Combined Best Practice Guide

This guide will give you tips and ideas on how to improve your programs, websites. It combines some of individual guides found on this site (http://ssjx.co.uk):

- Programming: Best Practices / Maths Loops / Securing User Input
- Website: Best Practice / General Tips / Graphics (Sprite sheets) / Images formats

A lot of the content overlapped and having this one guide made sense.

Note that any example code is demonstrated in the language that it is easiest to highlight the issue with, the issue still applies to any language. These are probably subjective and I do not always follow them, but here you go!

Enable Any Warnings/Strict Modes That Are Available

You will make mistakes, so it is better the compiler/interpreter catches them early before they turn into problems. Enabling any strict modes will force you to write better code, e.g.:

- For Javascript, "use strict"; should be at the start of all the programs. It forces a stricter subset of JavaScript which helps make programs more secure with less chance chance of errors.
- In Perl use strict; use warnings;. Like above, it will help flag more potential problems and help make your program more secure and maintainable by catching mistakes.
- In Pascal {$J-} {$R+}. Enables constants being constant(!) and range checking.
- In D, putting @safe where possible will enable additional protection to help make sure your program is memory safe.

Most compilers have switches to enable more thorough checking too. Use them.

It may be worth looking at using a language that enforces strictness by default such as D, Go or Rust .

Use Array Lengths In Loops

The following is in JavaScript to more clearly show the problem. The block below (and the equivalent in other languages) will happily run and compile. The problem is, if you go back and change the array size, you risk trashing memory and making your program unstable. Depending on how this is triggered this could be used in various attacks.

```
var score=new array(10);

for(var i=0;i<10;i++)
{
	// do something
}
```

A better way is to get the size of the array and use that:

```
var score=new array(10);

for(var i=0;i<score.length;i++)
{
	// do something
}
```

An even better way is to use a range based loop, depending on what you plan to do with the array, in D you could do the following:

```
int[10] score;

foreach(s;score)
{
	// do something (read only) with each item in the array
}

foreach(ref s;score)
{
	// do something (read/write) with each item in the array
}

foreach(i;0..score.length)
{
	// use i as a counter
}
```

Using strict modes and compiler warnings, this type of over/underflow error are usually caught but the above is still worth doing.

Avoid Unnecessary Type Conversions

This is probably best illustrated below:

```
dim as string one="1"
dim as integer two=2
dim as integer total

total=int(one)+two
```

Nothing wrong with the program, but it is wasteful converting a string to an integer when it should have been and integer to begin with.

Pre-Calculate / Reordering Of Any Complex Maths Within Loops

The code below fills an array (could be a screen or bitmap). The bit of interest is the (10*j)+i.

```
for j=0 to 9
    for i=0 to 9
        screen[(10*j)+i]=0
    next i
next j
```

As 'j' does not change inside the inner loop, part of the equation can be removed from it as shown below. This does add another variable (called start in the example) but that is usually a lower cost than a multiply.

```
for j=0 to 9
    start=(10*j)
    for i=0 to 9
        screen[start+i]=0
    next i
next j
```

Between the two code blocks we have gone from having 100 multiplies to just 10! In the small example above, the start value could even be turned into a pre-calculated array and taken out of the loop:

```
const int[] start={0,10,20,30,40,50,60,70,80,90}
```

meaning the screen line would look like:

```
screen[start[j]+i]=0
```

This means no multiplications at all in our loop. As a general rule, multiply / divide are usually slower than add / subtract.

Use Sizes That Work For Shift Instructions

In the above example we have the line start=(10*j) to read/write to our 10x10 grid. Using a better size would enable us to use a Shift instruction instead which are usually much faster.

If we used a 16x16 map, we could use a shift left of 4 instead of the multiply:

```
for j=0 to 15
    start=(j<<4)
    for i=0 to 15
        screen[start+i]=0
    next i
next j
```

Note that if the larger 16x16 size is allocated for a map, the actual used size could still be 10x10.

Choose The Best Data Type

Our latest game will use an array to store its map:

```
// Space for a 256 x 256 tile map
const w=256,h=256
dim as integer map(w*h)
```

Again, nothing wrong with the above, it allocates 256KB for the map. However, if we know that each value will be less than 255 the code below will use a quarter of the memory just by changing its type:

```
// Space for a 256 x 256 tile map
const w=256,h=256
dim as byte map(w*h)
```

For reasons of speed and processor optimisation, the first example may be best, but it is always worth checking.

Securing User Input

Whenever your program gets input from the user, there is a potential security risk. User input can be anything from a name typed into a form to a search query passed to a web page.

Either intentionally or unintentionally, incorrect input can cause your program to crash or in a worse case allow the user to see or do something they should not be able to.

Two very simple checks that are worth doing:

- Make sure the data you get is of a sensible length, reject it if not. E.g. A telephone number is unlikely to be longer than 20 characters.
- Make sure the data you get only contains the characters you expect. E.g. A telephone number will not have an @ in it.

An example of a function which does these checks:

```
// This example is shown using the D language, it checks that each character in the input string
// is in the list of valid characters.
import std.stdio;

@safe:

bool isvalid(const string input){
	if (input.length>6){
		writeln(input," is too long!");
		return false;
	}

	bool found;
	const string valid="0123456789";
```

```
	foreach(c;input){
		found=false;
		// See if our input character is in the list of valid characters
		foreach(v;valid){
			if (c==v){found=true;break;}
		}

		if (found==false){
			writeln(input," is NOT a number");
			return false;
		}
	}
	writeln(input," is a number");
	return true;
}

void main(){
	isvalid("0123");	// Valid
	isvalid("a0123");	// Not valid (a)
	isvalid("12.3");	// Not valid (dot)
	isvalid("-123");	// Not valid (minus)
	isvalid("1234567");	// Not valid (too long)
}
```

The function above checks the input against a list of valid characters, if it checked for invalid ones, we would need to include every possible invalid character which would likely miss some.

Re-Inventing The Wheel

Sometimes it is worth re-inventing if the alternatives are too complex, too big or worse than making it yourself. Other times it is not:

Standard rectangle within a rectangle collision function (in ActionScript / Flex):

```
 if (swans.x>=player.x && (swans.x<=(player.x+player.w))
{
	if (swans.y>=player.y && (swans.y<=(player.y+player.h))
	{

	}
}

// x4 for each corner....
```

Or we could just use the existing function which is less typing and most likely much faster:

```
 swans.rectangle(10,10,100,100).intersects(player.rectangle(10,10,100,100) )
```

The point being that it is worth checking if the language already has a function for what you are doing. Java and ActionScript have a huge number of methods as standard which may include something you are currently re-inventing.

Do You Really Need Large Dependencies?

Both programs below accomplish the same thing:

```
// Java Hello World
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}

// C Hello World
#include <stdio.h>

void main()
{
        printf("Hello World");
}
```

The key difference being the Java version has a massive dependancy on requiring the Java Runtime Environment. That's a 300MB install for what is a 2KB native program. There are many good reasons to use Java (portability / features), this is not one of them.

Other oversized dependencies include various database systems, does your project really need MS/MySQL? If you are just dealing with a small number of records then maybe something like SQLite would do instead. Or maybe just use the languages standard file methods?

Choosing The Correct Image Format

A quick overview of the the main image formats used on the web:

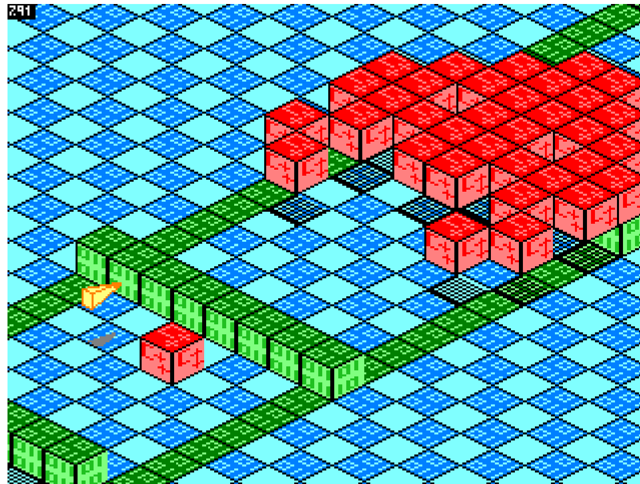
- GIF - 256 colours, used mainly for graphics such as icons. Image compression is usually not as good as PNG/WebP. Can also be used for animations.

- JPG - Used for photo images, supported by everything. Lossy format, the higher the compression, the more detail is lost.

- PNG - Lossless format, good for archiving photos but not for putting them on a website due to size. For lower colour images, it can offer better compression than GIF.

- WEBP - Has both lossy and lossless modes so good for photos and graphics. Excellent compression in both modes. The main negative is the lack of browser support, it is currently only supported in Chrome and the latest Firefox.

The image below is 320x240 and has 12 colours, as an uncompressed bitmap it is 1201KB in size. This kind of image is an average screenshot for this site.



The following are the sizes when resaved in the different formats:

- GIF Size : 10.7KB
- PNG Size : 6.92KB (Using IrfanView)
- PNG Size : 3.57KB (Using OptiPNG)
- WEBP Size : 3.45KB

The WEBP image is the smallest image,the only real downside is the lack of browser support. What this means is that your site could have 3 times as many PNG/WEBP images in the space that the GIF's take up, the download for a single image would also be around 3 times faster!

A JPEG is not the best choice for the above type of image due to its lossy nature. For purposes of comparison though, saving as a JPEG at 80% quality resulted in a 47KB file.

After deciding on the correct format to use, the next important task is to resize your image. Putting a large image on your website and letting the browser resize it means you are transferring a large image when a smaller one would be better. Although user internet speed is increasing, transferring a smaller image costs less in bandwidth and means an even faster download for the user which means an improved user experience.

Use Sprite Sheets

Some of the games on this website use many individual sprites. The first release of the block game ColourFall had 15 individual images totalling 14KB, not a huge amount but by combining some of the sprites I was able to get it down to 5 images totalling 7KB! The reasons for this:

- The resolution and colours used were low so the duplicated image headers/palette of the individual images contributed a fair bit to the total file space used. The single combined image header/palette did not contain much more information so stayed reasonably small.
- As not many colours were used in the combined image (all.gif), the gif compression works really well. For multicoloured complex graphics, the compression factor may not be quite as great. *It should be noted that this was a 3.74KB GIF image, as a PNG it became even*

smaller at 1.13KB!

There are many advantages to using a sprite sheet over individual sprite/image files:

- On the server side, it means fewer connections to the hosting server, a reduction in bandwidth required to serve your users and less file space required to store your game. Together these may help reduce hosting costs for your game.

- For the user, it will mean a faster download and less waiting giving the user a more positive experience.

It should be noted that these advantages can apply to games not hosted on websites too. All games benefit from being smaller, more efficient and with faster load times.

Aim Lower

For websites html5 is the latest version of the html standard, it offers many new features and deprecates a lot of unused features. Does your site actually need or use any of them though?

If you were to target html4 strict, you will ensure compatibility with more older browsers and with current browsers that have trouble with the newer standard.

Are you using a complex content management system where a static page would do the same job? A single static page is a lot easier to maintain and secure than the components needed to support a database driven web site.

Aiming lower also applies to programming. A lot of application make use of Intel's SSE2 extensions, used correctly they can make a big difference to performance. The downside is that it means the programs will not run on anything that does not have these extensions. Programs should make use of available features but include a fallback means users can use their existing computers for longer. *(This is basically my annoyance at Mozilla Firefox for dropping support for my AthlonXP which was fast enough for everything else I did...)*

Do Not Make Your Website Require Javascript

Javascript can be useful, it is used by games and activities on this site and to make the drop down menus work. There are many down sides to it though:

- **Compatibility** - Javascript is constantly evolving adding new features and removing others. If you use the latest features and your users do not have a compatible browser, they will get a poor user experience. Many users may not have the option to upgrade their browser, they could be using an older iPad or Android device which is no longer supported.

- **Slow** - Heavy use of Javascript can slow the browser and the user's computer. This can be caused by the script polling or refreshing too often. Script errors (sometimes caused by the above) can also lead to slowness as the script eats memory while silently failing.

- **No Javascript** - For the above reasons and for increased security (some sites may host malicious scripts), some users will not have Javascript enabled. A good web site should still work.

A good website should be enhanced by Javascript, not dependant on it.

Bandwidth Is Not Free!

On this site, some of the popular(!) downloads are offered as 7Zip files. These are compressed archive files much like Zip file, the difference being that they are even smaller. Animator is a 714KB zip file and a 499KB 7zip file, that makes the 7zip about a third smaller.

Although hosting two archives takes up more space on the site, more people will appreciate the quicker download. Those with download limits will prefer a smaller download even more. Although some people may not have the 7zip program installed, for large programs the 1MB 7zip installer could save a much bigger download. There is also the option of a 7zip self extracting file.

It should be noted that there are alternatives to 7zip, I use that as an example as it is the one that I use.

Files And Downloads

- **Avoid mystery download links** - If you link to a file the user can download (e.g. document or archive), make sure you say what the file is it is and how big it is. If it's an uncommon file type, maybe suggest a tool the user can download to open it.

- **Avoid obscure file formats** - Following from the above, if you do list a file for the user to download, try and use the most common file format you can so the user does not need to find a special tool. For example, although not popular, almost every device can open a PDF document but not everyone can open a Microsoft Word document.

- **Offer 7zip files as well as Zip file**s - Using Zip files to compress files is always a great idea, smaller files and some degree of error checking so the user knows if a file is damaged in the download. If possible, maybe offer 7Zip files as well? They offer even smaller files which means even quicker downloads, the downside is that they are not as common at the moment but more general use would be great.

Offline Documentation

The advantage of online documentation is that it is easy to update and is always current. I have noticed that it is becoming more common for applications to install a copy of the web documentation with their application. Although this seems like a good thing, it is a pain due to the following:

- **Searching does not work** – On the web, the web searches make use of server side technologies to find what you are looking for across all the documentation pages. Offline this does not work.

- **Hundreds of small files** – Instead of just one document, web documentation is made up of lots of small html files, usually hundreds of them. Copying many small files is a slow process and can mean more work for virus scanners. It can also take up more space than a single file.

As an example of good documentation, the FreeBASIC Manual is available as single CHM file, it is small (1.7MB) and quick to search.

This document started as six separate web pages, although it would have been possible for the user to print each as a PDF, by replacing with this single PDF (created using OpenOffice) there are many advantages:

- **Easier to read** – PDF readers are much nicer than browsers for reading documents. Using Foxit Reader (or Adobe Reader) I can easily adjust the colours and presentation, the bookmarks generated by OpenOffice allow the reader to jump to particular sections and searching for text is easy and fast.

- **Easy to share and keep** – Instead of multiple files and images it is just one self contained document. Once downloaded you never have to worry about having access to the information if the website or your internet goes down.

- **Easy to create / modify** – This is more of a Text Editor vs Word Processor argument but writing this in OpenOffice Writer is much nicer than Notepad++ (as great as it is) plus I get the benefit of a spell checker!

For the types of documentation mentioned at the start of this section, there are various tools which can automate the creation of PDF documents and single file documents like the mention CHM files.

Last updated 26/04/2020
by ssjx (http://ssjx.co.uk)