Week 1 - Friday

CS361
What did we talk about last time?
- Syllabus
- Colors
  - RGB
  - CMYK
  - HSL and HSV
Questions?
Image Formats
We will be thinking of images as linear buffers of data (which will usually store R,G,B and sometimes A values for each pixel)

Bitmaps (.bmp files) are almost that simple

Most common image formats (.jpg, .png, and .gif files) are more complex

They use different forms of compression to keep the image size small

Otherwise, an 800 x 600 image is 3 bytes per pixel x 800 x 600 = 1,440,000 bytes > 1 MB
JPEG or JPG images

- Stands for Joint Photographic Experts Group
- Good for images without too much high contrast (sharp edges)
- Photographs are often stored as JPEGs
- Uses crazy math (discrete cosine transform) to reduce the amount of data needed
- Lossy compression
PNG images

- Good for images with low numbers of colors and high contrast differences
- Has built-in compression sort of like zip files
- Similar to the older GIF (.gif) images
  - GIFs are problematic because they only support 256 colors
  - GIFs also suffered from legal battles over the algorithm used for compression
  - They have made a minor resurgence because of animation
- **Lossless** compression
Others

- **Tagged image file format** (.tiff or .tif) images are another standard sometimes used in computer graphics or for scanned images
  - The TIFF standard is really crazy, supporting layers, LZW style compressions, JPEG style compression
- **DirectDraw surface container** (.dds) files were designed for DirectX, allowing for the S3 Texture Compression algorithm
  - The pixel data is easily to decompress in hardware
- **TARGA** (.tga) files have a very simple structure and are still used for some textures
C#
## C#

### LIKE JAVA
- Primitive types and objects
  - All objects are references
- Mathematical operations are virtually identical
- Strings are immutable
- Garbage collection
- Single inheritance
- Exception handling
- Statically typed

### DIFFERENCES FROM JAVA
- True multidimensional arrays
- Methods by convention start with an uppercase letter
- Has properties
- Has operator overloading
- Not all methods are virtual
- Delegates (function pointers)
- Exceptions do not require a `try-catch` to compile
- Pointer arithmetic in unsafe mode
Hello, world in C# is very similar to the Java version

- Even so, it's highlighting differences in libraries and superficial structure that are not significant

```csharp
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```
Control structures

- `foreach` loops
  - Java:

```java
double[] data = getData();
double sum = 0.0;
for (double value : data )
    sum += value;
```

- C#

```csharp
double[] data = getData();
double sum = 0.0;
foreach (double value in data )
    sum += value;
```

- Switches (typos in the book)
  - `break` statements are not optional
  - If you want to go to another case, use a `goto` statement
### Classes and methods

- **structs**
  - Like classes but are value types
  - Have no inheritance
  - Intended for lightweight objects like point or rectangle

- By using the **partial** keyword, a single class can be defined in multiple files

- Methods are not all virtual by default
  - You can only override methods marked **virtual**
  - Overriding methods must be marked either **new** or **override**

- Parameters are passed by value by default but can be passed by reference or result
Multidimensional arrays

- There are no multidimensional arrays in Java (only arrays of arrays)
- To declare a multidimensional array in C#, use commas to separate the dimensions

```csharp
int[,] table = new int[4,9];
```

- How is this different from arrays of arrays?
  - Uses less memory
  - Cannot be ragged (different length second dimensions arrays)
  - Slightly slower to access data
In C#, there is a special construct for getters and setters called a **property**.

A property allows you to get or set a private member variable as if it were public.

It looks as if you are simply reading or writing a variable, but arbitrary code can be executed when you use a property.

They're a convenient way to call accessors and mutators.
class Person
{
    private string name = "No Name";

    // Declare a Name property of type string
    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value; //special value variable
        }
    }
}
With a property defined, getting or setting the name on a **Person** is easy

```csharp
Person samuel = new Person();
samuel.Name = "Samuel L. Jackson";
string badass = samuel.Name;
```

It looks like nothing is happening, but a method is getting called

```csharp
DateTime time1 = DateTime.Now;
Thread.Sleep(1000);
DateTime time2 = DateTime.Now; //one second later
```
MonoGame
RB Whitaker has made some very useful MonoGame tutorials that we'll be using:
- http://rbwhitaker.wikidot.com/monogame-tutorials

These tutorials are ports of his (more extensive) XNA tutorials, which are also useful to check out:
- http://rbwhitaker.wikidot.com/xna-tutorials
If you are only familiar with Eclipse, Visual Studio will seem similar.

- Work is organized in Solutions.
- Solutions can contain one or more Projects.
- In Eclipse, merely having a source code file in your project folder will cause it to be loaded as part of the project.
- In Visual Studio, you have to explicitly add items to your Projects.
  - For MonoGame, this can be game content as well as source code.
To create a new MonoGame project
- Select **New Project...**
- Under Visual C#, select **MonoGame**
- Name your project whatever you want

There will be a **Program.cs** that creates and runs an object of type **Game1** (which you can rename if you want)

Everything happens inside of the **Game1** class
Game1 class

- You can rename this class if you want
- It contains:
  - Constructor
    - Self-explanatory, not too important right now
  - `Initialize()` method
    - For initialization and loading of non-graphical content at the beginning of the game
  - `LoadContent()` method
    - For loading graphical content at the beginning of the game
  - `Update()` method
    - Update the state of your game, called each frame
  - `Draw()` method
    - Draw the state of your game on the screen, called each frame
Running the game

- Hit **Ctrl+F5** to run the game without debugging
- A window should pop up like this
- Each frame, no updating is done
- The screen is cleared to cornflower blue

Cornflower blue isn't important, but it is deliberately not black or white, since it's easier to produce black or white output by mistake
We're used to interacting with programs from the command line (console)
MonoGame was not designed with this in mind
  - It has pretty easy ways to read from the keyboard, the mouse, and also Xbox 360 controllers
But you'll need a console for Project 1 so that you can tell it which file to load and what kind of manipulations to perform on it
So that Console.Write() and Console.Read() work
  - Go to the Properties page for your project
  - Go to the Application tab
  - Change Output Type to Console Application
More information: http://rbwhitaker.wikidot.com/console-windows
You'll need a separate thread to read and write to the console if you don't want your game to freeze up
To draw a picture on the screen, we need to load it first.

Right click the Content folder in your game solution and choose **Add** and then **Existing Item...**
- Make sure the Build Action is **Content**
- Set Copy to Output Directory to **Copy if newer**

Find an image you want on your hard drive.

Create a **Texture2D** member variable to hold it
- Assume the member variable is called **cat** and the content is called **cat.jpg**

In **LoadContent()**, add the line:

```csharp
    cat = Content.Load<Texture2D>("cat.jpg");
```
Now the variable cat contains a loaded 2D texture
Inside the Draw() method, add the following code:

```csharp
spriteBatch.Begin();
spriteBatch.Draw(cat, new Vector2(x, y), Color.White);
spriteBatch.End();
```

- This will draw cat at location (x, y)
- All sprites need to be drawn between Begin() and End() spriteBatch calls
Upcoming
Next time...

- Graphics rendering pipeline
- Application stage
Reminders

- Read Chapter 2
  - Focus on 2.1 and 2.2