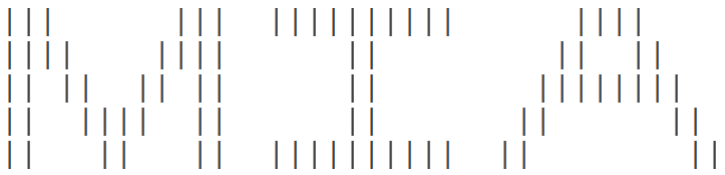

Multiple Integrated Applications (MIA)
Manual & Programming Documentation

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

Version – 0.041
Latest update: July 17, 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: July 17, 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.

1.1 Setting up the MIA Encironment for Developers

To set up MIA for development, simply git pull the project and begin working.

1.2 Setting up the MIA Environment for End Users

1. First, head to the below link. You may have to type it by hand if copy-paste does not work properly[1].

   ```
   https://github.com/torodean/Antonius-MIA
   ```

2. Select the "Clone or Download" button then "Download ZIP" as shown in Fig. 1.1.

3. Select "Save File" and then press "OK" in the window that appears. All necessary MIA files will be downloaded here. Depending on your browser (i.e Firefox, Google Chrome, Internet Explorer, etc) the downloaded files will be placed somewhere (generally the downloads folder). See Fig. 1.2 for this step.

[1] Sometimes the "-" is compiled through LaTeXas different ascii characters that different pdf readers or browsers do not understand as a generic dash.

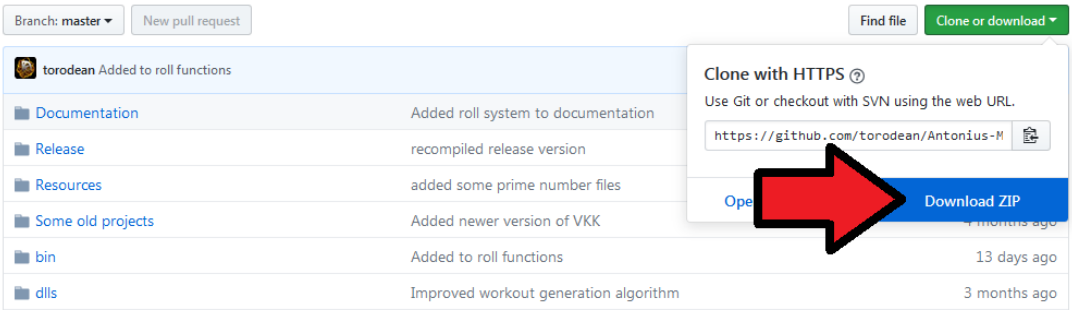

Figure 1.1: The release folder.

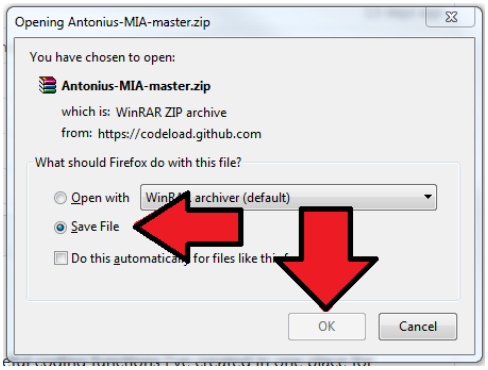

Figure 1.2: The release folder.

4. Navigate to your downloads folder. Right click on the "Antonius-MIA-master.zip" that should have been downloaded. If you have WinRAR installed, select "Extract here." If you have 7-ZIP installed, select "7-ZIP", then "Extract here." If you have another compression software installed, figure out how to extract it. If you do not have compression software installed you can do so easily by yourself or using Ninite (https://ninite.com/). See Fig. 1.3 for this step.

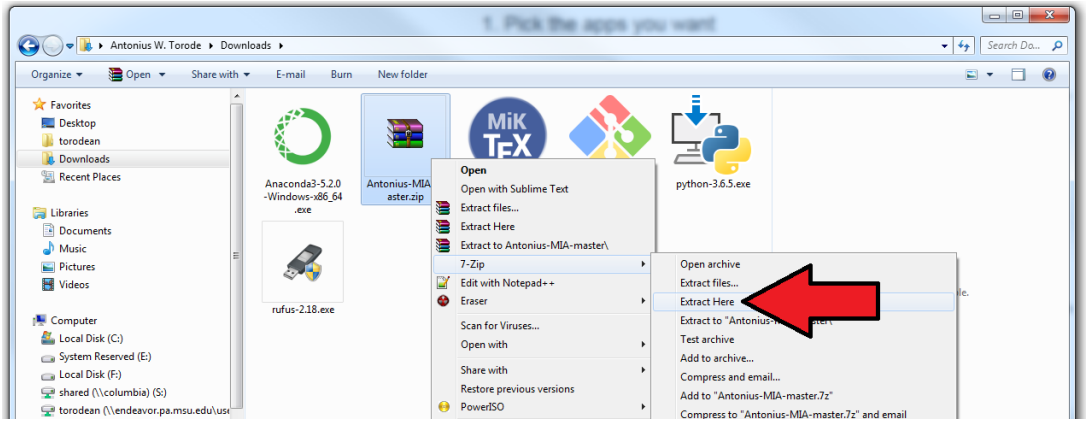

Figure 1.3: The release folder.

5. If done properly, a folder will have been created in your downloads folder called "Antonius-MIA-master." You can move this folder to wherever yo uwant the MIA program to be stored on your computer. Then, you can delete all items in the "Antonius-MIA-master" folder EXCEPT the "Release" folder, the "README.md" file, and the "Antonius' General Purpose License (AGPL)" file. After doing so, you should be able to run and use MIA

simple by opening the Release folder and clicking the “MIA.exe” file as shown in figure 1.4.

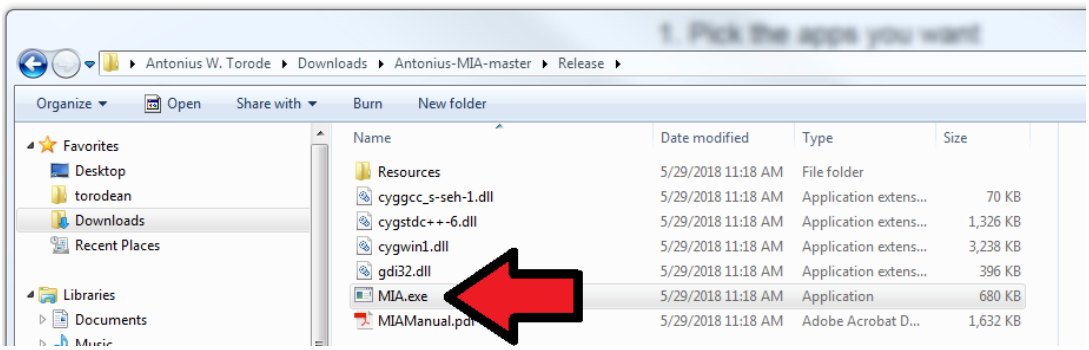

Figure 1.4: The release folder.

Chapter 2

MIAConfig file (MIAC)

2.1 MIAConfig Purpose Introduction

The MIAConfig.txt file (MIAC) is a file designed to hold program variables. The intent and purpose os this file is such that program variables can be changed after compilation and affect the program itself. This is useful for variables that are user dependent and in cases where the user is unable to re-compile the program themselves.

2.2 MIAConfig Contents and Usage

The MIAC will appear similar to the following.

```
#=============================================================================
# Name        : MIAConfig.txt
# Author      : Antonius Torode
# Date        : 1/10/18
# Copyright   : This file can be used under the conditions of Antonius'
#             General Purpose License (AGPL).
# Description : MIA settings for program initialization.
#=============================================================================

# Create a commented line using the '#' character. Comments must be on their own
    line.
# This file must be of the proper format to work with MIA.
# Create a setting parameter using 'settingVariable=value'.
# Do not include spaces unless within a string variable.
# This means, do not do 'variable= text' unless you intend to have " text" be the
     variable value.

# MIA file location variables.
defaultInputFilePath=../bin/Resources/InputFiles/
defaultCryptFilePath=../bin/Resources/EncryptedFiles/
workoutsFilePath=../bin/Resources/InputFiles/exercises.txt
workoutOutputFilePath=../bin/Resources/OutputFiles/workout.txt
excuseFilePath=../bin/Resources/Excuses.txt

# MIA program variables.
verboseMode=false

#... Other variables below
```

The contents of this file are simple. First, comments are made by adding the '#' character at the beginning of any line. These lines are ignored by MIA upon compilation and reading in the MIAC for use. Second, empty lines are also ignored by MIA. The MIAC file is pretty self explanatory, but it does require the correct format for proper use with MIA. Space characters

are important within the MIAC file and will be taken as intentional. This is because there may be MIA variables that are string inputs such that space characters are needed.

A MIA variable will only be effective in the MIAC if it is intended to work with the MIA program itself. MIA has a set amount of internal config variables which can be seen in section 2.3. The variables that are valid must be declared by their variable name (case sensitive) followed by an equal sign then followed by the value. After changing variables in the MIAC, you can save the MIAC and either reload the MIA program or type config in the MIA terminal to re-load the changed variables. In some cases MIA will reload the variables itself depending on if the commands ran are dependent on the MIAC.

Note. If a variable used by MIA is not found in the MIAC, a default value will be used that is determined upon compile time.

2.3 Valid MIAConfig (MIAC) variables

2.3.1 File and Folder Paths

The MIA program depends on multiple folder and file paths. These are all able to be adjusted within the MIAC. The file paths are by default defined relative to the MIA program and the available paths that can be defined are as follows.

```
# MIA file location variables.
defaultInputFilePath=../bin/Resources/InputFiles/
defaultCryptFilePath=../bin/Resources/EncryptedFiles/
workoutsFilePath=../bin/Resources/InputFiles/exercises.txt
workoutOutputFilePath=../bin/Resources/OutputFiles/workout.txt
excuseFilePath=../bin/Resources/Excuses.txt
```

2.3.2 MIA Program Related Variables

The MIA program related variables are variables that alter the way the MIA program will run. The available progrma variables are listed below.

```
# MIA program variables.
verboseMode=false
```

The program variable verboseMode determines whether MIA will print all possible text during runtime relating to the processes being ran. At the time of adding this feature, many functions were already developed to not include verbose text output and thus will still remain silent when this is enabled.

2.3.3 World of Warcraft Related Variables

MIA contains many World of Warcraft (WoW) related functions. These all are user dependent and thus contain MIAC available variables to editing. The available MIAC variables relating to WoW follow as

```
# WoW related variables.
WoWMailboxSendLetterLocationX=279
WoWMailboxSendLetterLocationY=647
WoWMailboxFirstLetterLocationX=55
WoWMailboxFirstLetterLocationY=265
WoWMailboxLootLetterLocationX=675
WoWMailboxLootLetterLocationY=600
WoWMailboxDeleteLetterLocationX=700
WoWMailboxDeleteLetterLocationY=650
```

```
# Variables relating to the fishbot implementation inside MIA.
WoWFishBotStartX=725
WoWFishBotStartY=360
WoWFishBotEndX=1230
WoWFishBotEndY=495
WoWFishBotIncrement=40
WoWFishBotNumOfCasts=10000
WoWFishBotDelay=10000
```

All of the `WoWMailbox` variables are related to coordinates within WoW. The required coordinates for the user can be determined using the 'find mouse' command in the correct locations. See section 6.1 for more details on these variables.

All of the `WoWFishBot` variables are related to coordinates and fishbot settings within WoW. The required coordinates for the user can be determined using the 'find mouse' command in the correct locations. See section **??** for more details on these variables.

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)

3.2.1 Static Commands

```
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 4.

```
crypt -d0s2
```

Encrypts a string using the d0s2 algorithm. This is explained more in chapter 4.

```
decrypt -d0s1
```

De-crypts a string using the d0s1 algorithm. This is explained more in chapter 4.

```
decrypt -d0s2
```

De-crypts a string using the d0s2 algorithm. This is explained more in chapter 4.

```
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.

```
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 5 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 5 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. See section 6.1 for more details.

```
wow unload
```

Unloads a number of letters from the WoW inbox. This is useful in conjunction with the 'wow dup letter' feature. This will simulate opening a letter, obtaining the contents of that letter (assuming it contains 1 item), then deleting said letter 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. See section 6.1 for more details.

```
exit
```

Quits MIA.

3.2.2 Fluid (Volatile) Commands

The fluid commands are commands that do not have a fixed input. They are generally formatted commands that can be entered with user specific input.

```
XXdYY    //Where XX and YY are integers.
1d20     //Example of rolling a 20 sided dice.
3d6      //Example of rolling three 6 sided dice.
```

Rolls a dice. The format of this command is XXdYY, where XX and YY are both integers. The value of XX determines the number of dice to roll and the value of YY determines the value of each dice.

Chapter 4

D0sag3 Command (D3C) Integration

4.1 D3C Introduction and Overview

The D3C encryption was an incorporation of an old encryption program I created many years ago as part of the D3C (d0sag3 command) program. The original code was made when I was first learning C++ and this was used as a project for educational purposes. The encryption algorithm utilizes random numbers, bit analysis, variable type conversions, and more.

4.2 d0s1 Encryption

The d0s1 encryption algorithm was the first implementation of encryption within the D3C program. The d0s1 algorithm is programmed solely to encrypt an input string value. To outline the algorithm that d0s1 uses, we will start with an example string "hello." The algorithm follows.

```
# Start with an input string
Hello

# Each character is examined individually.
H e l l o

# The string get's converted to integers based on the ascii value of each
    character.
72 101 108 108 111

# The integers are converted to a binary representation.
1001000 1100101 1101100 1101100 1101111

# A random number is generated for each character that existed.
103 70 105 115 81

# The random numbers are converted to binary representations.
1100111 1000110 1101001 1110011 1010001

# The string and random binary numbers are added to a trinary number.
   1001000 1100101 1101100 1101100 1101111
+  1100111 1000110 1101001 1110011 1010001
-----------------------------------------
=  2101111 2100211 2202101 2211111 2111112

# The random numbers selected before are converted to base 12 numbers.
103 70 105 115 81 = 87 5A 89 97 69

# The base 12 random numbers are placed at the end of the trinary string.
210111187 21002115A 220210189 221111197 211111269
```

```
# The ouput of the encryption is then these values.
2101111871002115A22021018922111197211111269
```

The encryption was meant to have a final stage to decrease the length of the output by assigning different characters to the number sequences output; however, this was never finished.

4.3 d0s2 Encryption

d0s2 encryption is a very similar algorithm to d0s1 with one major difference. The encryption of d0s2 requires a user input password that is added into the encryption process. The password and string are both encrypted and then added together in a way that the password is needed for quick decryption.

4.4 d0s3 Encryption

The d0s3 encryption algorithm was (as of the time writing this) never finished. The d0s3 was the actual D3C encryption that was originally desired with d0s1 and d0s2 being practice runs for the creator to experiment with C++ first before employing an actual complicated algorithm. MIA currently has parts of the d0s3 encryption programmed in but they are still in development and not yet deployed. The d0s3 encryption algorithm is not related to d0s1 and d0s3 but will instead have a unique and complicated algorithm that can encrypt entire files instead of just string values.

Chapter 5

Workout Generation

5.1 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 5.3 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.

5.2 Input File and Defining Workouts

5.2.1 Input File

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 **??** 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.

5.2.2 Workout Generation Parameters

```
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 5.3 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.

5.2.3 Defining Exercises

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
```

5.3 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.

5.3.1 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

$$S_{max}(s_{min}, s_{max}, d) \equiv S_{max} = \frac{s_{max} - s_{min}}{10^{4/3}} d^{2/3} + s_{min} \tag{5.1}$$

$$S_{min}(s_{min}, s_{max}, d) \equiv S_{min} = \frac{s_{max} - 1.9 \times s_{min}}{1.9 \times 10^{4/3}} d^{2/3} + s_{min}. \tag{5.2}$$

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

$$S_{ave}(s_{min}, s_{max}, d) \equiv S_{ave} = \frac{S_{max} + S_{min}}{2} \tag{5.3}$$

$$= \frac{(2.9 s_{max} - 3.8 s_{min})}{3.8 \times 10^{4/3}} d^{2/3} + s_{min} \tag{5.4}$$

$$S(s_{min}, s_{max}, d) = rand[S_{min}(s_{min}, s_{max}, d), S_{max}(s_{min}, s_{max}, d)]. \tag{5.5}$$

These are shown in Figure 5.1. The algorithm used to determine the sets per workout is identical to that of determining the number of exercises per set.

5.3.2 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 maxNumOfExercises, 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

$$E_{max}(e_{min}, e_{max}, d) \equiv E_{max} = \frac{e_{max} - e_{min}}{10^{4/3}} d^{2/3} + e_{min} \tag{5.6}$$

$$E_{min}(e_{min}, e_{max}, d) \equiv E_{min} = \frac{e_{max} - 1.9 \times e_{min}}{1.9 \times 10^{4/3}} d^{2/3} + e_{min}. \tag{5.7}$$

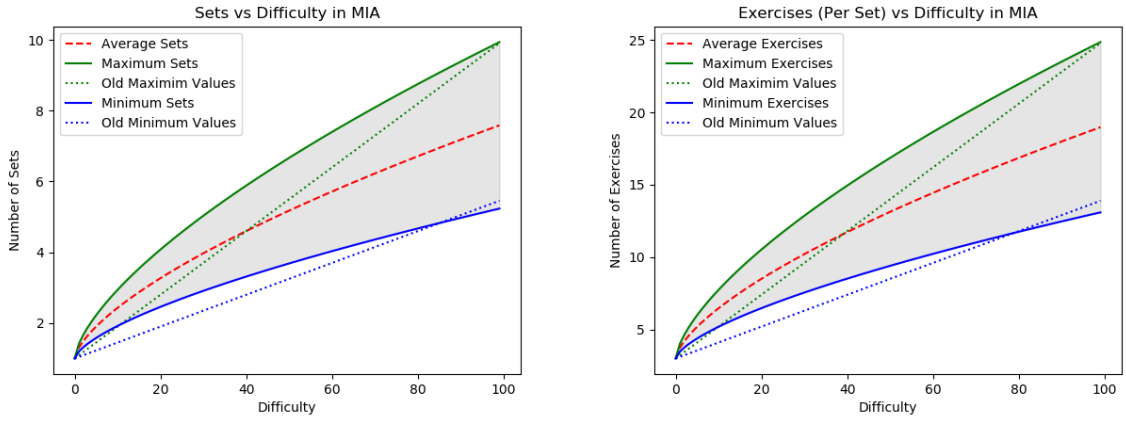

Figure 5.1: Number of sets per workout (left) and number of exercises pet set (right) based on the user input difficulty. The small dotted lines represent the original algorithm which was a simple linear increase in difficulty for each parameter. For the above two figures, values of $s_{min} = 1.0$, $s_{max} = 10.0$, $e_{min} = 3.0$ and $e_{max} = 25.0$ were used. The possible values for S and E are shown via the gray shaded areas.

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

$$E_{ave}(e_{min}, e_{max}, d) \equiv E_{ave} = \frac{E_{max} + E_{min}}{2} \tag{5.8}$$

$$= \frac{(2.9e_{max} - 3.8e_{min})}{3.8 \times 10^{4/3}} d^{2/3} + e_{min} \tag{5.9}$$

$$E(e_{min}, e_{max}, d) = rand[E_{min}(e_{min}, e_{max}, d), E_{max}(e_{min}, e_{max}, d)]. \tag{5.10}$$

These are shown in Figure 5.1. 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

$$E_{i,max}(e_{min}, e_{max}, d) \equiv E_{max} = \frac{e_{max} - e_{min}}{10^{4/3}} d^{2/3} + e_{min} \tag{5.11}$$

$$E_{i,min}(e_{min}, e_{max}, d) \equiv E_{min} = \frac{e_{max} - 1.9 \times e_{min}}{1.9 \times 10^{4/3}} d^{2/3} + e_{min} \tag{5.12}$$

$$E_{i,ave}(e_{min}, e_{max}, d) \equiv E_{ave} = \frac{E_{max} + E_{min}}{2} \tag{5.13}$$

$$= \frac{(2.9e_{max} - 3.8e_{min})}{3.8 \times 10^{4/3}} d^{2/3} + e_{min} \tag{5.14}$$

$$E_i(e_{min}, e_{max}, d) = rand[E_{i,min}(e_{min}, e_{max}, d), E_{i,max}(e_{min}, e_{max}, d)]. \tag{5.15}$$

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

$$E_W = \frac{1}{S} \sum_{i=1}^{S} E_i. \tag{5.16}$$

5.3.3 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

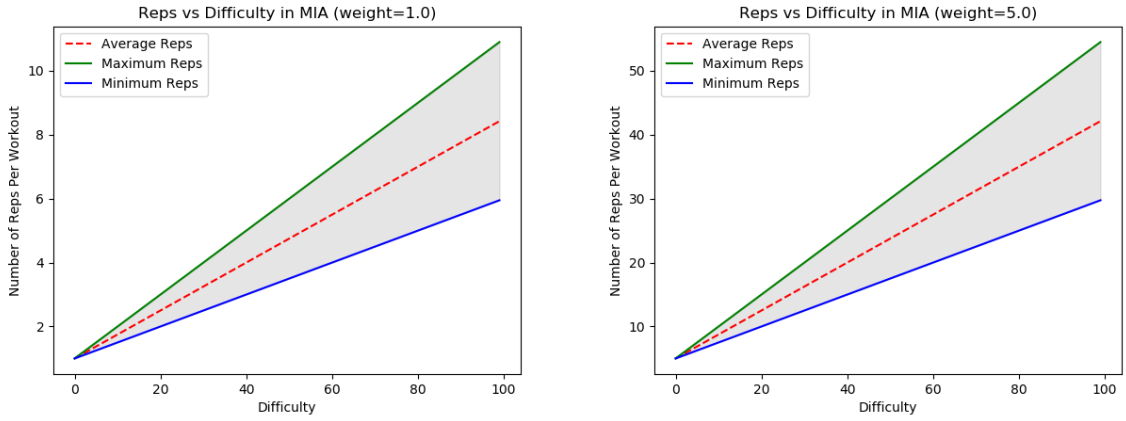

Figure 5.2: Number of reps per exercise. On the right, we have $R(0.1, d, 1)$ and on the left, $R(0.1, d, 5)$. The possible values for R are shown via the gray shaded area.

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

$$R_{max}(t, d, w) = tdw + w \tag{5.17}$$

$$R_{min}(t, d, w) = \frac{tdw}{2} + w \tag{5.18}$$

$$R_{ave}(t, d, w) = \frac{R_{max} + R_{min}}{2} = \frac{3tdw}{4} + w \tag{5.19}$$

$$R(t, d, w) = rand[Rmin(t, d, w), R_{max}(t, d, w)]. \tag{5.20}$$

These are shown in Figure 5.2. 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

$$R_{j,max}(t, d, w_j) = tdw_j + w_j \tag{5.21}$$

$$R_{j,min}(t, d, w_j) = \frac{tdw_j}{2} + w_j \tag{5.22}$$

$$R_{j,ave}(t, d, w_j) = \frac{R_{max} + R_{min}}{2} = \frac{3tdw_j}{4} + w_j \tag{5.23}$$

$$R_j(t, d, w_j) = rand[R_{j,min}(t, d, w_j), R_{j,max}(t, d, w_j)]. \tag{5.24}$$

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

$$R_{i,ave} = \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{R_j}{w_j}. \tag{5.25}$$

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

$$R_W = \frac{1}{S} \sum_{i=1}^{S} \frac{1}{E_i} \sum_{j=1}^{E_i} \frac{R_j}{w_j} \tag{5.26}$$

5.4 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

$$D \equiv SE_W R_W = S\left(\frac{1}{S}\sum_{i=1}^{S} E_i\right)\frac{1}{S}\sum_{i=1}^{S}\frac{1}{E_i}\sum_{j=1}^{E_i}\frac{R_j}{w_j}. \tag{5.27}$$

To demonstrate the possible values that can be output by D we can examine the maximum and minimum values. The maximum value would be when S, E, and R are at their maximums. Thus, we have

$$D_{max} = S_{max}\left(\frac{1}{S_{max}}\sum_{i=1}^{S_{max}} E_{i,max}\right)\frac{1}{S_{max}}\sum_{i=1}^{S_{max}}\frac{1}{E_{i,max}}\sum_{j=1}^{E_{i,max}}\frac{R_{j,max}}{w_j}. \tag{5.28}$$

Since each $E_{i,max}$ and $R_{j,max}/w_j$ are the same for all i, j respectively, then we can simplify this to

$$D_{max} = S_{max}\left(\frac{1}{S_{max}}S_{max}E_{max}\right)\frac{1}{S_{max}}S_{max}\frac{1}{E_{max}}E_{max}\frac{R_{max}}{w} \tag{5.29}$$

$$= S_{max}E_{max}\frac{R_{j,max}}{w_j}. \tag{5.30}$$

Similarly, the minimum value is

$$D_{min} = S_{min}E_{min}\frac{R_{j,min}}{w_j}, \tag{5.31}$$

where $R_{j,min}/w_j$ represents the normalized reps for each exercise in both of the above two cases. The possible values of D then lie between these two curves and can be seen in figure 5.3.

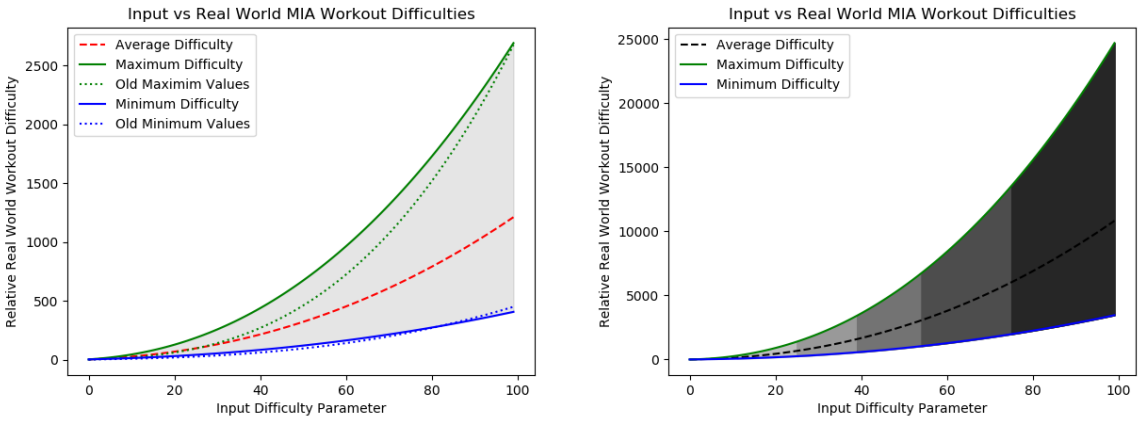

Figure 5.3: The real world difficulties D output by the MIA workout generation program versus the input difficulty parameter (LEFT). The possible values for D are shown via the gray shaded area. The old min/max values represent D based on the old S and E values (see figure 5.1). On the right is the same plot only depicting the difficulty ranges. The ranges run from very easy (light gray), to very hard (dark gray).

The real world difficulty D has an exponential increase. This is desired for a specific reason. As improvements are made, meaning as ones physique and ability to perform improves, a greater

challenge is needed. Similarly, there is a much larger variance in the real world difficulty D increases with respect to the input difficulty d. This is because as workout intensities increase, there is a desire to keep the body both guessing and not straining too often. Therefore, by increasing the variability of a workout, there is a greater effectiveness and an improved 'rest' or 'slow' period between intensive workouts.

5.5 Notes on Appropriate MIA Parameters For Usage

Based on the way MIA generates workouts (as described in the above sections), there are a few things to keep in mind when determining the proper settings. First, if a maxNumberOfExercises value of 'inf' is used, then the real world difficulty output will be proportional to the number of exercises defined in the input file. Therefore, if one places a thousand different workouts in this file, each difficulty will be drastically more intense than if there were only 25. Thus, it is important to experiment with this value and adjust accordingly based on the number of defined exercises you have in the input file.

Chapter 6

World of Warcraft (WoW) Features in MIA

6.1 Mailbox Management

Note. This program technically violates the blizzard license agreement and should be used with caution. However, it is no more complicated than an addon. I just happen to know C++ and not lua.

MIA contains some in game mail management for World of Warcraft (WoW) in order to assist in the process of creating in game letters in bulk. In game letters can't be sent between characters and then looted and stored in the character inventories. In some cases, such as role playing (RP) or guild recruitment, these letters may be desired by bulk. The process of creating these letters consists of entering in a letter recipient, subject, the contents of the letter, and then hitting a send button (see figure 6.1). After sending these letters, however many times it is done, they then need to be looted on the character that they were sent to. This consists of opening up the mailbox, clicking on the letter received, acquiring hte physical copy by looting it from the opened letter, then deleting the letter to clean up the mailbox (see figure 6.2). MIA is designed to automate this entire process.

6.1.1 Sending Duplicates of a Letter

As described above briefly, MIA has the capability of automating the sending of duplicate letters to a recipient in WoW. To do this, there are a few settings that need to be configured on the user end to ensure proper functionality. There are two variables contained in the MIAC (see section 2 for more information) that are used in conjunction with the `wow dup letter` function in MIA. These are listed below.

```
# WoW related variables.
WoWMailboxSendLetterLocationX=279
WoWMailboxSendLetterLocationY=647
```

By default, these values are set for the environment that was used when originally programmed into MIA. Both of these variables are used to represent the location of the send button shown in figure 6.1. The first, `WoWMailboxSendLetterLocationX` represents the x coordinate and the latter `WoWMailboxSendLetterLocationY` represents the y coordinate. Both of these values can be found by using the `find mouse` function in MIA.

Once the proper coordinates are set, one can automate this process of sending a duplicate letter by using the `wow dup letter` command in the MIA terminal. The `wow dup letter` has a few limitations when used. First, the recipient of the letter can only contain normal characters such as a, e, o, etc. The recipient cannot contain special characters such as ä, é, õ, etc. However, the

Figure 6.1: A screen shot of the WoW in game outgoing mailbox menu. The mail management within MIA utilizes the fields indicated by red squares in this menu. The red box around the send button represents the `WoWMailboxSendLetterLocation` location.

contents of the message that will be sent can contain special characters. The desired message contents should be copied to the clipboard before running the `wow dup letter` command. Upon running the command, the terminal will prompt the user to do so as well. The subject field will automatically be filled in with "subject."

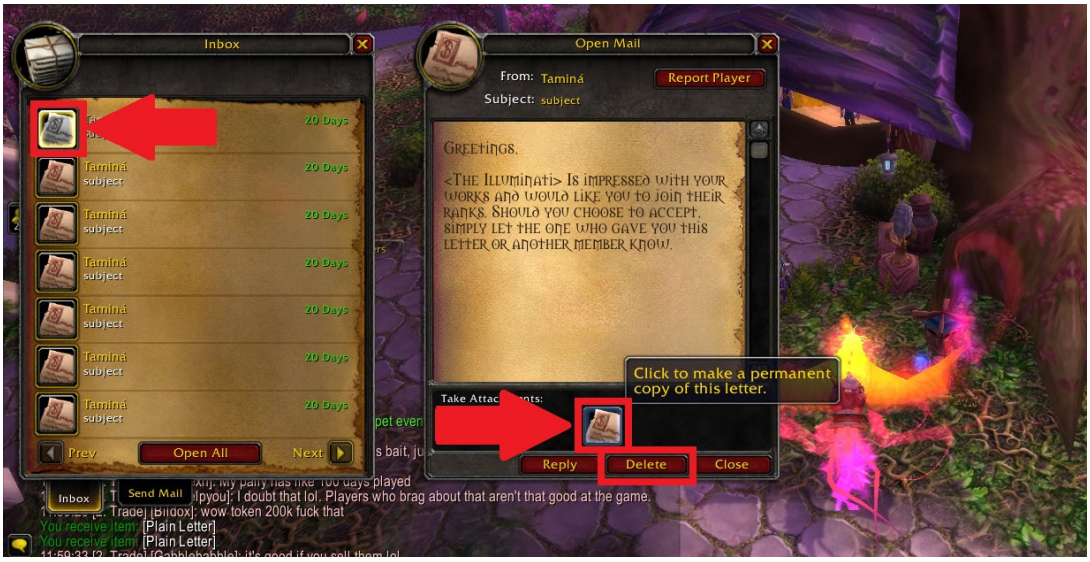

Figure 6.2: A screen shot of the WoW in game incoming mailbox menu. The mail management within MIA utilizes the fields indicated by red squares in this menu. The left arrow indicates the `WoWMailboxFirstLetterLocation` position. The right arrow indicates the `WoWMailboxLootLetterLocation` position. The square around the delete button represents the `WoWMailboxDeleteLetterLocation` location.

6.1.2 Unloading Duplicated letters

MIA contains a command `wow unload` which is designed to be used in conjunction with the `wow dup letter` command. This command will loot the letters from the incoming mailbox that are sent using the `wow dup letter` command. In order to use this command properly, there are

six different variables within the MIAC that need to be determined for the users environment. These variables are below.

```
# WoW related variables.
WoWMailboxFirstLetterLocationX=55
WoWMailboxFirstLetterLocationY=265
WoWMailboxLootLetterLocationX=675
WoWMailboxLootLetterLocationY=600
WoWMailboxDeleteLetterLocationX=700
WoWMailboxDeleteLetterLocationY=650
```

By default, these variables are set to values that were specific to the programmers environment. The variables need to be set based off of three different coorindates. The first, `WoWMailboxFirstLetterLocation` corresponds to the location of the first letter in the inbox of the user (see figure 6.2). The second, `WoWMailboxLootLetterLocation` corresponds to the location of the letter to loot from the user inbox (see figure 6.2). The last, `WoWMailboxDeleteLetterLocation` represents the locations of the delete button on a letter in the WoW inbox (see figure 6.2). For all three coordinates, there is an x and y value (represented by the variables in the MIAC) which can be determined through MIA by using the `find mouse` command.

Index