University of Hull

MonoGame and XNA in Teaching at Hull

Rob Miles

Agenda

- Computer Science Teaching at Hull
- Using XNA to Teach
- XNA Overview
- XNA and MonoGame
- Targeting Platforms

Computer Science Teaching at Hull

Computer Science at Hull

- We teach C# as a first language
- We move into C++ and other languages in the Second Year
- You can have our course for free if you like:

www.csharpcourse.com

C# Teaching Structure

- We teach algorithms and programming in the first semester and then move into objects in the second semester
- We've found that this works well, in that it keeps the focus on problem solving in the early stages

C# and XNA

- XNA is a framework for game creation
- It is provided as a set of libraries and a content management system
- It allows you to quickly create 2D sprite based games and can also be used as the basis of more ambitious work
- You can even write shaders if you want

Using XNA to Teach

Using XNA in the course

- We have found XNA useful for a number of reasons
  - It is a well thought out object framework that serves as a nice exemplar
  - It is very motivating for the students to be able to write games of their own
  - It provides a ready path to market (which quite a few students follow)

Teaching with XNA

- We spend one week of the second semester on XNA development
- This introduces the LoadContent/Draw/Update behaviours and the fundamentals of game development
- We also set a lab and coursework on XNA

Having Fun with XNA

- We use XNA as the basis of our “Three Thing Game” 24 hour game development events
- We have also promoted it as the basis of MonoGame team entries

XNA Overview

HOW GAMES WORK

- Every game that has ever been written has these fundamental behaviours:
- Initialise all the resources at the start
  - fetch all textures, models, scripts etc
- Repeatedly run the game loop:
  - Update the game world
    - read the controllers, update the state and position of game elements
  - Draw the game world
    - render the game elements on the viewing device

METHODS IN AN XNA GAME

- The XNA Game class contains methods that will provide these behaviours
- Initialise all the resources at the start
  - The `Initialize` and `LoadContent` methods
- Repeatedly run the game loop:
  - Update the game world
    - The `Update` method
  - Draw the game world
    - The `Draw` method

GETTING STARTED WITH XNA

- When you create a new XNA game project you are provided with empty versions of the game methods
- Creating an XNA game is a matter of filling in these methods to get the game behaviours that are required
- We are going to start by getting some clouds moving around the display
- Then we are going to add some complication and see where it gets us

Cloud and Games

- Apparently the future of computing is “in the cloud”
- Perhaps the future of games is too
- We can start with a drawing of a cloud and see where this takes us
- For me this is the fun of making games

Creating a Game World

```
// Game World
Texture2D cloudTexture;
Vector2 cloudPosition;
```

- A game needs a "Game World" which holds all the objects in the game
  - `LoadContent` will put content into them
  - `Update` will update their state
  - `Draw` will draw them
- To start with our game just contains a cloud texture and position

Loading the Cloud Texture

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    cloudPosition = new Vector2(0, 0);
    cloudTexture = Content.Load<Texture2D>("Cloud");
}
```

- This code loads our cloud texture
- It also sets the draw position for the cloud
- It also makes a `SpriteBatch`, which is used by XNA to batch up drawing operations

MonoGame Digression

- When we use MonoGame in Visual Studio 2012 we can't actually process the textures directly in the project
- We have to create the assets using an XNA project running under Visual Studio 2010 and copy them in
- More on this later

Drawing the Cloud Texture

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    spriteBatch.Begin();
    spriteBatch.Draw(cloudTexture, cloudPosition,
                                   Color.White);
    spriteBatch.End();
    base.Draw(gameTime);
}
```

- This code uses the `spriteBatch` to draw our cloud
- It also draws a blue sky as a background

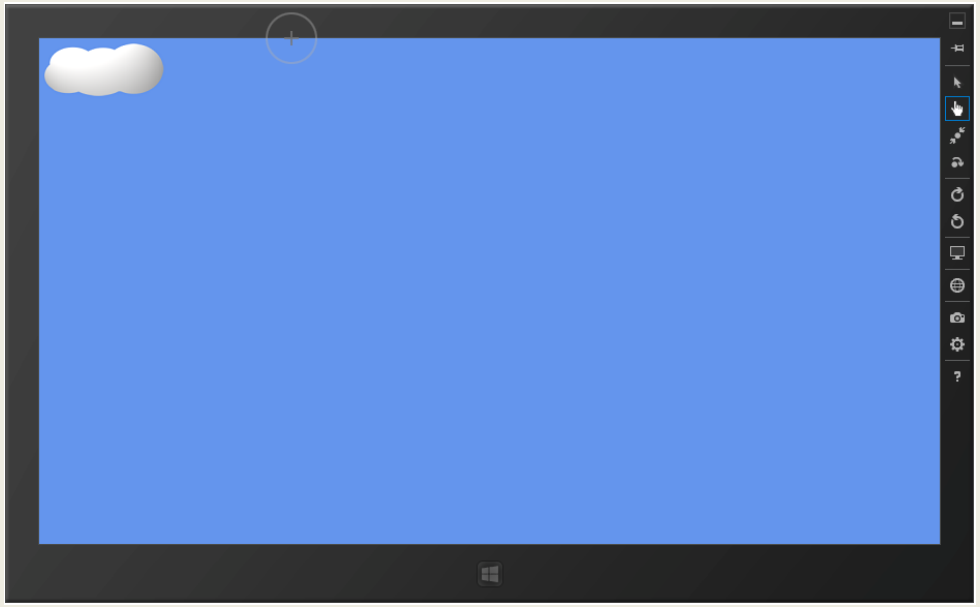
Drifting our Cloud

```
Vector2 cloudSpeed = new Vector2(1.5f, 0);

protected override void Update(GameTime gameTime)
{
    cloudPosition += cloudSpeed;

    base.Update(gameTime);
}
```

- The `Update` method is called 60 times a second
- We can use it to drift the cloud across the screen

1: Drifting Cloud

Rob Miles
University of Hull



demo

Creating Game Components

```
interface ISprite
{
    void Draw(CloudGame game);
    void Update(CloudGame game);
}
// Game World
List<ISprite> gameSprites = new List<ISprite>();
```

- We really need more than one cloud
- We can make a `ISprite` component which has `Draw` and `Update` behaviours
- We can make a list of these to use in our game

The Cloud Class

```
class Cloud : CloudGame.ISprite
{
    public Texture2D CloudTexture;
    public Vector2 CloudPosition;
    public Vector2 CloudSpeed;

    public void Draw(CloudGame game) ...

    public void Update(CloudGame game) ...

    public Cloud(Texture2D inTexture, Vector2 inPosition,
                 Vector2 inSpeed) ...
}
```

Making Random Clouds

```
Vector2 position =
    new Vector2(rand.Next(GraphicsDevice.Viewport.Width),
                rand.Next(GraphicsDevice.Viewport.Height));

Vector2 speed = new Vector2(rand.Next(0, 100) / 100f, 0);

Cloud c = new Cloud( cloudTexture, position, speed);

gameSprites.Add(c);
```

- The `Cloud` class has a constructor that takes a texture, position and speed and creates a `Cloud` instance
- This is then added to the sprites for this game

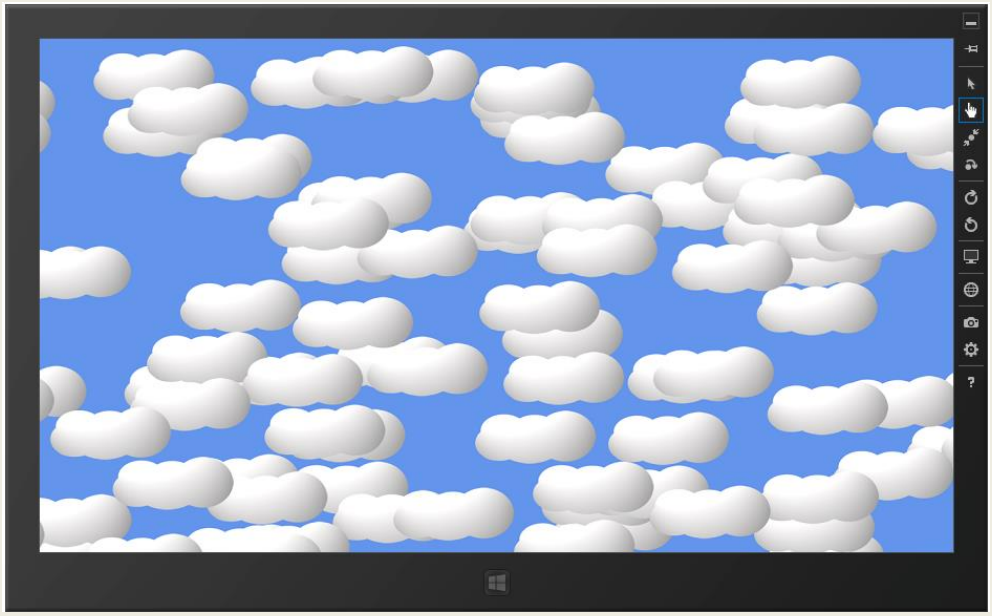
Updating Sprites

```
protected override void Update(GameTime gameTime)
{
    foreach (ISprite sprite in gameSprites)
        sprite.Update(this);

    base.Update(gameTime);
}
```

- The `Update` behaviour works through each game component and updates it
- The component is given a reference to the game so that it can affect the game state if required (e.g. update the score)
- There is a similar loop for `Draw`

2: Lots of Clouds

Rob Miles
University of Hull



demo

Cloud Bursting Game

- We could add a simple game mechanic so that players can burst the clouds by touching them
- To do this our game must detect touch events This is very easy to do
- Each cloud can check during its `Update` method to see if it has been burst

Getting Touch Locations

```
TouchCollection Touches;

protected override void Update(GameTime gameTime)
{
    Touches = TouchPanel.GetState();

    foreach (ISprite sprite in gameSprites)
        sprite.Update(this);
}
```

- The Touch Panel provides a method called `GetState` that supplies a collection of `TouchLocation` values that describe the current touch status

Using Touch Locations

```
foreach (TouchLocation touch in game.Touches)
{
    if (touch.State == TouchLocationState.Pressed) {
        if ( CloudContains(touch.Position) ) {
            CloudPopSound.Play();
            Burst = true;
            return;
        }
    }
}
```

- A program can get the status and location of the touch event and then burst the cloud that contains that location

Cloud Contains method

```
public bool CloudContains(Vector2 pos)
{
  if (pos.X < CloudPosition.X) return false;
  if (pos.X > (CloudPosition.X + CloudTexture.Width)) return false;
  if (pos.Y < CloudPosition.Y) return false;
  if (pos.Y > (CloudPosition.Y + CloudTexture.Height)) return false;
    return true;
}
```

- This method returns `true` if the cloud region contains a particular point

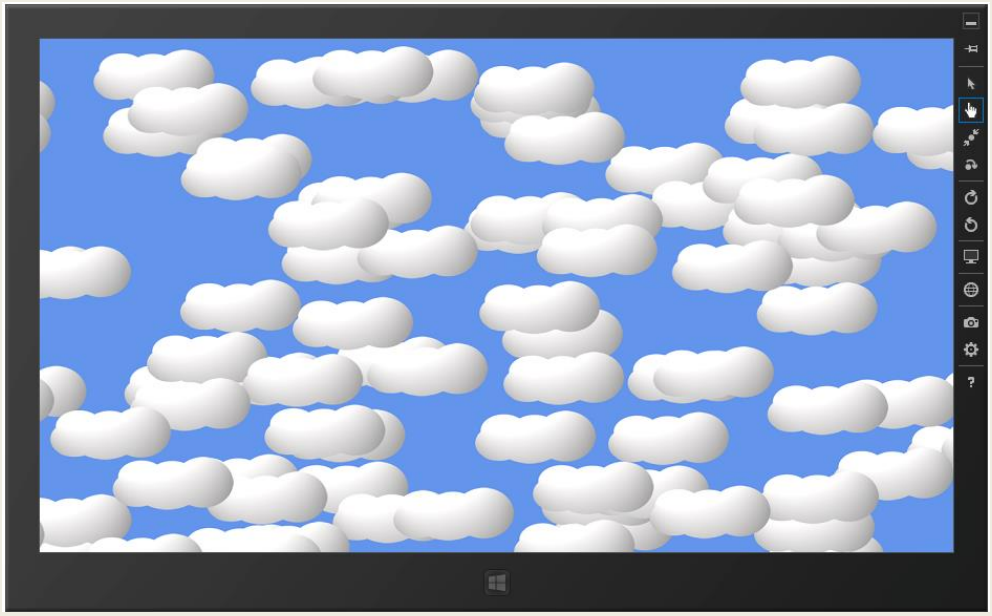
Pop Sound Effect

```
public SoundEffect CloudPopSound;
...
SoundEffect CloudPopSound =
Content.Load<SoundEffect>("Pop");
...
CloudPopSound.Play();
```

- Sound effects are loaded from WAV files which are added as content alongside image textures
- We can then play them on demand
- We can adjust pitch, location and volume of effects

3: Bursting Clouds

Rob Miles
University of Hull



demo

Touch States

- An active touch location can occupy one of a number of states:
  - TouchState.Pressed
  - TouchState.Moved
  - TouchState.Released
- Each touch location also has a unique ID so that it can be identified throughout its lifecycle
- This gives very good control of touch events at a very low level

Flicking Clouds using Gestures

- The touch panel also provides gesture support
- We can use the gestures to allow the player to flick clouds around the sky
- The game program can register an interest in gesture events
- These are then tracked and buffered by the touch panel
- Each gesture gives a particular set of information

Registering for a Gesture

```
TouchPanel.EnabledGestures = GestureType.Flick;
```

- The Flick gesture lets a game detect the direction and speed of a flick on the phone screen
- There are other gesture types:
  - Tap
  - DoubleTap
  - Hold
  - VerticalDrag
  - HorizontalDrag
  - Flick
  - Pinch

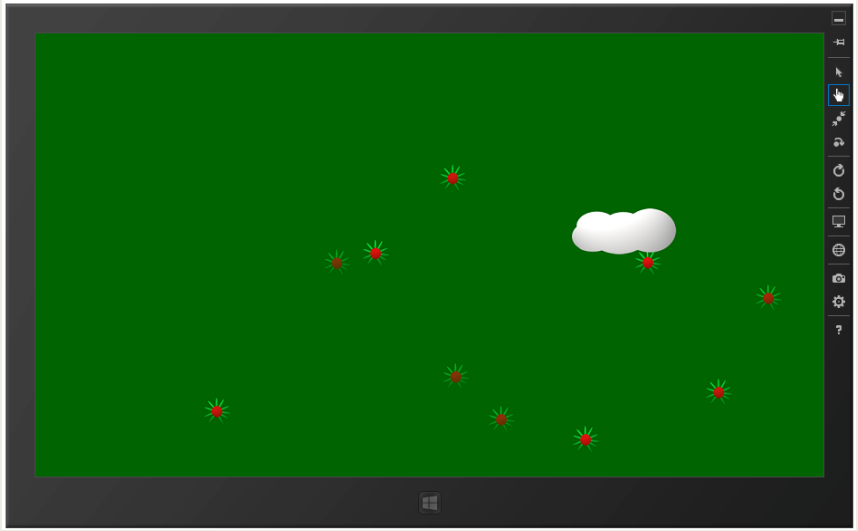
Using a Gesture

```
while (TouchPanel.IsGestureAvailable)
{
    GestureSample gesture = TouchPanel.ReadGesture();

    if (gesture.GestureType == GestureType.Flick)
    {
        waterCloud.CloudSpeed = gesture.Delta / 100f;
    }
}
```

- This code uses the Delta value of the flick gesture to allow players to flick the cloud towards the plant and water it

4: Flicking Clouds

Rob Miles
University of Hull



demo

Using the Accelerometer

- You can also use the accelerometer to control the behaviour of objects in your game
- The Accelerometer can measure acceleration in X, Y and Z
- You can use just the X and Y values to turn it into a replacement for a gamepad
- The values that are returned are in the same range
  - -1 to +1 in each axis
- A value of 1 means 1G (G is the acceleration due to gravity)

Accelerometer Events

- The accelerometer in XNA is event driven
- It generates events when new readings are available
- You must bind a method to the event
- The method can store the settings for later use
- The code in an `Update` method can read the settings and use them to update the position of game objects

Starting the Accelerometer

```
Accelerometer accel;
void startAccelerometer()
{
  accel = Accelerometer.GetDefault();

  if (accel != null)
   accel.ReadingChanged += accel_ReadingChanged;
}
```

- This sets up accelerometer and binds to the reading changed event

Accelerometer Event Handler

```
void accel_ReadingChanged(Accelerometer sender,
                AccelerometerReadingChangedEventArgs args)
{
    lock (this)
    {
        accelReading.X = (float)args.Reading.AccelerationX;
        accelReading.Y = (float)args.Reading.AccelerationY;
        accelReading.Z = (float)args.Reading.AccelerationZ;
    }
}
```

- When we get a new reading we copy this into a vector
- We use a lock so that this process is never interrupted

Reading the Accelerometer

```
Vector3 getAccReading()
{
    Vector3 result;

    lock (this) {
        result = new Vector3(accelReading.X, accelReading.Y,
                                              accelReading.Z);
    }
    return result;
}
```

- This method returns the most up to date reading

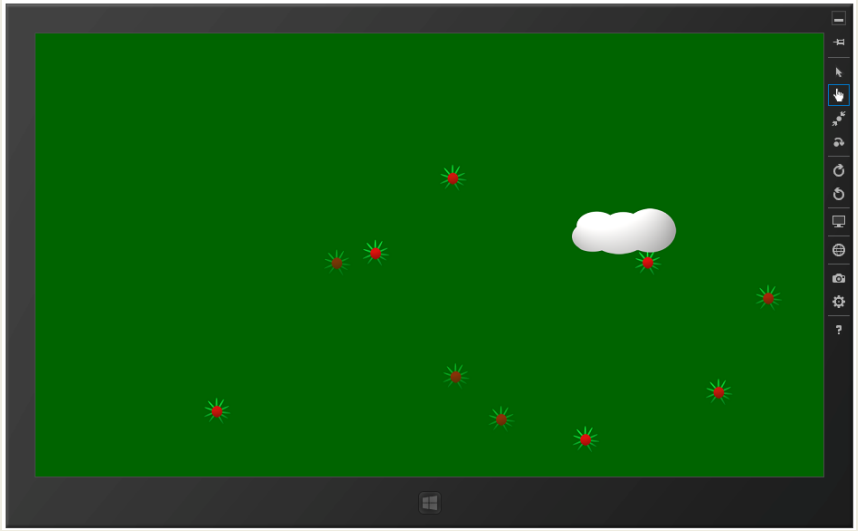
Acceleration and Physics

```
Vector3 acceleration = game.getAccReading();

CloudSpeed.X += (acceleration.X * 0.1f);

CloudPosition += CloudSpeed;
```

- This is a very simple physics model
- The tipping of the phone controls the acceleration of the clouds in the X direction
  - Note we have to use the Y value of the acceleration as the accelerometer readings are relative to portrait mode

5: Tipping Clouds

Rob Miles
University of Hull



demo

XNA and MONOGame

MonoGame

- MonoGame is an Open Source implementation of the XNA Game Development Framework
- It is based on Version 4.x of the framework
- It lets you take your XNA games into all kinds of interesting places

Running MonoGame

- MonoGame runs on top of the .NET framework
- On Microsoft platforms this means that MonoGame just works
- On non-Microsoft platforms this means that MonoGame must run on top of an installation of Mono which provides the .NET underpinnings
  - Although MonoGame is free, Mono is something you have to buy

Platform Abilities

Platform	2D	3D	Audio	Gamepad	Gamer services	Networking	Content pipeline
Windows	x	x	x			x	
MacOS X	x	x	x			x	
iOS	x		x	x	x	x	
Android	x		x			x	
Linux	x	x	x			x	

- This is the current level of platform support
- Note that this is continually changing

XNA Games for Windows 8

- I'm going to focus on writing XNA games for Windows 8
- You can make proper Windows 8 versions of your games which will run on all Windows 8 platforms, including Surface RT

XNA Games for Windows Phone 8

- You can also use MonoGame to make XNA games for Windows Phone 8
  - However this is not as well developed as it is for Windows 8, and at the moment I don't think you can put them in market place
- For Windows 8 Phones you can still use XNA 4.0 and target Windows Phone 7.5 devices as well

Getting Started on Windows 8

1. Install Visual Studio 2012
2. Install Visual Studio 2010

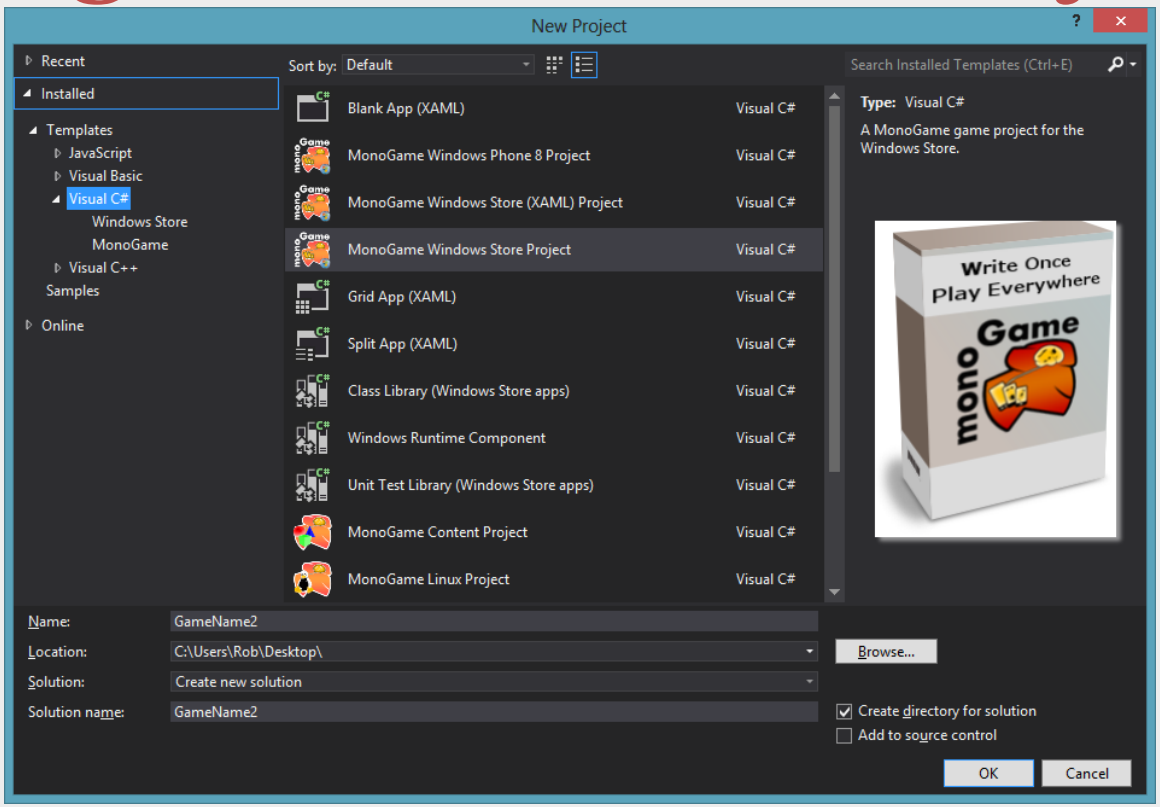
   http://www.microsoft.com/visualstudio/eng/downloads
3. Install the Games for Windows Client

   http://www.xbox.com/en-US/LIVE/PC/DownloadClient
4. Install XNA 4.0

   http://www.microsoft.com/en-us/download/details.aspx?id=23714
5. Install MonoGame

   http://www.monogame.net/

Making a MonoGame Project



- Monogame adds new project types

Running the application

- You can run the application on your development device, in an emulator on your device or remotely debug it in another device
  - Including a Microsoft Surface ☺
- You have to load the Debugging Tool into that to make it work

Content Management

Platform	2D	3D	Audio	Gamepad	Gamer services	Networking	Content pipeline
Windows	x	x	x			x	
MacOS X	x	x	x			x	
iOS	x		x	x	x	x	
Android	x		x			x	
Linux	x	x	x			x	

- There is no support for creating content for MonoGame solutions
- You have to use XNA for this

Making Content

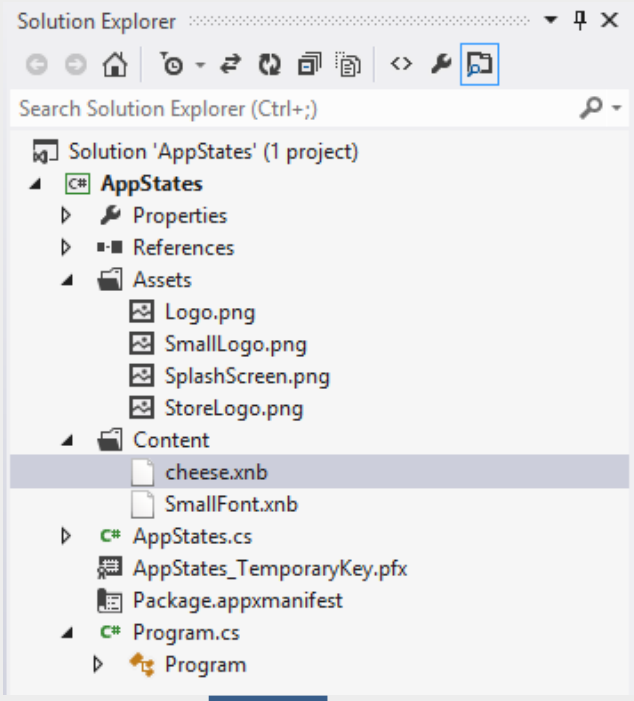
- This is a rather convoluted process

1. Create the content (Image and Sounds)
2. Make an XNA 4.0 project using Visual Studio 2010 and load it with the content
3. Drop the XNB files from the XNA project into your MonoGame project

Why we have to do this

- XNA manages content by creating intermediate files from the images and sounds we give it to put in our game
- These are the ".xnb" files
- When the game runs on the target platform these content files are loaded
- MonoGame only has the code that reads these content files

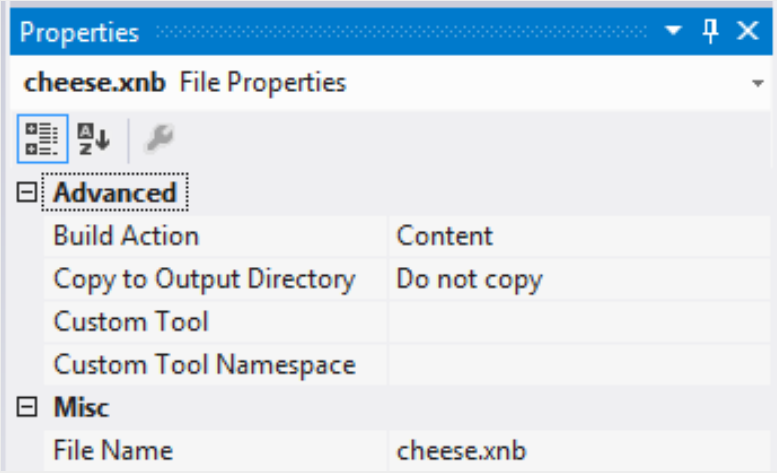
Placing the Resources




```
cheeseTexture = Content.Load<Texture2D>("cheese");
```

Content Management

- You will need to make a Content folder in the MonoGame project
- You will have to set the build action to “Content” for any content items that you add into this folder



Windows 8 Features

- MonoGame exposes the Windows interfaces in a similar way to the ones on Windows Phone
  - You get the touch screen, with gesture support, and the accelerometer
- These are very easy to use in a game

And so….

- Students like learning with XNA
- Writing XNA from scratch for Windows 8 is easy
- Converting existing XNA games is also easy
- MonoGame provides a path to many markets, including Windows 8

Resources

- MonoGame

  http://www.monogame.net/
- Mono Project

  http://www.mono-project.com
- Me

  http://www.robmiles.com