

**Year 10**

14

**St. Joseph’s College Hunters Hill**

Semester 1 IST Portfolio

Jordan Lewis

A personal profile and review of all the work done in Semester 1 of Year 10 IST, including three flash software programs and eight theory documents.

08

**Fall**

Acknowledgements

Dates: Saturday 17th May – Friday 30th May

Screenshots were taken by using SnagIt 2.

Video tutorials and code for Pizza Order Form, Pong, and Spritz by Mr. Tim Clark.

Pizza Image (Pizza Order Form): <http://www.kinggeorgepizzas.com/>

Secured by VeriSign Logo (Design your own GUI): <https://www.verisigninc.com/>

Quack Sound (Pong): <http://www.freesound.org/>

Abstract Background (Spritz): <http://www.full-hd-wallpaper.com/>

Copyright ©2014 Jordan Lewis. All rights reserved.

Table of Contents

Personal Profile 5

Who am I? & What do I enjoy doing? 5

What do I hope to do when I leave school? 5

What did I think of the Software Design Topic? 5

8.1.0: Project 1 – Pizza Order Form 6

What is the purpose of this product? 7

How did my product achieve this purpose? 7

What improvements could I make to this application? 7

8.2.0: Project 2 – Pong 8

What is the purpose of this product? 9

How did my product achieve this purpose? 9

What improvements could I make to this application? 9

8.3.0: Project 3 – Spritz 10

What is the purpose of this product? 11

How did my product achieve this purpose? 11

What improvements could I make to this application? 11

8.1.1 - Types of Languages 12

Satisfactory 12

Commendable 14

Exceptional 15

8.1.2 - Variables 17

Satisfactory 17

Commendable 17

Exceptional 18

8.1.3 - GUI Concepts 19

Satisfactory 19

Commendable 20

Exceptional 21

8.1.4 – Create your own GUI 22

8.2.1 - Data Types (8.2.1) 23

Elementary 23

6 Most Common Data Types 23

Basic 23

Strings 23

Integers 23

Real Numbers 23

Booleans 24

Sound 24

Thorough 24

Extensive 25

8.2.2 – Data Coding 26

8.2.3 - Operators and Algorithms 27

Elementary & Basic 27

Sound 28

Thorough 29

Pseudo Code 29

Extensive 29

8.3.1 - Subprograms and Languages 30

Elementary 30

Basic 31

Sound 31

Thorough 32

Extensive 33

8.3.2 - Testing and Error Detection and Correction 34

Elementary 34

Basic 34

Sound 36

Thorough 36

Extensive 37

Bibliography 38

# Personal Profile

## Macintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Personal Profile:Picture of Jordan.pngWho am I? & What do I enjoy doing?

Figure 1: Got a Hat Trick on DartsLive.

My name is Jordan Lewis. I am a 16-year-old student at St. Joseph’s College Hunters Hill. I enjoy working on computers and programming for iOS, which is done in the language Objective-C. I usually spend my free time working on my iOS application projects. Recently I have been working on an app that works as a logbook for learner drivers. I feel that this app I am working on at the moment will be a game changer for the RMS’s logbooks for learner drivers.

## What do I hope to do when I leave school?

When I leave school, I would like to go to the University of Technology Sydney and take a course in programming. After which, I would like to pursue a carrier in computer programming. I do enjoy working on my own personal projects so I hope to be able to work in a computer-programming job, while being able to work on my own apps and releasing them onto the iOS Appstore.

## What did I think of the Software Design Topic?

I think that the Semester 1 software design topics where much more enjoyable than last year’s, but still mundane for me. I think the topic presented many challenges to other students taking this course, but I found that I could complete most tasks in within an hour or two to the extensive level, which although gave me time to work on personal projects during class, makes me feel like I didn’t get much out of the topic. I am looking forward to doing Software Design and Development as an elective in years 11 and 12, because I think I will get a lot more out of it.

# Macintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Evaluation:Screenshots:Project 1:Main Screen.png8.1.0: Project 1 – Pizza Order Form

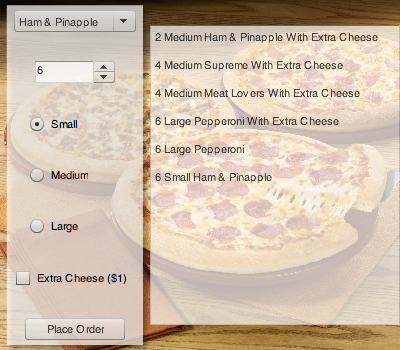


Figure 3: The UI after the form has been used.

Figure 2: Default look of my pizza order form.

## What is the purpose of this product?

The purpose of the Pizza Order Form (POF) is to give the user an elegant, easy to understand and use graphical interface for ordering pizzas from a shop online.

This was our first task and was an introduction into designing and creating a graphical interface for users to interact with. In the process of creating a pizza order form, we learnt about when to use and not to radio buttons, drop down menus, numeric steppers, text fields and buttons. Such as if you have to pick a state in the US, it will take up too much space on the screen to fill with radio buttons, so using a drop down menu will save screen real estate and look more elegant.

## How did my product achieve this purpose?

My product achieves the purpose of giving the user an easy to understand and use graphical interface by having the order form vertical and not so spaced out. By having the form be vertical, it makes the user work from top to bottom, which means that most of the time, the user will see all the important elements of the form and work through it methodically. It is important to have the form set out vertically so that the user will not accidentally skip parts and get lost, which would cause frustration to the user and may result in the loss of a (potential) customer. These types of vertically set forms are found in many places on the Internet, such as at <http://www.vfs-in-au.net/> where there are many inputs that must be filled.

## What improvements could I make to this application?

To improve this application, I could add a cart, which would allow the user to make just one order instead of multiple whenever he/she wants to order more than one type of pizza. This would make the process of ordering pizzas much less of a hassle and be more convenient for the user; this is because currently the user must purchase each type of pizza separate from the other types of pizzas. I think I could also add more types of pizzas as options for the user. Finally I could format the output better so that it is easier to view and understand.

# 8.2.0: Project 2 – Pong

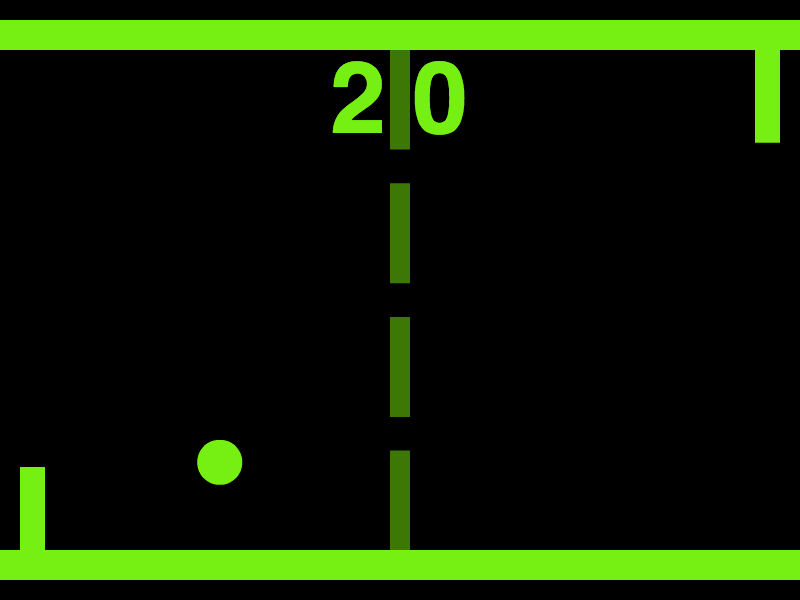
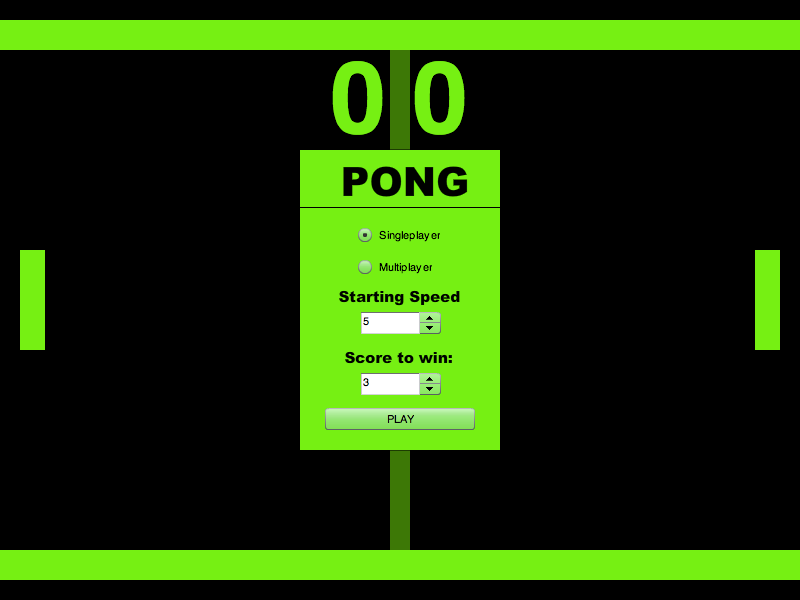


Figure 4: Pong main screen.

Figure 4: Pong game play.

Figure 4: Pong game over.

## What is the purpose of this product?

The purpose of this product is for it to be an entertainment product and to entertain users with the simple game of pong. This product should be able to play both single and multiplayer. The Pong program should be able to keep score and to decide a winner based on the score. I have added to this program extra functions for it to be able to take user input to let the user set a starting speed for the ball and a score that either the user or the computer must achieve to win. In the process of creating this pong game, we practically learnt about operators and algorithms, data coding, and data types.

## How did my product achieve this purpose?

My product achieved this purpose by presenting the classic game of pong to the user. My pong game also uses the same classic green on black colour scheme, which will present the user with a very familiar game. It is an easy to play game controlled simply by the up and down arrows on the keyboard. For multiplayer the second player can control his/her paddle with the ‘A’ and ‘Z’ keys. In the program we had to use our knowledge from data coding to recognize the keystrokes for the up and down arrow keys and the A and Z keys. Our knowledge from the operators and algorithms was also used to move the ball around and do hit tests and fix glitches in the game. I also used my knowledge from the theory DataTypes activity to create the correct variable types for the program.

## What improvements could I make to this application?

I think I could improve on this program, I think I could add difficulty levels to the AI so that it can be more playable by a larger audience. I also think I could add my personal touch of being able to customize the visuals of the game a lot, such as the colours for each of the user interface components. This will give the user more visual freedom and make a more personalized program. Also by adding difficulty levels the program can cater for young beginner players to older more experienced players. This idea works for almost all games such as Call of Duty or any other big titles. Some games will adjust the difficulty by changing the speed or rate, size or accuracy of enemies or other aspects of the game.

# Macintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Evaluation:Screenshots:Project 3:User Docs.pngMacintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Evaluation:Screenshots:Project 3:Running.pngMacintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Evaluation:Screenshots:Project 3:Nothing.png8.3.0: Project 3 – Spritz

Figure 8: Spritz user doc.

Figure 7: Spritz in use.

Figure 7: Spritz main screen.

## What is the purpose of this product?

The purpose of this product is to parse the text inputted by the user and to present the words one by one at a user settable speed so that the user can ‘speed read’. This product should make the centre character red to keep the locus of the word. This program is supposed be as similar as it can to the actual Spritz program on the Internet without infringing on any of its patents. In this program we learnt about subprograms and languages along with syntax and error detection and testing.

## How did my product achieve this purpose?

My Spritz app achieves the above purpose by parsing the text by finding the spaces and separating the words and placing them into an array. If the app finds punctuation in the text, it will add it to the array twice so that the words with punctuation attached to them will be displayed for twice as long. Then when the user clicks the read button, the program will take the value speed value that the user has set and creates a repeating timer to call a method to step through the array of words the subprogram will then call another subprogram. The called subprogram will then find the length of the word it will display and find the middle letter. It will then parse the word into the first half, middle letter and second half, where it will turn the middle letter into HTML text to make it red, after which, the program will concatenate the parts of the word together and display it in the label as html text.

## What improvements could I make to this application?

To improve on this app, I could make it so that the red letter is always centered in the label, regardless of the size of each letter in the font, which would keep the locus of the word in place and let the user speed read with more efficiency. I could also change the way that the reading speed is set, instead of setting a value between one and ten (where ten is the fastest), the app could take values such as 300 words per second so that user can have a more precise value for the speed that he/she can read at. By doing the above, the user would get a more seamless experience and will likely reuse the program.

# 8.1.1 - Types of Languages

## Satisfactory

Imperative: In computer science, imperative programming is a programming paradigm that describes computation in terms of statements that change a program state. In much the same way that imperative mood in natural languages expresses commands to take action, imperative programs define sequences of commands for the computer to perform. (Wikipedia, 2014)

Declarative: In computer science, declarative programming is a programming paradigm, a style of building the structure and elements of computer programs, which expresses the logic of a computation without describing its control flow. (Wikipedia, 2014)

Functional: In computer science, functional programming is a programming paradigm, a style of building the structure and elements of computer programs, which treats computation as the evaluation of mathematical functions and avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state—i.e. pure mathematical functions. It is a declarative programming paradigm, which means programming is done with expressions. (Wikipedia, 2014)

Logic: Logic programming is a programming paradigm based on formal logic. Programs written in a logical programming language are sets of logical sentences, expressing facts and rules about some problem domain. Together with an inference algorithm, they form a program. Major logic programming languages include Prolog and Data log. (Wikipedia, 2014)

Procedural: Procedural programming can sometimes be used as a synonym for imperative programming (specifying the steps the program must take to reach the desired state), but can also refer (as in this article) to a programming paradigm, derived from structured programming, based upon the concept of the procedure call. Procedures, also known as routines, subroutines, methods, or functions (not to be confused with mathematical functions, but similar to those used in functional programming), simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or its self. Procedural programming is a list or set of instructions telling a computer what to do step by step and how to perform from the first code to the second code. (Wikipedia, 2014)

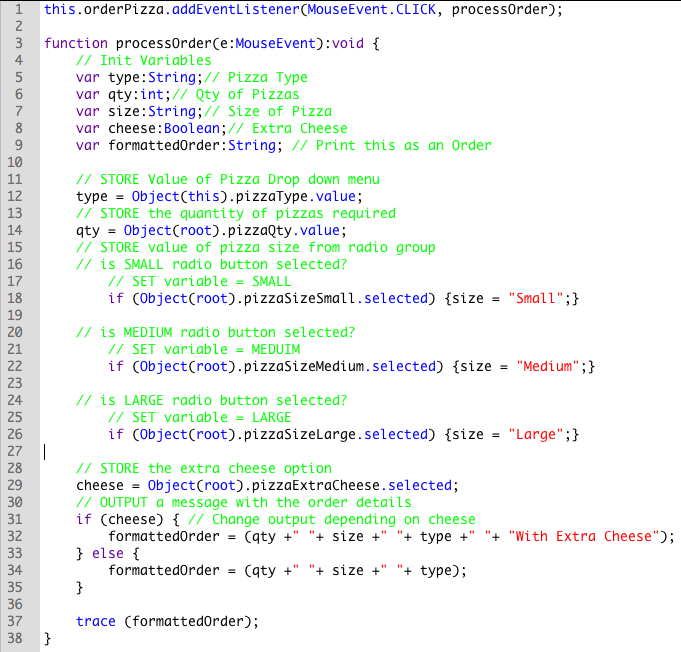
Event Driven: In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads. Event-driven programming is the dominant paradigm used in graphical user interfaces and other applications (e.g. JavaScript web applications) that are centered around performing certain actions in response to user input.

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected. In embedded systems the same may be achieved using hardware interrupts instead of a constantly running main loop. Event-driven programs can be written in any programming language, although the task is easier in languages that provide high-level abstractions, such as closures. (Wikipedia, 2014)

Object-Orientated: Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs. C++, Objective-C, Smalltalk, Java, C#, Perl, Python, Ruby and PHP are examples of object-oriented programming languages. (Wikipedia, 2014)

## Commendable

Figure 10: Code for the pizza order form.



Procedural Programming: This program uses procedural programming because it goes through a set order of steps to get the job done.

Object Orientated Programming: This is partially object orientated because it takes advantage of UI objects such as dropdown menus, radio buttons, checkboxes and a button. It accesses data off of the objects classes.

Event Driven Programming: This is event driven programming because it makes use of user events and event listeners to make the program.

## Exceptional

this.orderPizza.addEventListener(MouseEvent.CLICK, processOrder);

function processOrder(e:MouseEvent):void {

// Init Variables

var type:String;// Pizza Type

var qty:int;// Qty of Pizzas

var size:String;// Size of Pizza

var cheese:Boolean;// Extra Cheese

var formattedOrder:String; // Print this as an Order

// STORE Value of Pizza Drop down menu

type = Object(this).pizzaType.value;

// STORE the quantity of pizzas required

qty = Object(root).pizzaQty.value;

// STORE value of pizza size from radio group

// is SMALL radio button selected?

// SET variable = SMALL

if (Object(root).pizzaSizeSmall.selected) {size = "Small";}

// is MEDIUM radio button selected?

// SET variable = MEDUIM

if (Object(root).pizzaSizeMedium.selected) {size = "Medium";}

// is LARGE radio button selected?

// SET variable = LARGE

if (Object(root).pizzaSizeLarge.selected) {size = "Large";}

// STORE the extra cheese option

cheese = Object(root).pizzaExtraCheese.selected;

// OUTPUT a message with the order details

if (cheese) { // Change output depending on cheese

formattedOrder = (qty +" "+ size +" "+ type +" "+ "With Extra Cheese");

} else {formattedOrder = (qty +" "+ size +" "+ type);}

trace (formattedOrder);

}

The code is Imperative because it has a workflow.

Firstly: store all the data inputted by the user.

Secondly: format a string to be easily human readable.

Finally: print the string so the store attendant can read it or process the price of each item etc.…

# 8.1.2 - Variables

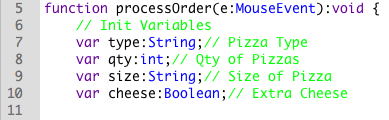
## Satisfactory

A variable is a space in memory where data such as numbers and strings are saved under an identifier name.

It is important to declare variables when programming because variables can be modified to suit the needs of a program. Without variables a program would not be much of anything at all.

Figure 11: Declaring variables.

## Commendable



Declaring the variable

Name the variable

Set the type of variable

Set a value to the variable

## Exceptional

Rules for naming thing in code:

|  |  |  |
| --- | --- | --- |
| Identifier Type | Rules for naming | Examples |
| Classes | Class names should be nouns in Upper[CamelCase](http://en.wikipedia.org/wiki/CamelCase), with the first letter of every word capitalized. Use whole words — avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). | class Raster;  class ImageSprite; |
| Methods | Methods should be verbs in lower[CamelCase](http://en.wikipedia.org/wiki/CamelCase) or a multi-word name that begins with a verb in lowercase; that is, with the first letter lowercase and the first letters of subsequent words in uppercase. | run();  runFast();  getBackground(); |
| Variables | Local variables, instance variables, and class variables are also written in lower[CamelCase](http://en.wikipedia.org/wiki/CamelCase). Variable names should not start with underscore (\_) or dollar sign ($) characters, even though both are allowed. This is in contrast to other [coding conventions](http://en.wikipedia.org/wiki/Coding_conventions) that state that underscores should be used to prefix all instance variables.  Variable names should be short yet meaningful. The choice of a variable name should be [mnemonic](http://en.wikipedia.org/wiki/Mnemonic) — that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters. | int i;  char c;  float myWidth; |
| Constants | Constants should be written in uppercase characters separated by underscores. Constant names may also contain digits if appropriate, but not as the first character. | final static int MAX\_PARTICIPANTS = 10; |

(Wikipedia, 2014)

# 8.1.3 - GUI Concepts

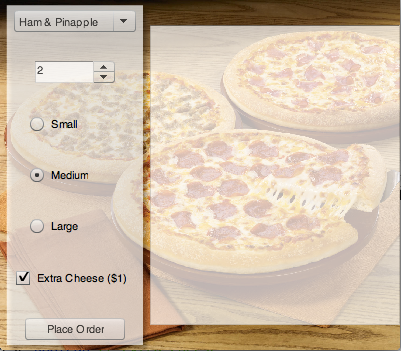
## Satisfactory

In computing, graphical user interface (GUI, sometimes pronounced "gooey" or "gwee") is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLI), which require commands to be typed on the keyboard. (Wikipedia, 2014)

The person that invented the idea of a GUI was Doug Engelbart. His work later went on to create the world’s first computer with a GUI, which was made by a company called Xerox PARC in Palo Alto. (Wikipedia, 2014)

The GUIs made computers almost infinitely more usable to the average Joe and children, because before the GUI the only interface a computer had was its command line. Now that there is a GUI more people use computers and get more work done.

## Commendable



This is a numeric stepper, which is used to determine how many pizzas the user would like to add to an order.

A stepper is used to increment up or down a numerical value to a maximum and a minimum set by the programmer.

This is a set of radio buttons. This is used to let the user select a pizza size for his/her order.

Radio Buttons are used when the user may only select one option out of a select few.

This is a button and the user presses it when he/she completes filling in their order.

In the program this activates an event/function/method that processes the above information inputted by the user.

This is a check box, which is used to give the user the option to order with extra cheese.

This is a simple visual tick box, which returns a True or False value in the program.

This is a drop down menu and is where the user selects the pizza he/she would like to order.

Dropdown menus are usually used when there are more than 5 items in a list that a user must select from.

Figure 12: GUI of the pizza order form.

## Exceptional

Figure 13: Example of a poor UI.



Very crowded UI, no clear flow for the user to follow, too much information in one page, too many options on screen.

# Macintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Evaluation:Screenshots:8.1.4 (Design your own GUI):With My Info.pngMacintosh HD:Users:38559:Documents:Year 10 2014:IST:Sem 1 Portfolio:Resources:Evaluation:Screenshots:8.1.4 (Design your own GUI):Main.png8.1.4 – Create your own GUI

Figure 14: What the GUI tool will look like when used.

Figure 14: GUI tool to collect user information (Unused)

# 8.2.1 - Data Types (8.2.1)

## Elementary

Data type: a particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it. (Grover, 2011)

### 6 Most Common Data Types

1. Characters
2. Strings
3. Integers
4. Real Numbers
5. Booleans
6. Date and Time

## Basic

### Strings

(Action Script 3) var stringValue:String = “This is a string”;

Strings can be used to output to text fields to display information (formatted/non-formatted).

### Integers

(Action Script 3) var intValue:int = 1998;

Integers can be used to do simple whole number calculations within the program.

### Real Numbers

(Action Script 3) var numberValue:Number = 1998.0225

Real Numbers can be used to do more complex mathematical calculations that require more precision.

### Booleans

(Action Script 3) var booleanValue:Boolean = true;

Booleans can be used in if statements to determine whether or not to do something in a program.

## Sound

Figure 14: Data types (Types of variables).

Integers are used to keep track of the left and right player score. Scores are a whole number in this game so integers are perfect for this.

A Boolean has been used to store whether or not the game has ended.

|  |  |  |
| --- | --- | --- |
| Data Name | Data Type | Justification |
| Name | String | Names are a sequence of characters (ASCII). |
| Age | Integers | An age is always an integer. |
| Email | String | An email is a sequence of characters. |
| Image | Bitmap | Bitmap maps the bits of the image. |
| Phone Number | String | A phone number is a string because it can have leading 0s or country codes that begin with a + |

## Thorough

## Extensive

By changing the data type of the variable scrollSpeed from an integer to a number the program is able to store the decimal values of the variable thrust that are added onto the scrollSpeed.

As integers cannot store decimal values, when given the value 0.5 the flash compiler floors the value to 0 so nothing is being added to the scrollSpeed.

When a Number is used the value 0.5 does not get floor but is stored as 0.5.

# Macintosh HD:Users:38559:Documents:Year 10 2014:IST:8.2.x:8.2.2:ASCII Table.pdf8.2.2 – Data Coding

# 8.2.3 - Operators and Algorithms

## Elementary & Basic



Binary Selection

Greater than

Add

Less than

Minus

Multiply

Nested Binary Selection

Figure 15: Operators and algorithms

## Sound

Ball hit right paddle

Start

xSpeed > 0

xSpeed \*= -1;

xSpeed -= ballSpeedUpFactor;

maxPlayerSpeed += ballSpeedUpFactor;

ySpeed > 0

ySpeed += ballSpeedUpFactor/2

ySpeed < 0

ySpeed -= ballSpeedUpFactor/2

Play Quack Sound

End

T

T

T

T

F

F

F

F

## Thorough

### Pseudo Code

BEGIN

IF xSpeed > 0 THEN

IF ball.hitTestObject(rightPaddle)) THEN

xSpeed \*= -1;

xSpeed -= ballSpeedUpFactor;

maxPlayerSpeed += ballSpeedUpFactor

IF ySpeed > 0 THEN

ySpeed += ballSpeedUpFactor/2;

END IF

IF ySpeed < 0 THEN

ySpeed -= ballSpeedUpFactor/2;

END IF

Play Quack Sound

END IF

END IF

END

## Extensive

I chose to nest the right paddle hit test inside a binary selection, which only returns true when the ball’s xSpeed is greater than 0. This is important because it means that the hit test only happens when the ball is moving towards the right side of the screen. This fixes the glitch where the ball gets stuck in the paddle bouncing back and forth. THEN if the if statement which tests the right paddle hit test returns true, the data operator \*= -1 will reverse the direction which the ball if moving.

# 8.3.1 - Subprograms and Languages

## Elementary

Syntax is a structure of statements in a programming language. The syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language. This applies both to programming languages, where the document represents source code, and mark-up languages, where the document represents data. The syntax of a language defines its surface form. Text-based computer languages are based on sequences of characters, while visual programming languages are based on the spatial layout and connections between symbols (which may be textual or graphical). Documents that are syntactically invalid are said to have a syntax error. (Wikipedia, 2014)

No, the syntax of programming languages can differ between programming languages. Such as in Python the way variable are declared is different from Objective-C or Action Script 3. E.g.: (Python) x = 1; (Objective-C) int x = 1; or (Action Script 3) var x:int = 1;.

5 Programming Languages are:

1. Pizza
2. Lilypond
3. Squirrel
4. Python
5. Dog

(Wikipedia, 2014)

## Basic

Object code: Object code is a sequence of statements or instructions in a computer language, usually a machine code language (i.e. 1s and 0s) or an intermediate language such as RTL. (Wikipedia, 2014)

Compiler: A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language. The most common reason for wanting to transform source code is to create an executable program. (Wikipedia, 2014)

Interpreter: In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language. (Wikipedia, 2014)

## Sound

Subprogram: In computer programming, a subprogram is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can be used in programs wherever that particular task should be performed. Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages a subprogram may be called a procedure, function, routine, method, or a subroutine. (Wikipedia, 2014)

Subprograms are used in programs to make the program more efficient and the to shorten the time taken to write a program. Instead of rewriting the same code every time it is needed in a program, it would be faster and easier to make that code a subprogram and *call* it whenever it is needed.

The shape for a subprogram in a flowchart is a double lined rectangle (below).

# Thorough

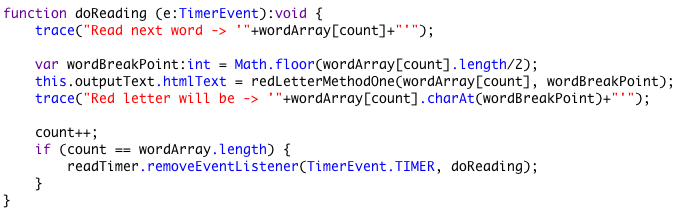
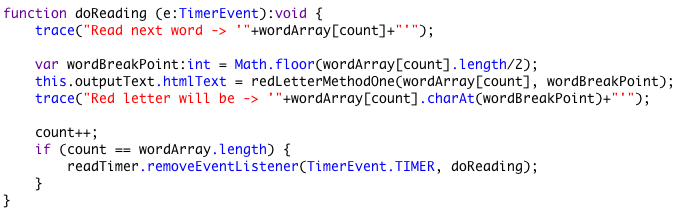
Parameter: numerical or other measurable factor forming one of a set that defines a system or sets the conditions of its operation.

Figure 16: Screenshot of a subprogram in Action Script 3.

Figure 17: Example of parameters in subprograms.



Parameter: e is a Timer Event.

## Extensive

Code for multiplying 2 random numbers between 1- 100.

var a:int = Math.floor(Math.random()\*100+1);

var b:int = Math.floor(Math.random()\*100+1);

multiplyNumbers(a, b);

function multiplyNumbers(value1:int, value2:int):int {

var product:int = value1\*value2;

trace(value1+" \* "+value2+" = "+product);

return product;

}

# 8.3.2 - Testing and Error Detection and Correction

## Elementary

The purpose of program testing is to test the code and to find any errors in the code, which cause unexpected results in the program. (Grover, 2011)

Unit testing is usually done by the programmer first, but may also be tested later by a software tester (someone with specific skills in program testing). To do this, the programmer will run the program using a set of test data. The programmer should already know what the program will do with each of the sets of data. The programmer compares the actual program output with the expected output. If the two sets of data don’t match, the programmer then attempts to find the cause for the difference, which may be due to program errors. (Grover, 2011)

## Basic

1. Syntax Errors: Like grammatical errors in code, such as forgetting a curly bracket. (Grover, 2011)
   1. Objective-C

NSString \*string = “This is a string”;

Error: Missing @ symbol before “This is a string”. Should be @”This is a string”.

* 1. Objective-C

for(int i = 0; i < 10; i++{

NSLog(@”%i”, i);

}

Error: Missing Right Parenthesis

1. Logical Errors: A miss understanding of what has to be done by the programmer, such as using an incorrect algorithm. (Grover, 2011)
   1. Multiplying when trying to divide (Objective-C)

int value1 = 50;

int value2 = 2;

NSLog(@”%i”, value1\*value2);

Expecting the output of 25 but the actual output is 100.

* 1. Forgetting a statement (Objective-C)

-(void)withdrawAmount:(float)amount {

float currentBalance = [self getCurrentBalance];

currentBalance -= amount;

[self spitOutMoney];

}

Forgot to update the balance of the account. The user can withdraw all the money he wants without losing any.

1. Run-time Errors: Errors that only show up when the program is executing (running). (Grover, 2011)
   1. Division by 0 (Objective-C)

float value = 1998/0;

NSLog(@”%f”, value);

Error: Illegal operation, division by zero.

Expecting to receive data of a specific type but receiving data of a type, which does not match. (Objective-C)

-(NSArray \*)statesInAustralia {

return @[@"ACT", @"NSW", @"NT", @"QLD", @"SA", @"TAS", @"VIC", @"WA"];

}

-(void)getStatesInAustralia {

NSString \*states = [self statesInAustralia];

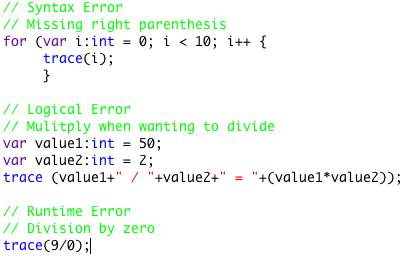
}

Error: Illegal conversion of NSArray to NSString.

## Sound

Breakpoints: In software development, a breakpoint is an intentional stopping or pausing place in a program, put in place for debugging purposes. It is also sometimes simply referred to as a pause. When a breakpoint is hit, various tools are used to inspect the state of the program or alter it. Stack trace of each thread may be used to see the chain of function calls that led to the paused instruction. A list of watches allows one to view the values of selected variables and expressions. There may also be tools to show the contents of registers, loaded program modules and other information. Many processors include hardware support for breakpoints (typically instruction and data breakpoints). As an example, the x86 instruction set architecture provides hardware support for breakpoints with its x86 debug registers. Such hardware may include limitations, for example not allowing breakpoints on instructions located in branch delay slots. This kind of limitation is imposed by the microarchitecture of the processor and varies from processor to processor. Without hardware support, debuggers have to implement breakpoints in software. (Wikipedia, 2014)

## Thorough



Syntax Error**:** Missing right parenthesis to close the parameters for the *for* statement.

This is a good example because forgetting to place parenthesis is a very common mistake amongst starting programmers.

Logical Error**:** Accidentally multiplying instead of dividing. This is a good example because firstly it was in the textbook, and because it can happen due to pure carelessness and may be hard to find in the code.

Runtime Error**:** Accidentally dividing by zero. This is a good example because it may happen at the end of a series of math statements or generating random numbers, which include zero as an output.

Figure 18: Examples of errors in Action Script 3.

## Extensive

Desk check: (Campion College, 2014)

Input(s): Number(s) = 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Line Number | Number | Count | Temp | Conditions | Input/Output |
| 1 | 4 |  |  |  | Input: Number = 4 |
| 2 |  | 0 |  |  |  |
| 3 |  |  |  | 0 < 4 ? is T |  |
| 4 |  |  | 0+1 = 1 |  |  |
| 5 |  |  |  |  | Output: Temp = 1 |
| 6 |  | 1 |  |  |  |
| 3 |  |  |  | 1 < 4 ? is T |  |
| 4 |  |  | 1+1 = 2 |  |  |
| 5 |  |  |  |  | Output: Temp = 2 |
| 6 |  | 2 |  |  |  |
| 3 |  |  |  | 2 < 4 ? is T |  |
| 4 |  |  | 2+1 = 3 |  |  |
| 5 |  |  |  |  | Output: Temp = 3 |
| 6 |  | 3 |  |  |  |
| 3 |  |  |  | 3 < 4 ? is T |  |
| 4 |  |  | 3+1 = 4 |  |  |
| 5 |  |  |  |  | Output: Temp = 4 |
| 6 |  | 4 |  |  |  |
| 3 |  |  |  | 4 < 4 ? is F |  |
| 7 |  |  |  |  | Output: Count = 4 |

The output if the user enters 4 the output will be as follows:

Temp = 1

Temp = 2

Temp = 3

Temp = 4

Count = 4

# Bibliography

Campion College. (2014, March 6). *Desk Check Guide*. Retrieved May 25, 2014, from sites.google.com: https://sites.google.com/a/campioncollege.com/it\_eveningschoool/problem-solving-and-programming/desk-check-guide

Grover, D. (2011). *Information and software technology.* (B. James, Ed.) Melbourne, Victoria, Australia: Pearson Australia.

Wikipedia. (2014, March 27). *Breakpoint*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Breakpoint

Wikipedia. (2014, May 25). *Compiler*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Compiler

Wikipedia. (2014, March 28). *Declarative Programming*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Declarative\_programming

Wikipedia. (2014, April 17). *Event-driven Programming*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Event-driven\_programming

Wikipedia. (2014, May 15). *Functional Programming*. Retrieved May 25 2014, from Wikipedia: http://en.wikipedia.org/wiki/Functional\_programming

Wikipedia. (2014, May 8). *Graphical user interface*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Graphical\_user\_interface

Wikipedia. (2014, April 24). *Imperative Programming*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Imperative\_programming

Wikipedia. (2014, May 20). *Interpreter (computing)*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Interpreter\_(computing)

Wikipedia. (2014, May 23). *List of programming languages*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/List\_of\_programming\_languages

Wikipedia. (2014, May 15). *Logic Programming*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Logic\_programming

Wikipedia. (2014, February 19). *Naming convention (programming)*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Naming\_convention\_(programming)

Wikipedia. (2014, May 19). *Object code*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Object\_code

Wikipedia. (2014, May 25). *Object-oriented Programming*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Object-oriented\_programming

Wikipedia. (2014, February 8). *Procedual Programming*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Procedural\_programming

Wikipedia. (2014, April 4). *Subroutine*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Subroutine

Wikipedia. (2014, May 24). *Syntax (programming languages)*. Retrieved May 25, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Syntax\_(programming\_languages)