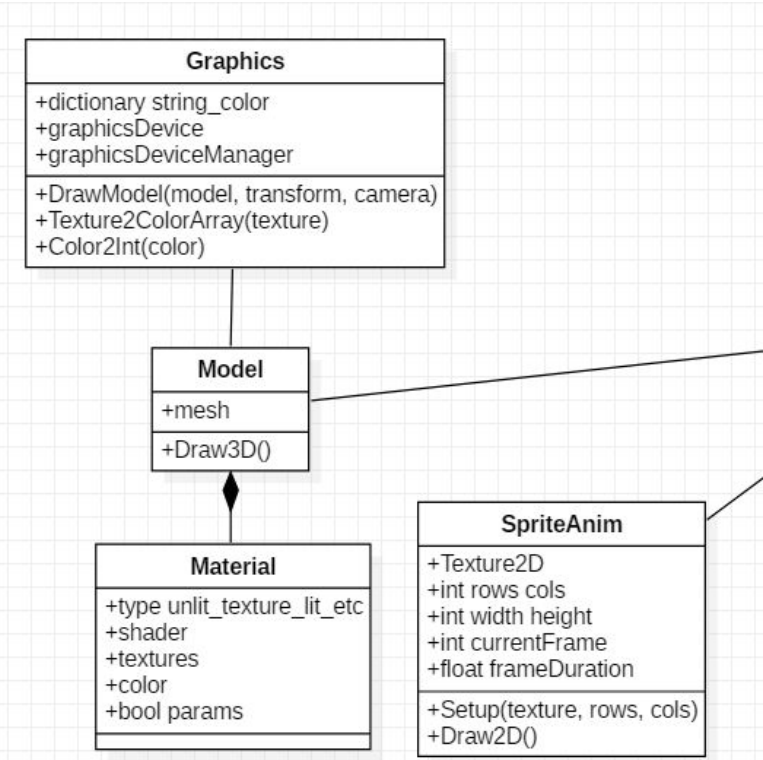
atlas, animations

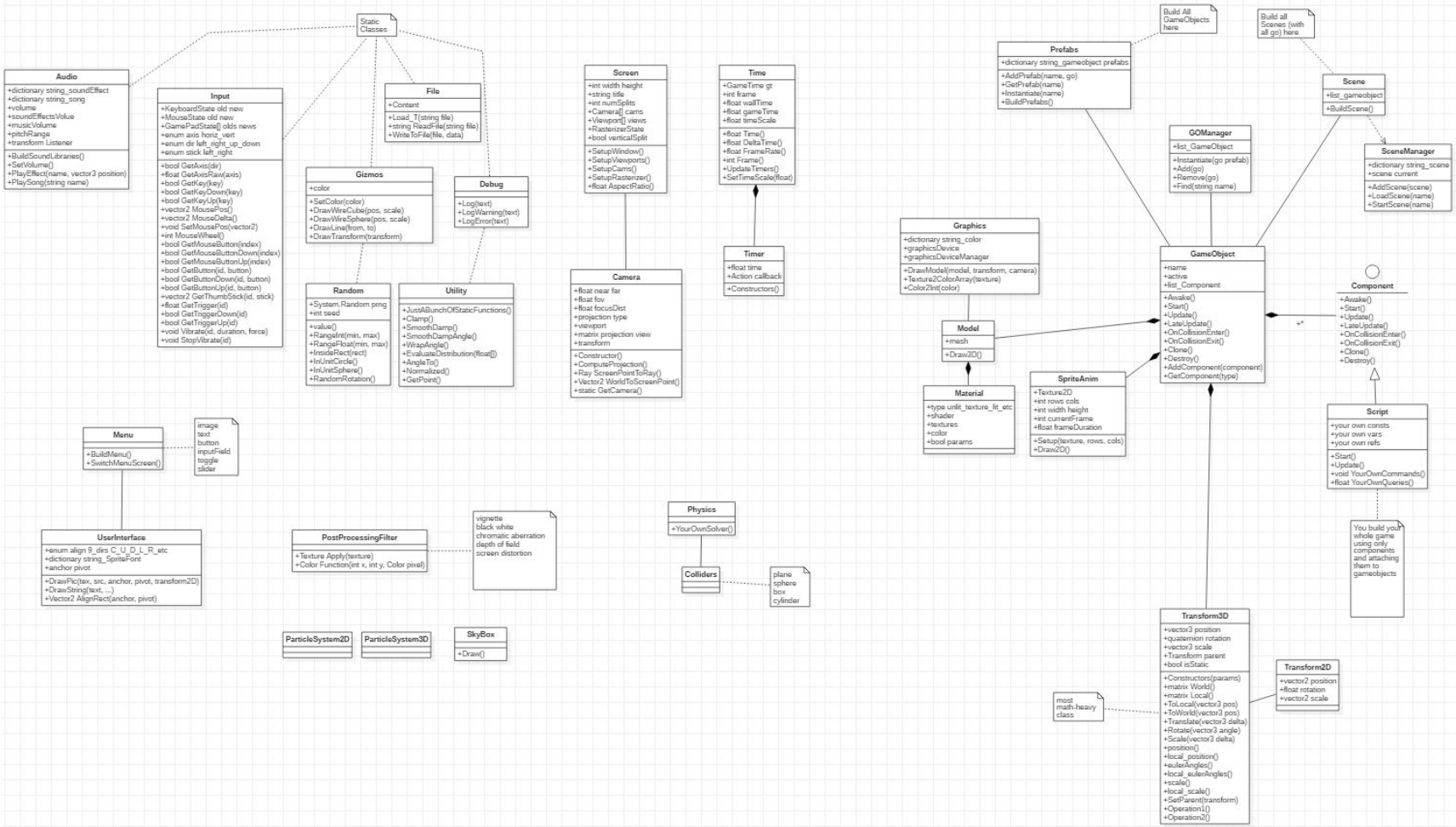


Models (3D)

mesh, material, texture, shader

GraphicsDeviceManager




Static Classes
Audio
+dictionary string_soundEffect
+dictionary string_song
+volume
+soundEffectsVolue
+musicVolume
+pitchRange
+transform Listener
+BuildSoundLibraries()
+SetVolume()
+PlayEffect(name, vector3 position)
+PlaySong(string name)
Input
+KeyboardState old new
+MouseState old new
+GamePadState[] olds news
+enum axis horiz_vert
+enum dir left_right_up_down
+enum stick left_right
+bool GetAxis(dir)
+float GetAxisRaw(axis)
+bool GetKey(key)
+bool GetKeyDown(key)
+bool GetKeyUp(key)
+vector2 MousePos()
+vector2 MouseDelta()
+void SetMousePos(vector2)
+int MouseWheel()
+bool GetMouseButton(index)
+bool GetMouseButtonDown(index)
+bool GetMouseButtonUp(index)
+bool GetButton(id, button)
+bool GetButtonDown(id, button)
+bool GetButtonUp(id, button)
+vector2 GetThumbStick(id, stick)
+float GetTrigger(id)
+bool GetTriggerDown(id)
+bool GetTriggerUp(id)
+void Vibrate(id, duration, force)
+void StopVibrate(id)
File
+Content
+Load_T(string file)
+string ReadFile(string file)
+WriteToFile(file, data)
Gizmos
+color
+SetColor(color)
+DrawWireCube(pos, scale)
+DrawWireSphere(pos, scale)
+DrawLine(from, to)
+DrawTransform(transform)
Debug
+Log(text)
+LogWarning(text)
+LogError(text)
Random
+System.Random prng
+int seed
+value()
+RangeInt(min, max)
+RangeFloat(min, max)
+InsideRect(rect)
+InUnitCircle()
+InUnitSphere()
+RandomRotation()
Utility
+JustABunchOfStaticFunctions()
+Clamp()
+SmoothDamp()
+SmoothDampAngle()
+WrapAngle()
+EvaluateDistribution(float[])
+AngleTo()
+Normalized()
+GetPoint()
Screen
+int width height
+string title
+int numSplits
+Camera[] cams
+Viewport[] views
+RasterizerState
+bool verticalSplit
+SetupWindow()
+SetupViewports()
+SetupCams()
+SetupRasterizer()
+float AspectRatio()
Time
+GameTime gt
+int frame
+float wallTime
+float gameTime
+float timeScale
+float Time()
+float DeltaTime()
+float FrameRate()
+int Frame()
+UpdateTimers()
+SetTimeScale(float)
Timer
+float time
+Action callback
+Constructors()
Camera
+float near far
+float fov
+float focusDist
+projection type
+viewport
+matrix projection view
+transform
+Constructor()
+ComputeProjection()
+Ray ScreenPointToRay()
+Vector2 WorldToScreenPoint()
+static GetCamera()
Build All GameObjects here
Build all Scenes (with all go) here
Prefabs
+dictionary string_gameobject prefabs
+AddPrefab(name, go)
+GetPrefab(name)
+Instantiate(name)
+BuildPrefabs()
Scene
+list_gameobject
+BuildScene()
GOManager
+list_GameObject
+Instantiate(go prefab)
+Add(go)
+Remove(go)
+Find(string name)
SceneManager
+dictionary string_scene
+scene current
+AddScene(scene)
+LoadScene(name)
+StartScene(name)
Graphics
+dictionary string_color
+graphicsDevice
+graphicsDeviceManager
+DrawModel(model, transform, camera)
+Texture2ColorArray(texture)
+Color2Int(color)
GameObject
+name
+active
+list_Component
+Awake()
+Start()
+Update()
+LateUpdate()
+OnCollisionEnter()
+OnCollisionExit()
+Clone()
+Destroy()
+AddComponent(component)
+GetComponent(type)
Component
+Awake()
+Start()
+Update()
+LateUpdate()
+OnCollisionEnter()
+OnCollisionExit()
+Clone()
+Destroy()
Model
+mesh
+Draw2D()
Material
+type unlit_texture_lit_etc
+shader
+textures
+color
+bool params
SpriteAnim
+Texture2D
+int rows cols
+int width height
+int currentFrame
+float frameDuration
+Setup(texture, rows, cols)
+Draw2D()
Script
+your own consts
+your own vars
+your own refs
+Start()
+Update()
+void YourOwnCommands()
+float YourOwnQueries()
You build your whole game using only components and attaching them to gameobjects
Menu
+BuildMenu()
+SwitchMenuScreen()
image
text
button
inputField
toggle
slider
UserInterface
+enum align 9_dirs C_U_D_L_R_etc
+dictionary string_SpriteFont
+anchor pivot
+DrawPic(tex, src, anchor, pivot, transform2D)
+DrawString(text, ...)
+Vector2 AlignRect(anchor, pivot)
PostProcessingFilter
+Texture Apply(texture)
+Color Function(int x, int y, Color pixel)
vignette
black white
chromatic aberration
depth of field
screen distortion
Physics
+YourOwnSolver()
Colliders
plane
sphere
box
cylinder
ParticleSystem2D
ParticleSystem3D
SkyBox
+Draw()
Transform3D
+vector3 position
+quaternion rotation
+vector3 scale
+Transform parent
+bool isStatic
+Constructors(params)
+matrix World()
+matrix Local()
+ToLocal(vector3 pos)
+ToWorld(vector3 pos)
+Translate(vector3 delta)
+Rotate(vector3 angle)
+Scale(vector3 delta)
+position()
+local_position()
+eulerAngles()
+local_eulerAngles()
+scale()
+local_scale()
+SetParent(transform)
+Operation1()
+Operation2()
Transform2D
+vector2 position
+float rotation
+vector2 scale
most math-heavy class

Full Game Code

```
// FULL GAME

void INITIALIZE(){
    //setup content
    //setup graphicsManagers

    File.Setup();
    Screen.SetupViewports();
    Screen.SetupCameras();
}

void LOAD(){
    //load gameobjects
    Prefabs.Build()
    Scene.Build()

    Scene.Load()

    //start
    foreach(gameobject) go.Awake();
    foreach(gameobject) go.Start();
}

void UNLOAD(){
    //do nothing. enjoy mem leak
}
```

```
void UPDATE(GameTime gt){
    //ENGINE
    Time.Update(gt);
    Input.Update();

    Audio.Update();
    Physics.Update();

    Gizmos.Update();
    Menu.Update();

    //SRIPTS
    foreach(gameobject) go.Update();
    foreach(gameobject) go.LateUpdate();
}

void DRAW(){
    // 3D
    foreach(viewport){
        //find appropriate camera
        // using graphics, models, material, shaders
        foreach(gameobject) go.Draw3D();
        Particles3D.Draw();
    }

    // 2D
    foreach(viewport){
        // using spriteAnim
        foreach(gameobject) go.Draw2D();
        Particles2D.Draw();
        Menu.Draw();
        UI.Draw();
    }

    //apply post-processing effects
}
```

Content Organization (Folders)

AUDIO (.waw, .mp3)
- effects
- songs
- speech

IMAGES (.png)
- icons
- lightmaps
- logos
- menu
- particles
- powerup
- textures
- UI
- ...

MODELS (.dae, .obj)
- elements
- powerups
- primitives
- scenes
- vehicles

OTHER (.fx, .spritefont)
- effects
- fonts
- shaders

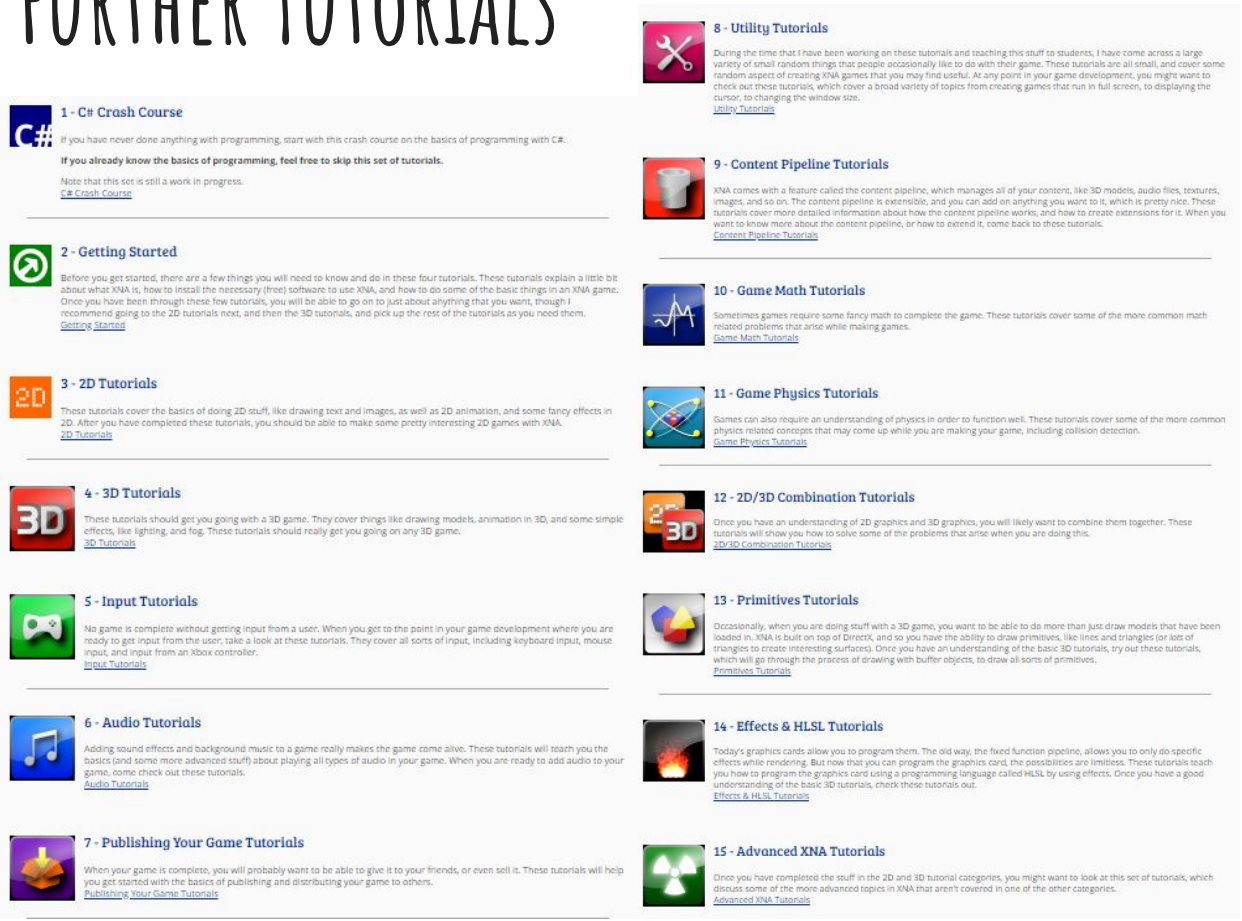
Tips

Have whole engine done in 4 weeks

Think about all this:

(& allocate time)

Further Tutorials



1 - C# Crash Course

If you have never done anything with programming, start with this crash course on the basics of programming with C#.

If you already know the basics of programming, feel free to skip this set of tutorials.

Note that this set is still a work in progress.
C# Crash Course

2 - Getting Started

Before you get started, there are a few things you will need to know and do in these four tutorials. These tutorials explain a little bit about what XNA is, how to install the necessary (free) software to use XNA, and how to do some of the basic things in an XNA game. Once you have been through these few tutorials, you will be able to go on to just about anything that you want, though I recommend going to the 2D tutorials next, and then the 3D tutorials, and pick up the rest of the tutorials as you need them.
Getting Started

3 - 2D Tutorials

These tutorials cover the basics of doing 2D stuff, like drawing text and images, as well as 2D animation, and some fancy effects in 2D. After you have completed these tutorials, you should be able to make some pretty interesting 2D games with XNA.
2D Tutorials

4 - 3D Tutorials

These tutorials should get you going with a 3D game. They cover things like drawing models, animation in 3D, and some simple effects, like lighting, and fog. These tutorials should really get you going on any 3D game.
3D Tutorials

5 - Input Tutorials

No game is complete without getting input from a user. When you get to the point in your game development where you are ready to get input from the user, take a look at these tutorials. They cover all sorts of input, including keyboard input, mouse input, and input from an Xbox controller.
Input Tutorials

6 - Audio Tutorials

Adding sound effects and background music to a game really makes the game come alive. These tutorials will teach you the basics (and some more advanced stuff) about playing all types of audio in your game. When you are ready to add audio to your game, come check out these tutorials.
Audio Tutorials

7 - Publishing Your Game Tutorials

When your game is complete, you will probably want to be able to give it to your friends, or even sell it. These tutorials will help you get started with the basics of publishing and distributing your game to others.
Publishing Your Game Tutorials

8 - Utility Tutorials

During the time that I have been working on these tutorials and teaching this stuff to students, I have come across a large variety of small random things that people occasionally like to do with their game. These tutorials are all small, and cover some random aspect of creating XNA games that you may find useful. At any point in your game development, you might want to check out these tutorials, which cover a broad variety of topics from creating games that run in full screen, to displaying the cursor, to changing the window size.
Utility Tutorials

9 - Content Pipeline Tutorials

XNA comes with a feature called the content pipeline, which manages all of your content, like 3D models, audio files, textures, images, and so on. The content pipeline is extensible, and you can add on anything you want to it, which is pretty nice. These tutorials cover more detailed information about how the content pipeline works, and how to create extensions for it. When you want to know more about the content pipeline, or how to extend it, come back to these tutorials.
Content Pipeline Tutorials

10 - Game Math Tutorials

Sometimes games require some fancy math to complete the game. These tutorials cover some of the more common math related problems that arise while making games.
Game Math Tutorials

11 - Game Physics Tutorials

Games can also require an understanding of physics in order to function well. These tutorials cover some of the more common physics related concepts that may come up while you are making your game, including collision detection.
Game Physics Tutorials

12 - 2D/3D Combination Tutorials

Once you have an understanding of 2D graphics and 3D graphics, you will likely want to combine them together. These tutorials will show you how to solve some of the problems that arise when you are doing this.
2D/3D Combination Tutorials

13 - Primitives Tutorials

Occasionally, when you are doing stuff with a 3D game, you want to be able to do more than just draw models that have been loaded in. XNA is built on top of DirectX, and so you have the ability to draw primitives, like lines and triangles (or lots of triangles to create interesting surfaces). Once you have an understanding of the basic 3D tutorials, try out these tutorials, which will go through the process of drawing with buffer objects, to draw all sorts of primitives.
Primitives Tutorials

14 - Effects & HLSL Tutorials

Today's graphics cards allow you to program them. The old way, the fixed function pipeline, allows you to only do specific effects while rendering. But now that you can program the graphics card, the possibilities are limitless. These tutorials teach you how to program the graphics card using a programming language called HLSL by using effects. Once you have a good understanding of the basic 3D tutorials, check these tutorials out.
Effects & HLSL Tutorials

15 - Advanced XNA Tutorials

Once you have completed the stuff in the 2D and 3D tutorial categories, you might want to look at this set of tutorials, which discuss some of the more advanced topics in XNA that aren't covered in one of the other categories.
Advanced XNA Tutorials

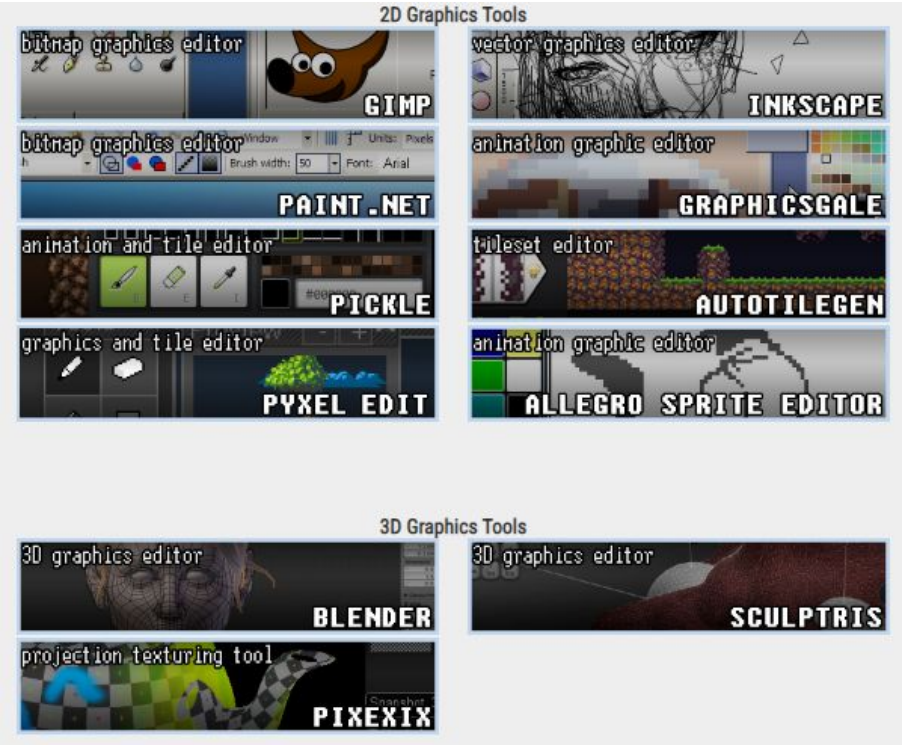
RB Whitaker's XNA Wiki

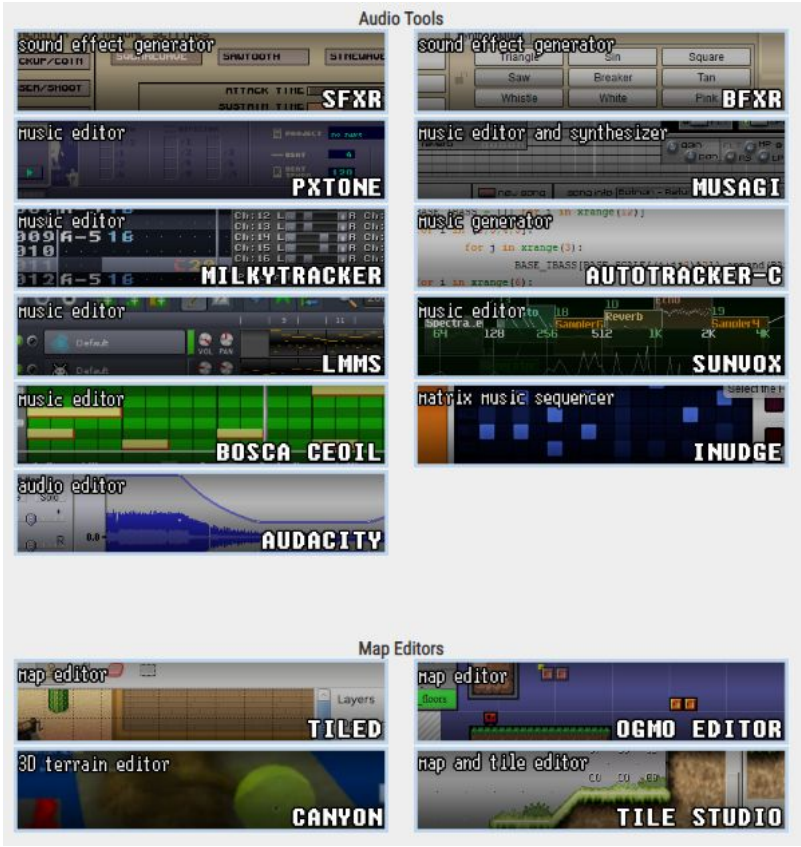
(http://rbwhitaker.wikidot.com/xna-tutorials)

Further Tools

Ludum Dare Tools

http://ludumdare.com/compo/tools/





Thank you!

Q?