

Coding for Game Development

Section One

XNA Game Programming

Introduction to the Module

1/17

Module Tutors

- Bob Lang (bob.lang@uwe.ac.uk)
Module Leader
Semester one lectures
- Sophie Pink (sophie.pink@uwe.ac.uk)
Semester one and two practical classes
Semester two lectures

2/17

Text Books

- There are **no** required books for this module, however a number of books are available for extra reading.
- Because XNA is developing rapidly, books go out of date very quickly.
- If you find a book you like, please tell us!

3/17

Running XNA at Home

- To run XNA on your home computers, you will need to download and install:
 - Windows 7 operating system (Ideally)
 - Visual C# Express 2010
 - Microsoft XNA Game Studio 4
- As a student you can sign up for Microsoft Developer Academic Alliance (MSDNAA). This gives you *free* downloads of all these products.
- NOTE: Only XNA 4 is acceptable for coursework submissions. If you have installed an earlier version then you *must* update now.

4/17

Lecture Outline Semester One

2D Games Programming

- Section 1: Introduction
- Section 2: Basic Sprite Handling
- Section 3: Game Input Devices
- Section 4 - 5: Object Orientation in C#
- Section 6: Finite State Machines
- Section 7 - 8: Object Oriented Sprite Handling
- Section 9: State Driven Sprites
- Section 10: Backgrounds, Fonts and Collisions

5/17

Practical Work Semester One

- The first few weeks there will be a series of worksheets to help you to get to grips with the material.
- Later in the semester there will be an individual assignment.
- **YOU ARE EXPECTED TO ATTEND EVERY CLASS.**
- From previous years, skipping classes leads to failing the module and now you only have one chance at referral.

6/17

Practical Examination

- The examination for this module will be computer based.
- You will be given partially complete C#/XNA projects that you should finish and get working in the time available.
- You will only pass the exam if you have spent a lot of time using Visual Studio/XNA Game Studio

7/17

The XNA Game Loop

- **Game** is the top level class of an XNA game. This class provides a game loop and controls its timing:
 - By default, XNA uses a fixed timing interval.
 - The timing interval is set by the `TargetElapsedTime` property. By default this is set to 1/60 sec.
 - The game loop calls the method **Update()** once per iteration. All game logic should be triggered in this method.
 - The loop also calls the method **Draw()** once per iteration. All screen updates should be placed in this method.
 - When there is a processing bottleneck, the game loop will omit calls to **Draw()** whilst maintaining calls to **Update()**.

8/17

Game Loop Logic

- The XNA loop takes this approximate form:

```
Game constructor
Initialize()
while ( ...game starting or monitor setup changed... )
{
    LoadGraphicContent();
    while ( ...game running and monitor setup unchanged... )
    {
        ...pause to start of next time frame...
        Update();
        if ( ... there are no timing issues... )
        {
            Draw();
        }
    }
    UnloadGraphicContent()
}
Dispose()
```

methods we must write

9/17

Boilerplate Code

- XNA Game Studio emits standard *boilerplate* code for each different project type. This code gives the programmer a head start in getting coding under way.
- The boilerplate code for a game includes skeletal forms of methods such as **Update()** and **Draw()** which we can edit to implement our games.
- The boilerplate methods contain:
 - *TODO* flags which we replace by our customizing code.
 - Standard code which must be left alone, otherwise we may disrupt the proper operation of the game.

10/17

A Simple Sprite Example

- We start by loading the sprite image file into XNA:
 - Right-click on Content section of Solution Explorer
 - Add > Existing Item...
 - Select file from chooser.
- A typical sprite file would be a PNG image whose size is up to 120x120 pixels.
- We use PNG because it supports alpha transparency.
- For some games, the file might be a *sprite sheet* containing multiple versions of the image.

11/17

Declaring the Sprite in C#

- Backgrounds and sprites are declared using the XNA class **Texture2D**.
- Usually, each image is declared as a field at the top of the **Game1** class:

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    Texture2D balloon; // Sprite graphic
    int balloonY; // Sprite's Y position
    ::::
}
```

Standard boilerplate

Fields added by programmer

12/17

The LoadContent() Method

- The image file must be explicitly loaded into the game using the **Load<Texture2D>()** method.
- The call must be put in the **LoadContent()** method:

```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
    balloon = this.Content.Load<Texture2D>("Balloon");
}
```

Texture2D field Method to load graphics File name (no extension) Boilerplate

13 / 17

The Update() Method

- The **Update()** method is where we put our code to update the game objects.
- The **Update()** method runs every 1/60 second.

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (... == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    balloonY--; // Move sprite up one pixel

    base.Update(gameTime);
}
```

Standard boilerplate

14 / 17

The Draw() Method

- We must draw the sprite in the **Draw()** method:

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();

    Vector2 position = new Vector2(300, balloonY);
    spriteBatch.Draw(balloon, position, Color.White);

    spriteBatch.End();

    base.Draw(gameTime);
}
```

Boilerplate

Programmer code

Boilerplate

15 / 17

Sprites and Background

- XNA does not make any separation between sprites, backgrounds, or any other graphical asset.
- All graphics must be loaded into the program as Texture2D objects.
- All graphics must be drawn in the correct order in the Draw() method:

```
SpriteBatch.Begin();
SpriteBatch.Draw(..background image..);
SpriteBatch.Draw(..sprite one..);
SpriteBatch.Draw(..sprite two..);
... more sprites and graphics ...
SpriteBatch.End();
```

Drawing order:
back to front

16 / 17

Naïve Sprite Handling

- It's very easy to create a small XNA program using this technique.
- As we add more sprites, we also need to add more fields for each of their parameters:
 - X-Y location; X-Y Velocity, Rotation, Scaling, etc...
- Very soon, we have an explosion of complexity because this naïve approach leads to poor design.
- The best way to tame this complexity is to combine two design techniques:
 - Finite state machine design.
 - Object orientated sprite handling.

17 / 17

18 / 17