CE318: Games Console Programming

Lecture 1: Introduction & XNA Basics

Philipp Rohlfshagen

prohlf@essex.ac.uk
Office 2.529

Outline

Outline

Course Overview

This course will teach you the fundamentals of games console programming. We will focus on 3D games, developed using Microsoft's XNA framework for the Xbox 360. Topics covered include:

- Structure of an XNA 4.0 game
- The game loop
- Game efficiency and optimisation
- Game content/assets
- Game design
- Game physics
- 2D/3D game implementations
- Simple game AI
- 2D and 3D graphics
- Vectors, matrices & triangles
- Texturing, shading, lighting
- 3D model creation
- Camera types
- Multiplayer games

The course is very practical, using numerous code samples throughout the lectures and encourages creative game design in the labs.

Learning Outcomes & Assessment

Following the completion of this module, you will be able to:

- Demonstrate an understanding of the architecture of 2D / 3D games
- Design and implement simple 2D / 3D games using XNA
- Implement basic AI behaviours for non-player characters
- Design and implement graphical effects in 3D
- Design and implement game objects (e.g., weapon systems)

The learning outcomes will be **assessed** by

- 2-hour exam in summer (70%)
- Single assignment (20%; due January 17, 2012)
- Progress test (10%; part of lecture in week 6)

Past exam papers and a sample progress test will be provided.

Contact Hours

There will be a 2-hour lecture every Thursday at 2pm in room 3.008.

- 2 x 50 minutes with break in-between
- Includes classroom discussions, software demos & exercises

Labs will take place every Friday at 9am and 12pm in CES Lab 5.

- You will be assigned to one lab session
- Step-by-step instructions, tutorials & open-ended exercises
- The teaching assistant is Diego Perez (`dperez@essex.ac.uk`)

	Day	Week	Time	Room
Lectures	Thursdays	2-11	2pm - 4pm	3.008
Labs	Fridays	2-11	9am - 11am, 12pm - 2pm	CES Lab 5

Finally, the progress test takes place in week 6 and there is a revision lecture in week 31. The assignment will be assessed in a lab session January 18, 2012.

Recommended Reading

Primary

1. XNA Game Studio 4.0 Programming by Miller and Johnson
2. Learning XNA 4.0 by Aaron Reed
3. C# Pocket Reference by Albaharai and Albahari
4. Artificial Intelligence for Games by Millington and Funge

Secondary

1. 3D Graphics with XNA Game Studio 4.0 by Sean James
2. XNA 4.0 Game Development by Example by Kurt Jaegers
3. Essential C# 4.0 by Mark Michaelis
4. Game Artificial Intelligence by Ahlquist and Novak

The course web-page is at

`http://courses.essex.ac.uk/ce/ce318`

where all course material will be made available over time.

Outline

Games Industry in the USA (ESA)

The video game industry has been growing steadily in the last decade despite economic unrest. In the **US**, it has been one of the fasted growing industries between 2005 and 2009:

- Growth rate $> 10\%$ (growth rate of U.S. economy: $< 2\%$)
- Direct employment grew at an annual rate of 8.6%
- US games industry employs 120000 people
- 2010: consumers spend USD 25.1b on games, hardware & accessories
- 72% of American households play computer or video games
- Average player is 37 years old and has been playing for 12 years
- Average age of the most frequent game purchaser is 41 years old
- 42% of all game players are women
- 29% of game players are over the age of 50
- 2010: most popular game genre was *Action* ($\sim 22\%$ of all games sold)

Doom: Start of a New Era

Doom's primary distinguishing feature at the time of its release was its realistic 3D graphics, then unparalleled by other real-time-rendered games running on consumer-level hardware.

from Wikipedia

Doom featured several novelties:

- Height differences
- Non-perpendicular walls
- Fully textured surfaces
- Varying light levels
- Custom palettes

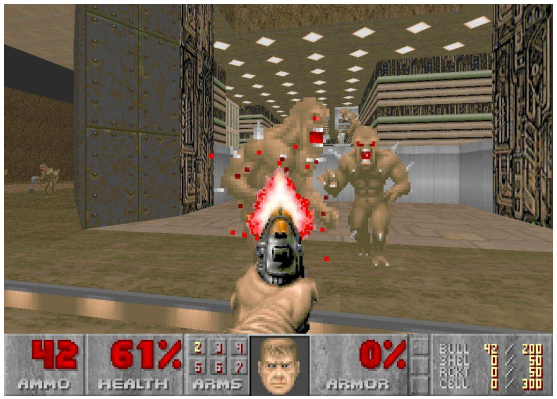
Development of 3D Graphics

Wolfenstein 3D

Released 1992

Doom

Released 1993

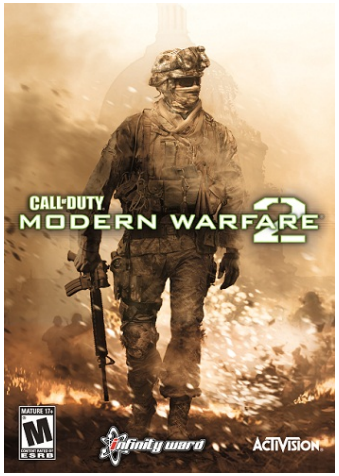


Rage

Released 2011

Case Study: COD MW 2 and GTA4

COD MW 2 released in 2010



Metacritic: 94%
Sold $>$ 20 million copies
Revenue of USD 1 billion
4.2 million units sold for Xbox 360

GTA4 released in 2008

Metacritic: 98%
Cost: USD 100 million
Involved $>$ 1000 developers
Sold $>$ 20 million copies

In 2008, Grand Theft Auto IV broke the world record for "Highest grossing video game in 24 hours" (Wikipedia).

Video Games Consoles: The Big Three

Nintendo Wii

Introduced 2006
Sold 88.2m units
44.6% marketshare

Sony PS3

Introduced 2006
Sold 56.3m units
28.5% marketshare

Microsoft Xbox 360

Introduced 2005
Sold 53.2m units
26.9% marketshare

But: big three for "serious gaming": PC, Xbox, PS3.

The handheld console market is dominated by Nintendo DS and Sony PSP (Vita to be released shortly).

Developing Console Video Games

Developing video games has become a major undertaking. The roles involved in developing 3D games include:

- Producer
- Publisher
- Graphic Artists
- Programmers
- (Level) Designers
- Sound Engineer
- Audio Specialists
- Actors / voice actors
- Testers

Unlike Sony and Nintendo, Microsoft allows non-certified developers to write games for their console:

- XNA Game Studio 4.0
- Visual Studio 2010
- C# 4.0
- XNA Game Studio Connect
- App Hub
- Xbox Live Community
- Xbox Live Indie Games

You will also have to use additional tools for textures, models and animations, such as Blender, Google SketchUp or Paint.NET.

The XNA Framework

> *XNA Game Studio 4.0 is an integrated development environment designed to make it easier to develop games for Microsoft Windows, Xbox 360, and Windows Phone. XNA Game Studio extends Microsoft Visual Studio with support for the XNA Game Studio Framework and tools. The XNA Game Studio Framework is a managed-code class library that contains features targeted specifically at game development. In addition, XNA Game Studio includes tools for adding graphic and audio content to your game.*
>
> *Microsoft*

In short:

> *XNA liberates developers from spending too much time writing mundane, repetitive boilerplate code.*
>
> *Microsoft*

The XNA Framework

Some of the libraries included with XNA are:

`Microsoft.Xna.Framework`
: General framework features, math, and game objects

`Microsoft.Xna.Framework.Graphics`
: All graphics features, including 2D and 3D

`Microsoft.Xna.Framework.Audio`
: All audio features

`Microsoft.Xna.Framework.Input`
: All input features, including game pads, keyboard and mouse

`Microsoft.Xna.Framework.Media`
: Media features for pictures, music, etc.

`Microsoft.Xna.Framework.Content`
: Content pipeline features (importing media and models)

XNA Profiles

XNA Game Studio 4.0 introduces the concept of XNA Framework profiles. A profile is a feature set that is implemented in hardware that allow you to create a clear development environment while targeting broad or specific platform releases.

Reach profile

- DirectX 9
- HLSL Shader Model 2.0
- Designed for compatibility
  - Windows Phone 7
  - Windows-based PCs
  - Xbox 360 consoles

HiDef profile

- DirectX 10
- HLSL Shader Model 3.0
- Can use platform showcase features
  - Xbox 360 consoles
  - Windows-based PCs

In short, the profiles determine what feature sets are available. One does no longer need to query the graphics card directly. You can set the profile in the solution's properties.

We do not need to worry too much about these (more details in lecture 10).

Visual Studio 2010: XNA Project Setup

How to set up a solution:

1. Open Visual Studio 2010
2. Click on *File* (in menu), then *New* and *Project ...*
3. A dialog box appears with installed templates
4. Select *XNA Game Studio 4.0* underneath the *Visual C#* node
5. A set of available projects appears in the right pane.
6. Select *Windows Game (4.0)*, supply a title for your project and click *OK*

Note: for convenience, we will develop Windows Games in most cases. In order to port a game to the Xbox, right-click on game and choose *Create copy of Project for Xbox 360* from the menu. Wired gamepads can be connected directly to the PC.

Outline

Introduction to C#

All XNA games are written in C# (pronounced c-sharp), an object-oriented programming language developed by Microsoft within its .NET framework. XNA Game Studio 4 makes use of C# 4.0 which was released in April, 2010.

`http://msdn.microsoft.com/en-us/vcsharp/aa336706`

1. Open Visual Studio 2010
2. Click on *File* (in menu), then *New* and *Project ...*
3. A dialog box appears with installed templates
4. Select *Windows* under the Visual C# node
5. Select *Console Application* from the templates available

Comment code: [CRTL+K], [CTRL+C] – Uncomment code: [CRTL+K], [CTRL+U]
Build the solution: [CTRL+F5] – Create a new class: [SHIFT + ALT + C]

The Basics of C#: From a Java Perspective

```
1  using System;
2
3  namespace HelloWorld
4  {
5    class Hello
6    {
7      static void Main(string[] args)
8      {
9        PrintHelloWorld();
10       Console.WriteLine("Press any key to exit.");
11       Console.ReadKey();
12     }
13
14     static void PrintHelloWorld()
15     {
16       Console.WriteLine("Hello World!");
17     }
18   }
19 }
```

The word `using` is used to import packages each of which is known as a `namespace`. You can have many classes in each namespace and the filename does not need to correspond to either the namespace nor any of the enclosed classes. Method names start with a capital letter, whereas the built-in types `class` and `string` do not.

The Basics of C#: Variables and Parameters

Variables and constants:

```
1 private int var1 = 100;
2 protected double var2 = 20d;
3 public bool var3 = true;
4 string var4 = "A string";
5 string var5 = null;
6 public const int con1 = 3;
```

`ref` keyword:

```
1 int a = 5; int b = 10;
2 Swap(ref a, ref b);
3
4 void Swap(ref int a, ref int b)
5 {
6   int tmp = a;
7   a = b;
8   b = tmp;
9 }
```

`out` keyword:

```
1 int a;
2 Out(out a)
3
4 void Out(out int a)
5 {
6   a = 10;
7 }
```

`var` keyword:

```
1 public void MyMethod()
2 {
3   var v = 20d;
4 }
```

Conversions (implicit and explicit):

```
1 static int i = 10;
2 static double d = i;
3 static short s = (short)d;
```

`params` keywords (like Java '...'):

```
1 void Pars(params int[] numbers)
```

Optional parameters:

```
1 void Opt(int a, int b = 10)
```

Named arguments

```
1 private void Name(int a, int b)
2
3 Name(1, 2);
4 Name(b: 2, a: 1);
```

The Basics of C#: Classes, Interfaces and Inheritance

Classes and interfaces are similar to Java. In C#, a class can only ever subclass one class at a time but may implement numerous interfaces.

A simple class:

```
1  class MyClass1
2  {
3    public int id;
4
5    public MyClass(int id)
6    {
7      this.id = id;
8    }
9
10   public int getID()
11   {
12     return id;
13   }
14 }
```

A simple interface (notice 'I'):

```
1 interface IMyInterface
2 {
3   void AMethod();
4 }
```

Class implements interface:

```
1 class MyClass2 : IMyInterface
2 {
3   public void AMethod()
4   {
5     // Do something
6   }
7 }
```

Can instantiate class with public fields:

```
1 C c = new C(N = "N", ID = 0);
2
3 class C
4 {
5   public string N;
6   public int ID;
7
8   public SomeClass () { }
9 }
```

Use `base` to access super class.

The Basics of C#: Namespaces

Namespace are like packages.
You can create a hierarchical organisation of your code by nesting namespaces.

```
1  namespace Outer.Middle.Inner
2  {
3    class Class1 {}
4    class Class2 {}
5  }
```

which is identical to

```
1  namespace Outer
2  {
3    namespace Middle
4    {
5      namespace Inner
6      {
7        class Class1 {}
8        class Class2 {}
9      }
10   }
11 }
```

You import namespaces using the `using` directive.

The Basics of C#: Properties

Properties are shortcuts for the getters and setters usually associated with class variables.

```
1 private String Name;
2
3 public String Name
4 {
5   get { return Name; }
6   set { Name = value; }
7 }
```

An even shorter version is:

```
1 public String Name { get; set; }
```

Now the class has a property called `Name` which can be set by calling

```
1 Class.Name="My Name";
```

Use access modifiers (e.g., `private`) to prevent anyone from changing the value of the property.

The Basics of C#: Structs

Like a class but a value type. Also, they do not support inheritance.

```
1  struct Square
2  {
3    public int Width { get; set; }
4    public int Height { get; set; }
5
6    public Rectangle(int width, int height)
7    {
8      Width = width;
9      Height = height;
10   }
11
12   public Rectangle Add(Rectangle rect)
13   {
14     Rectangle newRect = new Rectangle();
15
16     newRect.Width = Width + rect.Width;
17     newRect.Height = Height + rect.Height;
18
19     return newRect;
20   }
21 }
```

Default constructor initialises all variables to 0; you cannot assign a value to them (unless they are `const`). Structs do not require instantiation of an object on the heap.

The Basics of C#: Arrays & Lists

C# has two types of array: **rectangular** and **jagged**. In almost all cases, you will use rectangular arrays (more efficient).

```
1  int[] a = new int[3];
2  int[] b = new int[] { 4, 3, 8 };
3  int[] c = { 4, 3, 8 };
4
5  int[][] d = new int[3][];
6  d[0] = new int[3];
7  d[1] = new int[4];
8  d[2] = new int[1];
9
10 double[,] e = new double[2, 2];
11
12 int[,] f = new int[2, 2]
13 {
14   {0,1},
15   {2,3}
16 };
```

For individual dimensions:
Jagged arrays, use `.Length`
Rectangular arrays, use `getLength(DIM)`

```
1  List<string> l = new List<string>();
2
3  l.Add("one");
4  l.Add("two");
5
6  l.Insert(0, "three");
7  l.Sort();
8
9  foreach(String s in l)
10   Console.WriteLine(s);
```

List uses generics (like Java)

The Basics of C#: Enumerations

Enumerations are used frequently in games to indicate different states of the game (e.g., splash screen, paused, etc.):

```
1 public enum AnEnumeration { TYPE1, TYPE2, TYPE3, TYPE4 };
```

Enumerations are often used in conjunction with switch statements:

```
1  public void IllustrateEnumerations(AnEnumeration type)
2  {
3    switch (type)
4    {
5      case AnEnumeration.TYPE1:
6        Console.WriteLine("Type 1");
7      break;
8      case AnEnumeration.TYPE2:
9        Console.WriteLine("Type 2");
10     break;
11     case AnEnumeration.TYPE3:
12       Console.WriteLine("Type 3");
13     break;
14     case AnEnumeration.TYPE4:
15       Console.WriteLine("Type 4");
16     break;
17   }
18 }
```

Each `enum` member has an integral value (following declaration order).

The Basics of C#: Loops

C# has numerous iteration statements, similar to Java.

```
1  string[] a = new string[2];
2  a[0] = "one";
3  a[1] = "two";
4
5  for (int i = 0; i < a.GetLength(0); i++)
6    Console.WriteLine(a[i]);
7
8  foreach (string s in a)
9    Console.WriteLine(s);
10
11 int count = 0;
12
13 while (count < a.GetLength(0))
14   Console.WriteLine(a[count++]);
15
16 count = 0;
17
18 do
19 {
20   Console.WriteLine(a[count]);
21 }
22 while (++count < a.GetLength(0));
```

C# also has `break`, `continue` and `goto` statements.

The Basics of C#: Additional Concepts

Comments:

```
1 // Line comment
2
3 /* Multi-line
4  * comment
5  */
6
7 /// <sumary> XML </summary>
```

Directives

```
1 # if DEBUG
2   Console.WriteLine("Debug");
3 # else
4   Console.WriteLine("No debug");
5 # endif
```

```
1 # region NAME / DESCRIPTION
2   // Some code
3 # endregion
```

Often have: `# if WINDOWS || XBOX360`

Anonymus types:

```
1 var data =
2   new { Name = "SomeName", ID = 2}
```

Lambda Expressions:

```
1 int[] scores = { 90, 71, 82, 93 };
2 int highScoreCount = scores.Where(
3   n => n > 80).Count();
```

Generics:

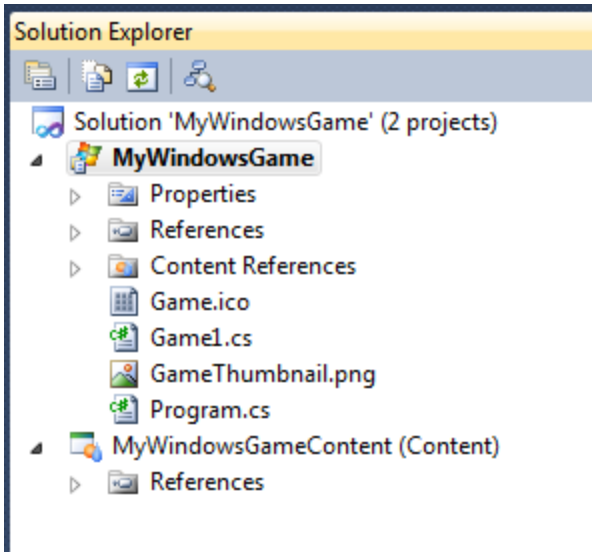
```
1 List<int> myInts = new List<int>();
2
3 Dictionary<int, string> customers
4   = new Dictionary<int, string>();
```

Outline

Back to XNA

When you create a new game project in VS2010, XNA sets up a **solution** that contains a **code project** and a **content project**.

- The code project is where your code goes
- A solution may contain numerous projects
- You can set the active project (right-click)
- Run in debug or build mode
- The content project holds all the assets



XNA adds 2 files to the project: `Game1` inherits from `Microsoft.XNA.Framework.Game` and provides the game loop. The class already contains the methods necessary to write a game. The second class `Program` executes the game.

Game1.cs Template (I)

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Microsoft.Xna.Framework;
5  using Microsoft.Xna.Framework.Audio;
6  using Microsoft.Xna.Framework.Content;
7  using Microsoft.Xna.Framework.GamerServices;
8  using Microsoft.Xna.Framework.Graphics;
9  using Microsoft.Xna.Framework.Input;
10 using Microsoft.Xna.Framework.Media;
11
12 namespace WindowsGame1
13 {
14   public class Game1 : Microsoft.Xna.Framework.Game
15   {
16     GraphicsDeviceManager graphics;
17     SpriteBatch spriteBatch;
18
19     public Game1()
20     {
21       graphics = new GraphicsDeviceManager(this);
22       Content.RootDirectory = "Content";
23     }
24
25     protected override void Initialize()
26     {
27       // TODO: Add your initialization logic here
28       base.Initialize();
29     }
```

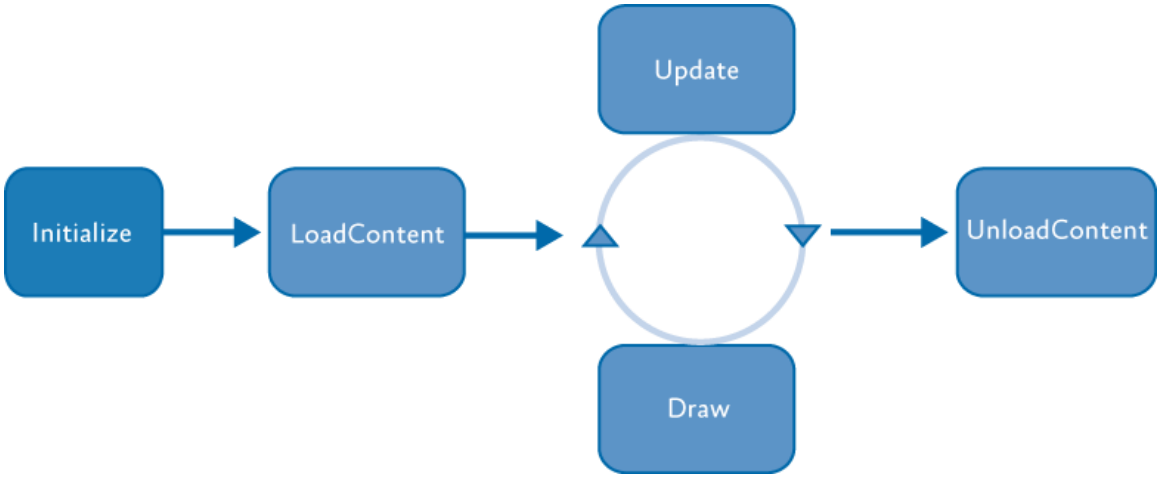
Game1.cs Template (II)

```
30      protected override void LoadContent()
31      {
32        spriteBatch = new SpriteBatch(GraphicsDevice);
33        // TODO: use this.Content to load your game content here
34      }
35
36      protected override void UnloadContent()
37      {
38        // TODO: Unload any non ContentManager content here
39      }
40
41      protected override void Update(GameTime gameTime)
42      {
43        if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
              ButtonState.Pressed)
44          this.Exit();
45
46        // TODO: Add your update logic here
47        base.Update(gameTime);
48      }
49
50      protected override void Draw(GameTime gameTime)
51      {
52        GraphicsDevice.Clear(Color.CornflowerBlue);
53        // TODO: Add your drawing code here
54        base.Draw(gameTime);
55      }
56    }
57  }
```

The Game Loop

In order to write a game, all one needs to do is to replace the `// TODO` comments with some game-specific code.

- `Constructor()` Constructor of the game
- `Initialize()` Initialise assets that do not require a GraphicsDevice
- `LoadContent()` Load game assets such as models and textures
- `UnloadContent()` Release game assets
- `Update()` Update the game's logic
- `Draw()` Render all game objects and backgrounds on the screen



Game Loop Timing

Following initialisation and loading content, XNA repeatedly calls the `Update()` and `Draw()` methods.

- A Game is either fixed step (default) or variable step
- The type of step determines how often `Update()` will be called
- A fixed-step Game tries to call `Update()` every `TargetElapsedTime`
- Default `TargetElapsedTime` is 1/60th of a second (60 fps)
- Game only calls `Update()` when its time but possibly skips call to `Draw()`
- If `Update()` takes too long, Game sets `IsRunningSlowly` to true
- You can check value of `IsRunningSlowly` in `Update()` to detect dropped frames
- You can reset the elapsed times by calling `ResetElapsedTime`

All games we deal with use fixed steps. Variable update steps and performance issues (frame rates) will be covered in lecture 10.

Game Loop Timing

XNA calls `Update()` and `Draw()` 60 times a second. This ensures there is no flickering on the monitor displaying the graphics. It is possible to change the frame rate in the constructor:

```
1 TargetElapsedTime = new TimeSpan(0, 0, 0, 0, 50);
```

This forces XNA to call `Update()` only very 50 milliseconds (i.e., 20 fps).

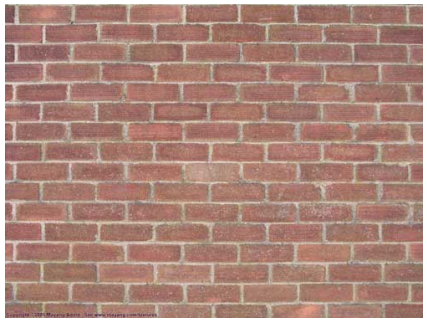
Many objects will depend on the number of times `Update()` is called. Can use *animation frames*:

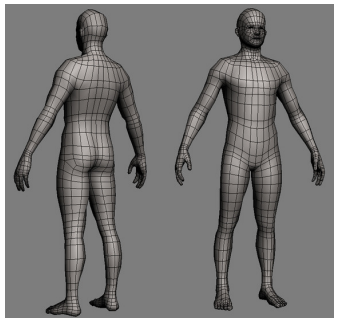
```
1 int timeSinceLastFrame = 0;
2 int millisecondsPerFrame = 50;
```

These variables can be used to update an object only if `millisecondsPerFrame` have passed. You can get the time elapsed in between calls to `Update()` using `gameTime.ElapsedGameTime.Milliseconds`.

The Content Pipeline

Any video game relies heavily on **assets** such as textures, sprites, 3D models, fonts, sounds or even complex structures such as game levels. Most content will be created using a digital content creation (DCC) tools and XNA provides a convenient way to integrate assets into your game.





The XNA Game Studio Content Pipeline is a set of processes applied to the assets at build time to produce data that can be retrieved and used within an XNA Game Studio game through the XNA Framework Class Library. Assets are processed into a managed code object and serialised to a file that is included in the games executable.

The Content Pipeline

XNA's content pipeline is very efficient and provides several benefits:

- Game artists can use the DCC tools of their choice
- Game developers can use files produced by game artists directly
- If file format is supported, can use Standard Importers and Processors
- Content Pipeline can easily be customised to import a new file formats
- Custom content processors can produce novel assets from existing ones

XNA Game Studio supplies standard importers and processors for many popular DCC file formats:

- .x, .fbx (meshes), .fx (effects), .spritefont (fonts)
- .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga (textures)

Assets may be used by the game via the `ContentManager.Load()` method of the XNA Framework.

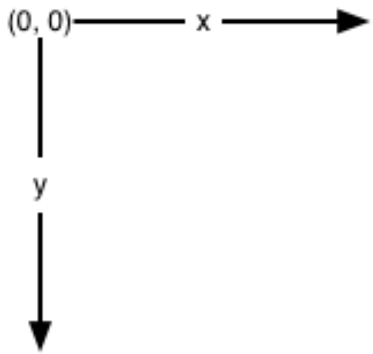
Outline

2D Games

The majority of "serious" games is 3D but there is still demand for 2D games: many casual "mini-games" are in 2D as well as most games on mobile platforms.

Developing some simple 2D games lays a useful foundation upon which to build more complex 3D games. In particular, the following aspects apply to all types of games:

- Game design
- (Most) Game logic
- Game loop & speed
- User inputs
- Sounds



Finally, almost all 3D games need to draw 2D graphics at some point.

Drawing Sprites

The term *sprite* is used for images drawn on the screen. In `Game1`, the following lines are included in the class definition:

```
1 GraphicsDeviceManager graphics;
2 SpriteBatch spriteBatch;
```

The `GraphicsDeviceManager` allows access to the graphics device `GraphicsDevice` and the `SpriteBatch` does the actual drawing of the sprite.

Running `Game1` creates a frame with blue background:

```
1 GraphicsDevice.Clear(Color.CornflowerBlue);
```

In order to display an image, we need to

1. Add the image to the game's content pipeline
2. Load image into game
3. Draw the image

Drawing Sprites

Add the image:

1. Right-click on the content project header
2. Select *Add* - *Existing Item ...*
3. Choose file

We then add a variable for the image in `Game1`:

```
1 Texture2D image;
```

and load the image (in `Game1.LoadContent()`):

```
1 image = Content.Load<Texture2D>("image")
```

where "image" is the name of the file (extension is not required).

Finally, draw the image (in `Game1.Draw()`):

```
1 spriteBatch.begin();
2 spriteBatch.Draw(image, GraphicsDevice.Viewport.Bounds, Color.White);
3 spriteBatch.end();
```

Drawing Sprites

The `SpriteBatch` has 7 `Draw()` methods (overloads). We just used:

```
1 SpriteBatch.Draw (Texture2D, Rectangle, Color)
```

where

- `Texture2D` is the sprite to be drawn
- `Rectangle` is the output rectangle (image will be stretched)
- `Color` is the tinting colour (white does nothing)

```
1 SpriteBatch.Draw (Texture2D, Vector2, Color)
```

Draws the image at point `Vector2` (original size).

```
1 SpriteBatch.Draw (Texture2D, Rectangle, Nullable<Rectangle>, Color)
```

Specifies a source rectangle and a destination rectangle. Useful for animations.

`SpriteBatch.Draw()` Arguments

Texture: `Texture2D`
: The texture to be drawn

Position: `Vector2`
: The coordinate for the upper-left corner of the drawn image

SourceRectangle: `Rectangle`
: Allows you to draw only a portion of the source image

Color: `Color`
: The tinting color

Rotation: `float`
: Rotates the image

Origin: `Vector2`
: Indicates an origin around which to rotate

Scale: `float`
: Scales the image

Effects: `SpriteEffects`
: Uses the SpriteEffects enum to flip the image horizontally or vertically

LayerDepth: `float`
: Allows you to specify which images are on top of other images

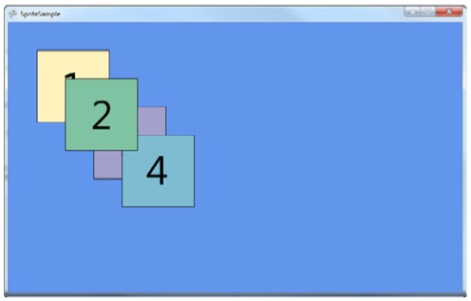
Displaying Multiple Sprites

The call to `SpriteBatch.Begin()` also allows for parameters:

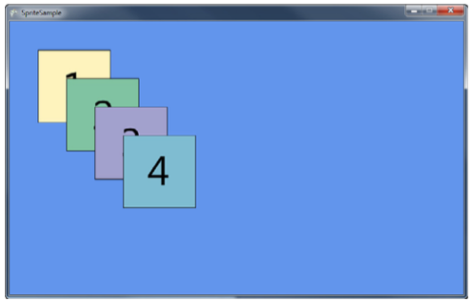
- `SpriteSortMode`: `Deferred`, `Immediate`, `Texture`, `BackToFront`, `FrontToBack`
- `BlendState`: `AlphaBlend`, `Additive`, `NonPremultiplied`, `Opaque`

For instance:

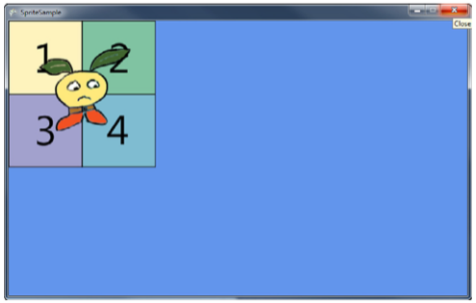
- `SpriteBatch.Begin(SpriteSortMode.Texture, null)`
- `SpriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend)`



no control



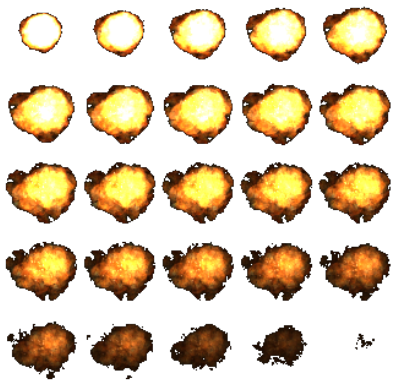
layer control



alpha blending

2D Animations

A simple way to generate animated sprites is to use **sprite sheets**.



- A sequence of images on one sheet
- Images created in some DCC package
- Use source rectangle to choose image
- Move source rectangle across sheet
- Creates a moving image

```
1 Point frameSize = new Point(50, 50);  // individual image size
2 Point currentFrame = new Point(0, 0); // start top left
3 Point sheetSize = new Point(5, 5);    // rows and columns
```

In `Game1.Update()`, loop over sprite sheet, then draw:

```
1 spriteBatch.Begin();
2 spriteBatch.Draw(texture, Vector2.Zero, new Rectangle(currentFrame.X *
      frameSize.X, currentFrame.Y * frameSize.Y, frameSize.X,
     frameSize.Y), Color.White, 0, Vector2.Zero, 1, SpriteEffects.None
     , 0);
3 spriteBatch.End();
```

Texts and Fonts

In order to draw text on the screen, a font is required:

1. Add a new item to content project:
   - Right-click: *Add*, *New Item ...* and choose `SpriteFont`
2. Enter a name for the font and click *Add*
3. Edit XML file to suit preferences: font type, size, spacing, etc.
4. Create a `SpriteFont` variable in `Game1` (`font`, say)
5. Load font: `font = Content.Load<SpriteFont>("MyFont");`

You can then draw text on the screen using that font:

```
1 spriteBatch.Begin();
2 spriteBatch.DrawString(font, "Hello!", Vector2.Zero, Color.White);
3 spriteBatch.End();
```

Often useful to know size of string:

```
1 Vector2 stringSize = font.MeasureString("Hello!");
```

Can then use attributes like `stringSize.Y` to place string on screen.

Sounds

Sounds can be very easy to integrate into an XNA game. One can use the standard `SoundEffect` class or the Cross-Platform Audio Creation Tool (XACT; lecture 4).

There are two ways to use `SoundEffect`:

- File and Forget (which we will use)
- Making use of `SoundEffectInstance`

Add a sound file (e.g., mp3) to your content project and add it to `Game1`:

```
1 SoundEffect soundEffect;
```

In `Game1.LoadContent()`:

```
1 soundEffect = Content.Load<SoundEffect>("mySound");
```

Then do something with the sound:

```
1 soundEffect.Play();
2 soundEffect.Play(1.0f, 0.0f, 0.0f); // volume, pitch and panning
```

Note: if VS2010 shows an error, make sure to set the asset's content processor to *Sound Effect*.

SquareChase Game

We will now create a (very) simple Windows game based on the sample by Jaeger (pp 12-21). In this game, the player has to click a randomly moving square for points. First, create an 16x16 white square ("square.bmp") in some graphics program and add it to the content pipeline.

In `Game1`:

```
1 Random rand = new Random();
2 Texture2D squareTexture;
3 Rectangle currentSquare;
4 int playerScore = 0;
5 float timeRemaining = 0.0f;
6 const float TimePerSquare = 0.75f;
7 Color[] colors = new Color[3] {Color.Red, Color.Green, Color.Blue};
```

In `Initialize()`:

```
1 this.IsMouseVisible = true;
```

In `LoadContent()`:

```
1 squareTexture = Content.Load<Texture2D>("square");
```

SquareChase Game

In `Update()`:

```
1  if (timeRemaining == 0.0f)
2  {
3    currentSquare = new Rectangle(
4      rand.Next(0, this.Window.ClientBounds.Width - 25),
5      rand.Next(0, this.Window.ClientBounds.Height - 25), 25, 25);
6    timeRemaining = TimePerSquare;
7  }
8
9  MouseState mouse = Mouse.GetState();
10
11 if ((mouse.LeftButton == ButtonState.Pressed) &&
12     (currentSquare.Contains(mouse.X, mouse.Y)))
13 {
14   playerScore++;
15   timeRemaining = 0.0f;
16 }
17
18 timeRemaining = MathHelper.Max(0, timeRemaining -
19   (float)gameTime.ElapsedGameTime.TotalSeconds);
20 this.Window.Title = "Score : " + playerScore.ToString();
```

In `Draw()`:

```
1  spriteBatch.Begin();
2  spriteBatch.Draw(squareTexture, currentSquare,
3    colors[playerScore % 3]);
4  spriteBatch.End();
```

Deployment to Xbox

In order to deploy games to the XBox, an App Hub membership is required. This usually costs USD 99 but students are able to obtain a free 12-months membership through DreamSpark.

1. Create Microsoft Live account (formerly hotmail)
2. Register at `www.dreamspark.com`
3. Verify student status on DreamSpark via university
4. Select XNA App Hub membership from list of products

5. Create account on Xbox with verified email address
6. Get free Xbox Live Silver membership
7. Install XNA Game Studio Connect on Xbox (Xbox Live marketplace)
8. Connect Xbox to PC using the XNA Game Studio Device Center
9. Use 5x5 code to authenticate connection

You are now able to deploy games to the Xbox. If you wrote your game as a Windows game, simply convert it.

Deployment to Xbox

When you create an Xbox copy of your Windows game, XNA creates a new code project for the Xbox which shares the files with the original project. As such, any changes to any of the files will affect both projects.

To account for the differences in hardware and input devices, it is possible to simply add new code for different inputs – XNA will ignore them if they are unavailable. In case you want to restrict certain code to either platform, you can use directives:

```
#if(!XBOX360)
  // Do something if not running on Xbox
#endif
```

```
#if(WINDOWS)
  // Do something if running on PC
#endif
```

There are some additional subtleties one needs to account for to ensure that Xbox games run smoothly. We will cover those in lecture 10.

Outline

Summary

In this lecture you have been introduced to Microsoft's XNA framework. XNA allows game developers to focus on the creative side of game development without the need to worry about repetitive boilerplate code.

We covered

- XNA framework & profiles
- Visual Studio 2010
- The game loop
- The content pipeline
- C#
- Setting up the Xbox
- Sprites
- 2D animations
- Text and fonts
- Sounds
- A simple 2D game

Lab Preview

In the lab you will

- In the first part of the lab, you will write some simple C# code to familiarise yourself with the language. This will include classes, interfaces, inheritance, arrays, structs, enumerations, generics, control statements and a selection of data structures.
- In the second part of the lab, you will substantially extend the SquareChase game by adding animations, sounds and a splash screen. Instructions will be provided. Once finished, you are encouraged to be creative by adding your own ideas to the game.

Next lecture: code organisation, user input and a complete 2D game.