CODING STANDARDS

Classes and Functions

All Classes and Functions will be no longer than 20 characters in length and be descriptive. No short hand terms may be used. They must all begin with a capital letter and have a capital letter for each first letter of a new word in the name. Any and all parenthesis must be spaced after the object name and within the parenthesis themselves. Any arguments contained within the parenthesis must also be spaced, allowing for a space between the first and last arguments and the parenthesis.

Examples:

```
function ShowBox ( ID )
function MouseEvent ( Event, nXPosition, nYPosition )
```

Variables

Variables shall be no longer than 10 characters (if it can be helped). Due to the conventions of JavaScript, a variable can be any type you set it to (from strings to numbers to whole functions and classes). Due to this, the first letter of each variable must indicate what that variable is used for. If the variable is for a number (integer, float, double) indicate it by starting with "n" for number. If the variable is a string, use "s". If the variable is a function or class object, use "o". If the variable contains HTML, use "h". If the variable is an array, you must first use the type prefix then add an "a". Using it in the other way may cause confusion such as "anArrray" or "asName". Matrices, unlike arrays, must only use "m" for a prefix. This need not apply for any temporary single use variables such as for iterators. Specialty variables such as identifiers (ID), events (keyboard and mouse), etc must be named justly to indicate its purpose, foregoing the prefix notation for conventional variables.
The first letter following the indicator variable must be uppercase. Unlike classes and functions, short hand terms may be used, but only if they are obvious as to their meaning. You must also try to keep variables either local or a part of a class. Due to the conventions of JavaScript, always use the "var" name identifier for each new variable created. Not doing so will create a variable automatically, making it hard to understand.

Strings and HTML code will all use double quotes " ". This is to allow for HTML code to have strings within strings using single quotes ' '.

Example:

```
var sName = "Conception";
var nPosition = 100;
var hDivBox = "<div onClick='DoSomething()'></div>";
```

Methods (Member Functions)

All methods shall be descriptive of their functionality and be no longer than 20 characters. All methods shall be prefixed by a description of the action they are performing (excluding functions that are an extended or implemented form of an existing interface). Functions that modify data shall be prefixed with "set". Functions that query for data shall be prefixed with "get". Functions that return a Boolean value shall be prefixed with "is".

Example:

```
getBoxID ( );
setBoxID ( ID );
isColliding ( );
```

Code Blocks & Iteration

All code blocks shall conform to the same basic principles; this includes functions. All blocks of code (the statements in between the curly braces) must be indented within the block itself by one tab. anything that can make use of the code block must place the beginning curly brace one line below its statement line. The end curly brace must be place after the last statement within the code block, moving back one tab for a clean finish. Encapsulated code blocks will continue the tabbed trend, no matter how many are used.

Example:

```
while ( true )
{
      getBoxID();
}
```

Iteration statements such as "for" loops that make use of variables are free to use single letter variables. The conditions and statements that make up a "for" in particular must use proper spacing to make the

statement readable as a whole. This includes spacing, like a function, for each parenthesis and arguments after each semicolon.

Example:

```
for ( var i = 0; i < 10; i++ )
{
      document.innerHTML += i "<br />";
}
```

Only C/C++ styles of iterations may be used. The C# method ("for (x in Array)") will not be used.

Arrays

JavaScript has three ways to declare arrays; first is the regular method, the second is called a condensed array, the third is the literal array. All arrays can follow either the traditional or condensed array styling. Literal arrays can be cryptic to understand at first glance.

Example:

```
// Regular Array
var saCars = new Array ( );
saCars[0] = "Saab";
saCars[1] = "Volvo";
saCars[2] = "BMW";
// Condensed Array
var saCars = new Array ( "Saab", "Volvo", "BMW" );
```

Matrix arrays must be created using the regular array method.

Example:

```
// 2 x 2 matrix with 1s and 0s
var mBinary = new Array ( 2, 2 )
mBinary[0][0] = 0;
mBinary[0][1] = 1;
mBinary[1][0] = 1;
mBinary[1][1] = 0;
```

Commenting

Due to the change to JavaScript, we can no longer rely on JavaDoc. We must therefore comment code in a clear and concise manner. Comments are to make use of both the traditional one line comment "//" and the multiline comment box "/**/". Single line comments are only to be used for simple comments where needed. After the double slash, use a space after for easier reading. Comment blocks are to be used to make in depth explanations, examples, and variable makeup.

Each JavaScript file for any particular code must contain the following information at the top of each page, following the indentation using tabbing.

```
/*****************************
* Name: Danny Vargas
* File: [Filename].js
* Project: {AlgorithmA}; 2010
* Date Created: February 2, 2010
* Date Modified: February 3, 2010
* Purpose:
* [List the purpose of this file. Comment
* where and how it is used. Be sure to
* give any details of what it changes]
* Dependencies:
* [Files, functions, and variables required
* to run the code in the file.]
* Notes/Changelog:
* [List any notes such as, what has been
* changed and when]
* Functions:
* [Function name] : [One line description]
* […]
*****************************/
```

Each portion of this is made so as to let anyone know at a quick glance what they are getting into with this particular file. The modified date must be done after each successful change to the code is made. The purpose must provide an in depth reasoning for the file. Dependencies must list each file, function, and variable used. The exception for this is for common utility files such as animation libraries. Notes must let the reader know what has changed. Functions

are a list of all functions and classes found within the file. List their name, arguments if possible, and a short one line description of its task.

Before each function must be an in depth explanation of the task the function performs. This is to cut down on the number of single line comments made within the code.
Single line comments must be made on blocks of code difficult to read or requiring a special operation to complete. Iterations must be documented as to their functionality and so forth.

HTML and PHP Coding Standards

The accompanying code following this text will implement every detail that is defined in this section.

General HTML

At the beginning of each HTML file, a five-line header will be implemented like the JavaScript files above. The first line will have the name of the Software Engineer. The second line will contain the Date which this file was created. The third line will contain the file name and line four will contain the revision number of the file. The final line of the header will be a detailed description of the file. It will explain the functionality and possible outcome. The header will be enclosed within comments of the form "<!-- *** -->". This header will be implemented before the major HTML tag. Indenting with tab will make the overall header easier to read, following the same context as the JavaScript header.

Unlike previous versions of AlgorithmA, we will compact the HTML the overall design of the site by not using a separating line between each and every major tags used. Spacing will be used before the use of a major tag (<HTML>, <HEAD>, <BODY>) and after. Indentation will be used heavily, treating them as you would code blocks in JavaScript. This does not apply however to the major tags, however all tags placed within must be indented appropriately. All singular elements will conform to the latest standard of HTML using self-closing tags (i.e.: "
<img />"). These closed tags will be spaced from the rest of the tag itself as shown.

```
<!--
- Name: Danny Vargas
- File: Simple.html
- Project: {AlgorithmA}; 2010
- Date Created: February 2, 2010
- Date Modified: February 3, 2010
- Description: A basic intro to HTML for CSCI 455
- Notes:
- - Notes usually placed here.
-->
<html>
<head>
      <title>Welcome AlgorithmA Team!</title>
      <!-- Insert scripts and CSS here -->
</head>
<!-- Sets background to a light grey, text to white -->
<body bgcolor = ”#E9E9E9” text = “#FFF”>
      <!-- Unordered list -->
      <ul>
            <li>How do you do?</li>
            <li>What are you doing?</li>
            <li>Happy New Year!!!</li>
      </ul>
      <!-- centered statement paragraph -->
      <p align=”center”> Good luck with AlgorithmA 2010!!!</p>
</body>
</html>
```

General PHP

For PHP, the same methodology as HTML and JavaScript must be employed. At the beginning of each PHP file, a five line header will be implemented. The first line will have the name of the Software Engineer. The second line will contain the date when this file was created. The third line will contain the file name of the file. The fourth line will contain the revision number of the file. The final line of the header will be a detailed description of the file. It will explain the functionality and possible outcome. The header will be enclosed within comments of the form “/* --- */”. This header will be embedded within the pre-defined PHP tags (“<?php” for opening and “?>” for closing).

PHP Variables

Each variable will start with a dollar sign ($) and will be no longer than eight (8) characters long. The variable will be named according to the values that will be stored in it, starting with lowercase for the first word and uppercase for the second.

String values will be enclosed in single quotes, also called 'apostrophes' (' and '). Every control structure within the PHP code will contain its truth parameters and the body to be executed.

PHP Code Blocks

All control structures containing only one line, or many lines, of code in its code body will utilize an opening brace "{" before, and a closing brace "}" after the code body in that section. All lines of code within each block will be indented by a single tab (4 spaces) to simplify and increase the readability of the code. For encapsulated code blocks, another indent will be used and will continue for every code block within that one.

PHP Functions

Functions will be used to simplify the creation of larger bodies of organization on a web page and will not exceed fifteen (15) characters long. The function name shall fully describe the functionality and purpose of that function. Each word of the name will be capitalized, starting with the first name and every subsequent name following. Function bodies will be enclosed in curly braces.

PHP Comments

Much like with JavaScript, comments will follow the same guidelines issued before. Single quotes "//" must be used to describe brief functionality descriptions before major lines of code. A space must be put after the double slash for easier reading. Comment blocks "/**/" will be used to go into detail on how much larger portions of code operate. Before each function, you must document in a block comment what the function performs. This is to cut down the amount of commenting you need within the code itself. The beginning of each PHP file will contain the creator's name, filename, project, creation

date, modified date, description, notes, a list of dependencies, and a list of functions with a brief description of each found in the file.

```
<?php
/*****************************
* Name:          Danny Vargas
* File:          [Filename].php
* Project:       {AlgorithmA}; 2010
* Date Created:  February 2, 2010
* Date Modified: February 3, 2010
* Purpose:
      * [List the purpose of this file. Comment
      * where and how it is used. Be sure to
      * give any details of what it changes]
* Dependencies:
      * [Files, functions, and variables required
      * to run the code in the file.]
      * Notes/Changelog:
      * [List any notes such as, what has been
      * changed and when]
* Functions:
      * [Function name] : [One line description]
      * Create_Layout : Create layout depending on choice
*****************************/
// This variable stores choice of user for selected layout.
$choice = $_POST['choice'];

// Counter used in for loop.
$count = 1;
// Will be used in debugging later on.
$check = 'Debugger!';
// Calls function Create_Layout
Create_Layout($choice);
//Debugging Line
print $check;
print "<br /><br />";
//If loop checks if inputted choice is correct.
if ($choice != '1' || $choice != '2')
      {
            print "You have chosen wrongly! <BR>";
      }
//Defines the layout that was chosen by user.
```

```
function Create_Layout($variable)
{
if ($choice == '1') //Choice One: Three Frames
      {
            print "You Have Chosen Layout One! <br />";
            print "Thank you for choosing! <br/ >";
            print "Layout one contains: A Logo, A Body, and A
            Menu<BR>";
//Message
      }
      if ($choice == '2') //Choice Two: Three Frames
      {
            print "You Have Chosen Layout Two! <br />";
            print "Thank you for choosing! <br />";
            print "Layout two contains: Two Logo, Body, and A Drop-Down
            Menu<br />";
      }
} //End function

//Testing arithmetic
for ( $i; $i < 5; %i + 1 )
      {
            print "Testing the operations";
            print "<br />";
            $multiplication = 20 * $i;
            print $multiplication; //Prints result
      }
?>
```

Pseudo code Standards

All coding teams will use a standard system of pseudo code to convert interactivity diagrams to Java code. The pseudo code language "AL," will be used for all such purposes for clarity and consistency. The "AL" pseudo code language is fully described in self contained documentation elsewhere and is available upon request.

In general, the "AL" pseudo code language is logically similar to the actual code, but does not require the syntactic technicalities of any actual coding language.

The pseudo code submitted by each team or engineer will be consistent with all interactivity diagrams submitted for the same programming task, and will be used to generate the actual source code for that task.

To maintain the required consistency, any changes to actual programming code (for any reason, including "bugs," change in rationale, or new configuration requirements) that deviates from, or in any way modifies the underlying logic, will also be changed in the associated pseudo code. Therefore, for maintenance purposes, all supporting documentation will consistently represent the executable code.

The pseudo code will also adhere to the same logical construction as the programming language, relating to repetition structures, condition ("decision") structures, and assignment structures.

File Storage Standards

This section is defined to prevent the ambiguity of file and directory names that can arise from multiple files or directories of the same name, such as

```
MyFile.CPP
MyFile.cpp
myfile.cpp
MyDirectory
Mydirectory
```

This is caused by a case sensitive OS that distinguishes between upper and lower case letters in file names, allocating separate storage for files whose names would, except for the case of the characters used, be the same file (or directory) name.

Therefore, all file and directory names will be defined in all lower case letters. The structure itself will follow the MVC model of design, with names corresponding to their respective section and what type of files they contain (i.e.: "image" for images and "JavaScript" for JavaScript). Files will also be lowercase, using names to actively describe what they represent and are used for.

Metrics

The metrics used by each team are as follows:

Quality Metrics

- Bugzilla error and exception tracking software will be used to manage code faults.
  - Faults will be logged and tracked by members of any team not working on that particular section of the code.
  - Faults will be managed by the Team Leads.
- Faults per line of code.
  - There shall be 5 or fewer per every 100 lines of code.
  - This is due in part of the nature of JavaScript not needing to be large to be useful.
- Compliance with defined standards.
  - The product shall pass reviews and audits, and be in compliance with pre-defined SQAP standards.

- The number of functional requirements.
  - This method will be used to gauge the completeness of the project at the time of the review.

Management Metrics

- How many features have been implemented?
- Personnel allocation.
  - How much they are able to accomplish as reported by the Team Leads.
- How many tasks has been completed/in-progress.

Teams will use the tools provided by ClockingIT.com, SVN, reviews, and audits to measure progress and track goals. Standards and regulations for completed products will be adhered to as defined in the SPMP. The metrics collection plan will implement the above metrics, while considering the standards and regulations governing the finished product.

ClockingIT.com will track overall progress of the project by taking into consideration the number of tasks that have been completed by assigned teams. This data can be represented as a percentage, graphs,

and charts. For descriptions of specific project deliverables, refer to the SPMP.

These metrics shall be determined using status meetings, milestones, and code inspections. The status meeting will allow the management to gauge the overall progress of the project. Milestones will allow the management team to set specific dates for product submission, and based on the time of submission can be used to gauge the progress of this project, and to predict future milestone dates. Code review will allow the management team to analyze the product's overall functionality and gauge the areas of the product where most errors are occurring and resolve any issues discovered.