DITAA Specification

Version 1.0

October 12, 2020

Diagrams Through ASCII Art Specifications

David Holdren
Kevin Porter
Adam Shiveley

Table of Contents

1.0. Introduction

1.1 Purpose

The purpose of this document is to develop the specification for the Designs Through ASCII Art (DITAA) program. The DITAA program has been explained in full in a previous document named "Software Requirements Specifications" found at the GitHub address: https://github.com/dimeandpenny/CS_7140_Group_Project.

1.2 Scope of Document

This document will describe the general outline the DITAA program will take. A pseudo code written in a Very High Level Language (VHLL) will explain the general flow of controlling functions of the program. There are other functions that will be used within the program, but those functions are considered helper functions. Helper functions are not controlling functions within a program and will not be discussed.

2.0. Specifications of Design

2.1 Very High Level Language Functions

This section describes at a very high level the implementations of the CLI and GUI as described in this team's SRS.

2.1.1 CLI Input

```
while user_input_not_enter:
    collect_user_input()
args = string_of_user_args
```

filename = user_filename

parsed_args = parse_args(args)

textfile_contents = user_input(filename)

if is_valid(textfile_contents):

generate_graphics(textfile_contents)

save_diagram(output_file)

2.1.2 Parse Arguments

parse_args(arg_string):

 # pre: list of arguments for program are entered

 # post: list of arguments are parsed

parsed_args = list()

for arg in arg_string:

parsed_args.add(collect_and_validate_arg(arg))

return parsed_args

2.1.3 User Input

user_input(textfile):

 # pre: user launches CLI with name of textfile

 # post: return contents of textfile

return contents(textfile)

2.1.4 Graphical User Interface (GUI)

main_menu_dialog():

 # pre: user has executed the .jar file

 # post: user has created zero or more output ASCII files

while running:

wait_for_user_input()

```
    if select_file:
        infile = file_dialog(true)
    if configure_arguments:
        parsed_args = arguments_dialog()
    if select_outfile:
        outfile = file_dialog(false)
    if run_converter:
        run_converter(infile, outfile, parsed_args)
    if exiting:
        # close window
```

2.1.5 File Dialog

```
file_dialog(input_mode):
    # pre: user has opened the file selection dialog
    # post: user has selected a file
    wait_for_user_input()
    if file_selected and user_select("OK"):
        return file_selected.path
```

2.1.6 Arguments Dialog

```
arguments_dialog():
    # pre: user has opened the arguments dialog
    # post: user has selected arguments
    wait_for_user_input()
    if user_select("OK"):
        arguments = get_selected_arguments()
```

return arguments

2.1.7 User Input

user_input(textfile):

 # pre: user has supplied a textfile name

 # post: return contents of textfile

return contents(textfile)

2.1.8 Run Converter

run_converter(infile, outfile, parsed_args):

 # pre: user has selected an input file

 # pre: user has configured arguments

 # pre: user has designated an output file

 # post: infile contents have been converted and stored in outfile

infile_content = get_content(infile)

if is_valid(infile_content):

draw_and_save(infile_content, parsed_args)

return

2.1.9 Get Content

get_content(file):

 # read and return content from file

2.1.10 Valid

is_valid(textfile_content):

 # pre: structure with textfile content

 # post: true/false whether the textfile contains convertible contents

textfile_content = get_content(file)

if textfile_content.match(required_pattern):

return True

else:

return False

2.1.11 Draw and Save

draw_and_save(textfile_content):

 # pre: textfile_content contains valid ASCII art

 # post: return rendered image of ASCII art

list_of_objects = find_objects(textfile_content)

for object in list_of_objects:

diagram = draw_diagram(object)

save_diagram(outfile)

3.0. Non-Functional Requirements

3.1 Code Reduction

The goal is to limit the program to only the required code. Assuming such a program already exist, then the goal of this project would be to use the existing program as a basis upon which to add features and reduce the code base. The overall reduction percentage will not be based on lines of code, but rather subject matter expert opinion's when comparing both code basis. This version of DITTA will focus on condensing looping conditions and implementing an object-oriented programing approach.

3.2 Limiting Bugs

The program will strive to achieve limiting common bugs within a program. Such bugs are unbound buffers, not binding printf statements, under and over flows, etc. These bugs will be detected during unit test and eliminated in the final product.

3.3 Testing

The program will be ran through a series of tests. Each unit of the test will contain known outputs along with unknown outputs. Mixed in with the known outputs will be cases in which the program should pause and wait for user input. Such cases could contain unknown ASCII values in the file, different file encoding, or different file language to name a few. Should an error occur, the error will de displayed to the user/developer and action taken accordingly.