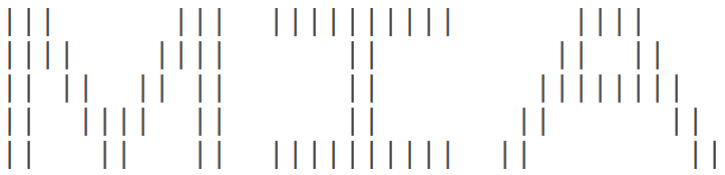

Multiple Integrated Applications (MIA)
Manual & Programming Documentation

Developer: Antonius Torode
Michigan State University
Department of Physics & Astronomy

Version – 0.025
Latest update: March 29, 2018

The Original Maintainer of this work is: Antonius Torode.

The Current Maintainer of this work is: Antonius Torode.

This document is designed for the sole purpose of providing documentation for Multiple Integrated Applications (MIA), a program written for and created for personal use. Throughout this document, "MIA" will be used as a reference to "Multiple Integrated Applications." This acronym is solely designed for use in conjunction with the program and was originally created by the creator of this document. MIA is designed to be used for multiple purposes based on the continual addition of functionality. It can be adapted to work in other situations and for alternate purposes.

This document is continuously under development.
Most Current Revision Date: March 29, 2018

Torode, A.
MIA Manual.
Michigan State University –
Department of Physics & Astronomy.
2017, Student.
Includes Source Code and References

Contents

This page intentionally left blank.
(Yes, this is a contradiction)

Chapter 1

Multiple Integrated Applications (MIA)

MIA is designed to be a collection of scripts, tools, programs, and commands that have been created in the past and may be useful in the future. It's original idea was a place for the original author to combine all of his previous applications and codes into one location that can be compiled cross platform. MIA is written in C++ but will contain codes that were originally designed in C#, Java, Python, and others. MIA is created for the authors personal use but may be used by others if a need or desire arises under the terms of Antonius' General Purpose License (AGPL).

The MIA acronym was created by the original author for the sole purpose of this application. The design of MIA is a terminal prompt that accepts commands. There are no plans to convert MIA into a GUI application as there is currently no need; however, some elements may be programmed in that produce a GUI window for certain uses such as graphs. The MIA manual is designed to be an explanation of what MIA contains as well as a guide of how to utilize the MIA program to it's fullest.

As MIA is continually under development, this document is also. Due to this, it may fall behind and become slightly outdated as I implement and test new features into MIA. I will attempt to keep this document up to date with all of the features MIA contains but I can only do so if time permits.

Chapter 2

MIAConfig file

2.1 MIAConfig Purpose Introduction

2.2 MIAConfig Usage

Chapter 3

Commands and Syntax

3.1 Valid Syntax

MIA is designed to be used similar to a terminal or command prompt. One enters commands and the uses the 'Enter' key to perform the commands. MIA commands are NOT case sensitive. By default, all commands are changed to lower case before executing through the MIA program. If a command is not typed exactly how it is intended (including spaces and newline characters) it may not execute.

3.2 Complete List of Valid Commands (CLVC)

```
help
```

Displays a valid lists of commands and a brief description to go along with each.

```
add
```

Adds two positive integers of any length. This adds two strings together using a similar algorithm one would when adding large numbers by hand. It is possible to get results by entering non-number entries but will serve no significance due to the way MIA internally converts strings to integers by shifting the ASCII values.

```
button spam
```

Spams a specified button (key press). This function asks for a key input as well as asks for a number of times the user would like the button spammed. Not all keys are programmed in and the time between button spamming is currently fixed (this will be updated at a later time). This function currently only works on Windows OS.

```
button spam -t
```

Does the same as the "button spam" command but also simulates the tab key in between each key press.

```
collatz
```

Produces a collatz sequence based on a specified starting integer. This method uses the login data type which means if a number of the sequence extends the storage of a long, the results will become untrustworthy.

```
config
```

Reloads the MIAConfig.txt file and prints the variables.

```
crypt -d0s1
```

Encrypts a string using the d0s1 algorithm. This is explained more in chapter **??**.

```
crypt -d0s2
```

Encrypts a string using the d0s2 algorithm. This is explained more in chapter **??**.

```
decrypt -d0s1
```

De-crypts a string using the d0s1 algorithm. This is explained more in chapter **??**.

```
decrypt -d0s2
```

De-crypts a string using the d0s2 algorithm. This is explained more in chapter **??**.

```
digitsum
```

Returns the sum of the digits within an integer of any size. Similar to the add command, this converts a string to an array of integers using ASCII shifting and then sums the values together. Due to this, you can also find values for entering non-numerical strings.

```
error info
```

Returns information regarding an error code.

```
error info -a
```

Returns information regarding all error codes.

```
eyedropper
```

Returns the RGB value of the pixel located at the cursor.

```
factors
```

Returns the number of factors within an integer. The integer must be smaller than C++'s internal storage for the long data type.

```
find mouse
```

This function will locate the position of the usere mouse pointer after 5 seconds and print the coordinates it is located at.

```
fishbot
```

A working and configurable WoW fishbot.

```
lattice
```

Returns total lattice paths to the bottom right corner of an n x m grid. This function is only valid for situations in which the answer will not exceed the internal storage of a long data type.

```
mc dig
```

Simulates key strokes for continuous Minecraft diggigg. This function will press and hold the w key and the left mouse button for forward momentum whilst digging in Minecraft. This function currently only works on Windows OS.

```
mc explore
```

Explores a Minecraft map using /tp. This is for server exploration given someone with /tp power and creative mode. You can enter a range of coordinates and MIA will emulate the keystrokes to /tp over a large area in order to generate the map. This is handy when using mc server plugins such as dynmap which will display explored map areas via a web browser. This function also asks for the user to specify a time between each /tp so that it can be adapted for use on both fast and slow servers/computers. This function currently only works on Windows OS.

```
multiply
```

Multiplies two integers of any length. Similar to add, this multiplies two strings together using a similar algorithm one would when multiplying large numbers by hand. It is possible to get results by entering non-number entries but will serve no significance due to the way MIA internally converts strings to integers by shifting the ASCII values.

```
palindrome
```

Determines if a positive integer is palindrome. The integer must be smaller than C++'s internal storage for the long data type.

```
prime
```

Determines if a positive integer is prime or not. The integer must be smaller than C++'s internal storage for the long data type.

```
prime -help
```

Displays help defailts for prime functions.

```
prime -f
```

Determines all of the prime factors of a positive integer. The integer must be smaller than C++'s internal storage for the long data type.

```
prime -n
```

Calculates the n'th prime number up to a maximum number of 2147483647.

```
prime -n -p
```

Creates a file of all prime numbers up to a maximum number of 2147483647.

```
prime -n -c
```

Clears the file created by 'prime -n -p'.

```
quadratic form
```

Calculates the solution to an equation of the form $ax^2+bx+c=0$. This function accounts for imaginary answers.

```
subtract
```

Finds the difference between two integers of any length. Similar to add, this subtracts two strings together using a similar algorithm one would when subtracting large numbers by hand. It is possible to get results by entering non-number entries but will serve no significance due to the way MIA internally converts strings to integers by shifting the ASCII values.

```
randomFromFile
```

Prints a number of random lines from a text file. This will read in each line from a text file and print a user specified number of the lines by choosing them randomly.

```
triangle
```

Determines if a number is a triangle number or not. The integer must be smaller than C++'s internal storage for the long data type.

```
workout
```

Generates a workout from the values defined in workouts.txt. See section 4 for more information.

```
workout -w
```

Generates a weeks worth of workouts from the values defined in workouts.txt. This generation outputs to workout.txt found in the Output files folder. See section 4 for more information.

```
wow dup letter
```

Duplicates a letter in WoW a specified number of times. This will simulate entering a recipient, subject, and then pasting a message into the body followed by hitting the send button through the in game mailbox on World of Warcraft. The user specifies needed information and number of letters to send. This function is useful for RP scenarios.

```
exit
```

Quits MIA.

Chapter 4

Workout Generation

4.1 MIA Workout Generation Overview and Introduction

The workout generation in MIA is created for producing a workout with customization from the user. The generation of a workout within MIA has some dependencies on random value generation and thus is capable of creating different workouts each run. MIA is also capable of creating an entire weeks worth of workouts and outputting it to a file for the user. The entire generation process depends on a few input values, such as maximum number of sets, maximum number of exercises per set, and more which are all defined in a exercises file. Upon running the workout generation, the user enters a difficulty and MIA generates a workout with appropriate difficulty based on this number and the input file (see section 4.3 for more details).

Throughout this section, workout can be defined as the complete output generated by MIA containing some number of sets, some number of exercises per set, and some number of reps per exercises.

4.2 Input File and Defining Workouts

The MIA workout generation utilizes an input file to determine exercises, exercise weighted values and various generation values. By default, this file is located in ../bin/Resources/InputFiles/exercises.txt but this file name and path can be changed via the MIAConfig file (see section 2 for more details). The contents of this input file look similar to the following.

```
#============================================================================
# Name        : exercises.txt
# Author      : Antonius Torode
# Date        : created on 3/14/18
# Copyright   : This file can be used under the conditions of Antonius'
#               General Purpose License (AGPL).
# Description : Different workouts with weighted values for workout generation.
#============================================================================

# Various comments blocks explaining usage.
# ...
# ...
toughness = 0.1
minNumOfExercises = 3.0
maxNumOfExercises = inf
minNumOfSets = 1.0
maxNumOfSets = inf

# Exercises and weights.
push_up = 8.0; reps
sit_up = 15.0; reps
pull_up = 0.75; reps
```

```
squat = 3.0; reps
jumping_jack = 30.0; reps
```

This file must be in the correct format in order for the MIA workout generation to function properly. First, commented lines are created using the '#' character. These lines are ignored by MIA when running internal algorithms. Within this input file, spaces are not important. Upon initialization, the MIA program will ignore all spaces within this file. Next, there are a few variables that the user can customize and define within this file which are below.

```
toughness = 0.1
minNumOfExercises = 3.0
maxNumOfExercises = inf
minNumOfSets = 1.0
maxNumOfSets = inf
```

These values must appear in the input file before any defined exercises. To begin, toughness is a global variable that helps define the number of reps MIA will output per workout chosen. Increasing this value is a global increase to the workout generation difficulty. The default value for toughness is 0.1 (see section 4.3 for more details). Next, There are minNumOfExercises and maxNumOfExercises variables which are used to determine the minimum number of exercises MIA will choose per set and the maximum number of exercises MIA can choose per set. The maxNumOfExercises value is read in such that a value of 'inf' is allowed. If 'inf' is selected, MIA will set the total number of exercises within the input file to be the maximum. Similarly, there are minNumOfSets and maxNumOfSets values which work in identical ways to minNumOfExercises and maxNumOfExercises only defining a minimum and maximum for number of sets per generated workout instead of number of exercises per set.

Note. At the time of writing this, the MIA program is not designed to account for a value of maxNumOfExercises that is larger than the actual number of exercises defined in the input file.

Following these program variables, the main part of the input file is the defined exercises. The exercises are defined similar to below.

```
# Exercises and weights.
push_up = 8.0; reps
sit_up = 15.0; reps
pull_up = 0.75; reps
squat = 3.0; reps
running = 0.1; mile
jumping_jack = 30.0; reps
```

Each exercise is defined using a common form. As shown below, the line must begin with an exercise name. Following this comes an equal sign and then a weighted value. This weighted value is defined to be relative to all other weighted values. This mean that in the above example, the file is claiming 8.0 push ups are equivalent to 15.0 sit ups, and similarly, 0.75 pull ups, etc. The MIA program will assume and each weight value for each exercise is of the same real world difficulty to the user. Following the weighted value must come a semi-colon and then a unit. The equal sign and semi-colon are important because they define how MIA separates the values. As stated previously, spaces are irrelevant in these definitions (See below).

```
# Proper format for definind an exercise in the input file.
exercise_Name = exercise_Weighted_Value; exercise_Unit

#The following three lines are equivalent when read in by MIA.
pull ups = 15.0; reps
pullups=15.0;reps
p   ull u ps   =   15  . 0   ; reps
```

4.3 Generation Algorithm

This section contains an outline of the algorithm used to generate the MIA workouts. The MIA generation is based on creating two curves (maximum and minimum) for a parameter and then deciding upon which parameter to use for a workout by taking a random value between both curves. For the purposes of this section, we will denote a random number between two values q_1 and q_2 as $rand[q_1, q_2]$. We will denote a complete workout with W.

4.3.1 Number of Sets Per Workout

The number of sets, denoted $S(s_{min}, s_{max}, d) \equiv S$ is dependent on three variables. The first two are from the input file which are minNumOfSets, denoted s_{min} and minNumOfSets, denoted s_{max}. The last is the difficulty d which is input by the user upon generation. The maximum number of sets was originally based on a linear increase, however for better optimization of the real world workout difficulties, a custom algorithm was created. The maximum $S_{max}(s_{min}, s_{max}, d)$ and minimum $S_{min}(s_{min}, s_{max}, d)$ number of sets per workout are given by

$$S_{max}(s_{min}, s_{max}, d) \equiv S_{max} = \frac{s_{max} - s_{min}}{10^{4/3}} d^{2/3} + s_{min} \tag{4.1}$$

$$S_{min}(s_{min}, s_{max}, d) \equiv S_{min} = \frac{s_{max} - 1.9 \times s_{min}}{1.9 \times 10^{4/3}} d^{2/3} + s_{min}. \tag{4.2}$$

Thus since $S(s_{min}, s_{max}, d)$ is a random value between these curves, we have

$$S_{ave}(s_{min}, s_{max}, d) \equiv S_{ave} = \frac{S_{max} + S_{min}}{2} \tag{4.3}$$

$$= \frac{(2.9 s_{max} - 3.8 s_{min})}{3.8 \times 10^{4/3}} d^{2/3} + s_{min} \tag{4.4}$$

$$S(s_{min}, s_{max}, d) = rand[S_{min}(s_{min}, s_{max}, d), S_{max}(s_{min}, s_{max}, d)]. \tag{4.5}$$

These are shown in Figure 4.1. The algorithm used to determine the sets per workout is identical to that of determining the number of exercises per set.

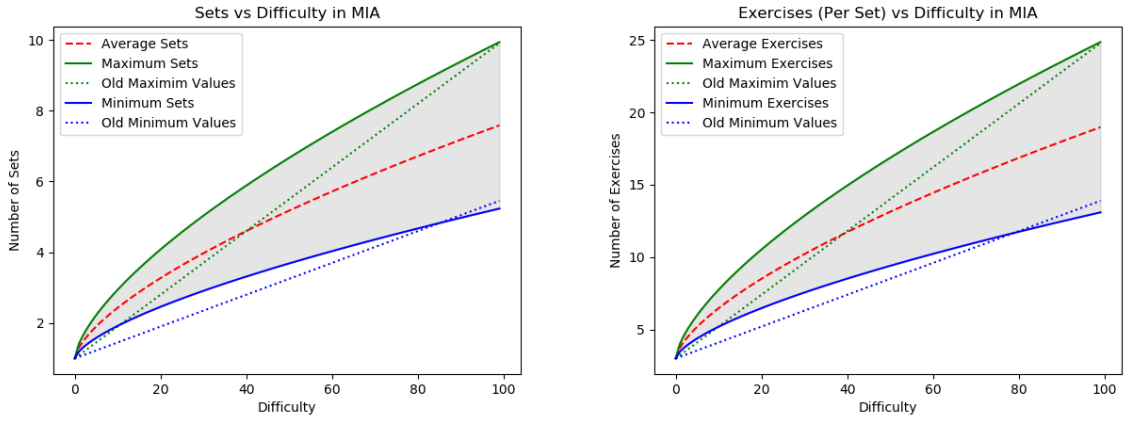

Figure 4.1: Number of sets per workout (left) and number of exercises pet set (right) based on the user input difficulty. The small dotted lines represent the original algorithm which was a simple linear increase in difficulty for each parameter. For the above two figures, values of $s_{min} = 1.0$, $s_{max} = 10.0$, $e_{min} = 3.0$ and $e_{max} = 25.0$ were used. The possible values for S and E are shown via the gray shaded areas.

4.3.2 Number of Exercises Per Set

The number of exercises, denoted $E(e_{min}, e_{max}, d) \equiv E$ is dependent on three variables. The first two are from the input file which are minNumOfExercises, denoted e_{min} and minNumOfExercises, denoted e_{max}. The last is the difficulty d which is input by the user upon generation. The maximum number of sets was originally based on a linear increase, however for better optimization of the real world workout difficulties, a custom algorithm was created. The maximum $E_{max}(e_{min}, e_{max}, d)$ and minimum $E_{min}(e_{min}, e_{max}, d)$ number of sets per workout are given by

$$E_{max}(e_{min}, e_{max}, d) \equiv E_{max} = \frac{e_{max} - e_{min}}{10^{4/3}} d^{2/3} + e_{min} \tag{4.6}$$

$$E_{min}(e_{min}, e_{max}, d) \equiv E_{min} = \frac{e_{max} - 1.9 \times e_{min}}{1.9 \times 10^{4/3}} d^{2/3} + e_{min}. \tag{4.7}$$

Thus since $E(e_{min}, e_{max}, d)$ is a random value between these curves, we have

$$E_{ave}(e_{min}, e_{max}, d) \equiv E_{ave} = \frac{E_{max} + E_{min}}{2} \tag{4.8}$$

$$= \frac{(2.9e_{max} - 3.8e_{min})}{3.8 \times 10^{4/3}} d^{2/3} + e_{min} \tag{4.9}$$

$$E(e_{min}, e_{max}, d) = rand[E_{min}(e_{min}, e_{max}, d), E_{max}(e_{min}, e_{max}, d)]. \tag{4.10}$$

These are shown in Figure 4.1. Since this value depends on each set, and there are generally numerous sets per workout, we denote the set within the workout with i such that $1 \leq i \leq S$, where S is the total number of sets within a workout. Using this index, E_i becomes

$$E_{i,max}(e_{min}, e_{max}, d) \equiv E_{max} = \frac{e_{max} - e_{min}}{10^{4/3}} d^{2/3} + e_{min} \tag{4.11}$$

$$E_{i,min}(e_{min}, e_{max}, d) \equiv E_{min} = \frac{e_{max} - 1.9 \times e_{min}}{1.9 \times 10^{4/3}} d^{2/3} + e_{min} \tag{4.12}$$

$$E_{i,ave}(e_{min}, e_{max}, d) \equiv E_{ave} = \frac{E_{max} + E_{min}}{2} \tag{4.13}$$

$$= \frac{(2.9e_{max} - 3.8e_{min})}{3.8 \times 10^{4/3}} d^{2/3} + e_{min} \tag{4.14}$$

$$E_i(e_{min}, e_{max}, d) = rand[E_{i,min}(e_{min}, e_{max}, d), E_{i,max}(e_{min}, e_{max}, d)]. \tag{4.15}$$

Following this, the average number of exercises done for a given workout would be

$$E_W = \frac{1}{S} \sum_{i=1}^{S} E_i. \tag{4.16}$$

4.3.3 Number of Reps Per Exercise

The number of reps, denoted $R(t, d, w) \equiv R$ is dependent on three variables. The first is toughness t which is gathered from the input file or defaults to 0.1. The second is difficulty d which is input by the user upon generation. Lastly, the weight w of a given exercise is needed to provide the total reps that are output. The reps are determined by a linear trend given by

$$R_{max}(t, d, w) = tdw + w \tag{4.17}$$

$$R_{min}(t, d, w) = \frac{tdw}{2} + w \tag{4.18}$$

$$R_{ave}(t, d, w) = \frac{R_{max} + R_{min}}{2} = \frac{3tdw}{4} + w \tag{4.19}$$

$$R(t, d, w) = rand[Rmin(t, d, w), R_{max}(t, d, w)]. \tag{4.20}$$

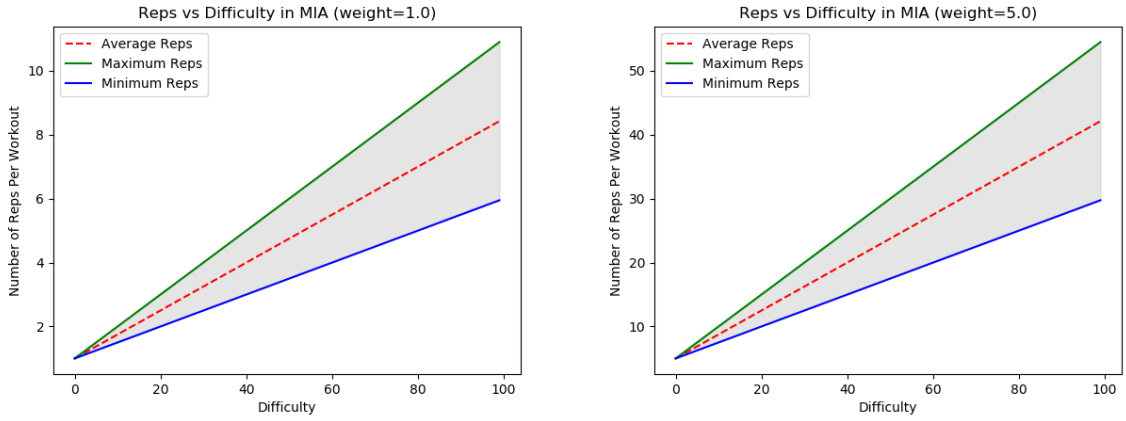

Figure 4.2: Number of reps per exercise. On the right, we have $R(0.1, d, 1)$ and on the left, $R(0.1, d, 5)$. The possible values for R are shown via the gray shaded area.

These are shown in Figure 4.2. Since this value depends on each exercise, and there are generally numerous exercises per set, we denote the exercises within the set by an index j such that $1 \leq j \leq E_i$, where E_j is the total number of exercises within the given set i. Using this index, R becomes

$$R_{j,max}(t, d, w_j) = tdw_j + w_j \tag{4.21}$$

$$R_{j,min}(t, d, w_j) = \frac{tdw_j}{2} + w_j \tag{4.22}$$

$$R_{j,ave}(t, d, w_j) = \frac{R_{max} + R_{min}}{2} = \frac{3tdw_j}{4} + w_j \tag{4.23}$$

$$R_j(t, d, w_j) = rand[R_{j,min}(t, d, w_j), R_{j,max}(t, d, w_j)]. \tag{4.24}$$

Following this, the average number of normalized reps done per set would be

$$R_{i,ave} = \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{R_j}{w_j}. \tag{4.25}$$

Then, the average number of reps (normalized by the weights) done per workout would be given by

$$R_W = \frac{1}{S} \sum_{i=1}^{S} \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{R_j}{w_j} \tag{4.26}$$

4.4 Real World Difficulties

Due to the way that we set up the weighted system for each exercise, we can easily determine how difficult, denoted D a workout is in reality based upon the generation. First, a workout is directly proportional to the number of sets it contains and thus $D \propto S$. Similarly, the difficulty of each set is proportional to the number of exercises are contained within each set and thus we use $D \propto E_W$. Lastly, the difficulty of each exercise is proportional to the number of normalized reps, and thus $D \propto E_W$. By combining all of these components we get

$$D \equiv SE_W R_W = S \left(\frac{1}{S} \sum_{i=1}^{S} E_i \right) \frac{1}{S} \sum_{i=1}^{S} \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{R_j}{w_j}. \tag{4.27}$$

References

[1]