## 

ECE103 Engineering Programming

## Project Report 1

Kevin DeLeon ([kdeleo2@pdx.edu](mailto:kdeleo2@pdx.edu))

Tom Ottero ([otero@pdx.edu](mailto:otero@pdx.edu))

Edward Rees ([edrees@pdx.edu](mailto:edrees@pdx.edu))

|  |  |
| --- | --- |
|  | |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## Introduction

ECE103: Engineering Programming focuses on the C programming language. The language was initially designed by Dennis Ritchie in 1972; however, it was not the sole language being developed in the 70s. BASIC, a programming language that continued to evolve since the 60s, was widely taught and the first language for many neophyte programmers. While the two languages originated around the same time, their application and implementation have noteworthy differences. This project shall explore those differences by porting a classic BASIC computer game, Super Star Trek, to C. Part 1 of this project involves researching the advantages and constraints of both languages, analyzing the game to understand the rules and flow, and creating a plan for porting the game to C for Part 2.

## Research

BASIC is a high level programming language originally created at Dartmouth College in 1964 with the intended purpose of enabling students in fields other than science and mathematics to use computers. BASIC is an interpreted language. According to Wikipedia the advantages are:

* [platform independence](https://en.wikipedia.org/wiki/Platform_independent)
* [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_science)) and reflective use of the evaluator
* [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing)
* smaller executable program size
* [dynamic scoping](https://en.wikipedia.org/wiki/Scope_(programming)#Dynamic_scoping)

Disadvantages include:

* without [static type-checking](https://en.wikipedia.org/wiki/Static_type-checking), which is usually performed by a compiler, programs can be less reliable.
* Interpreters can be susceptible to [Code injection](https://en.wikipedia.org/wiki/Code_injection) attacks.
* Slower execution compared to direct native [machine code](https://en.wikipedia.org/wiki/Machine_code) execution on the host [CPU](https://en.wikipedia.org/wiki/CPU).
* Source code can be read and copied (e.g. [JavaScript](https://en.wikipedia.org/wiki/JavaScript) in web pages), or more easily reverse engineered through [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_science)) in applications where intellectual property has a commercial advantage.

In contrast, C is a compiled language designed to provide low level access to memory and language constructs efficiently mapped to machine instructions. Despites its machine level capabilities, the language encourages cross-platform programming. According to Data Flair and TekSlate, the advantages of C include:

* Powerful and efficient
* Portability
* System programming

Disadvantages include:

* Lack of OOP
* Lack of exception handling
* Run-time checking
* Once compiled, the code is platform dependent

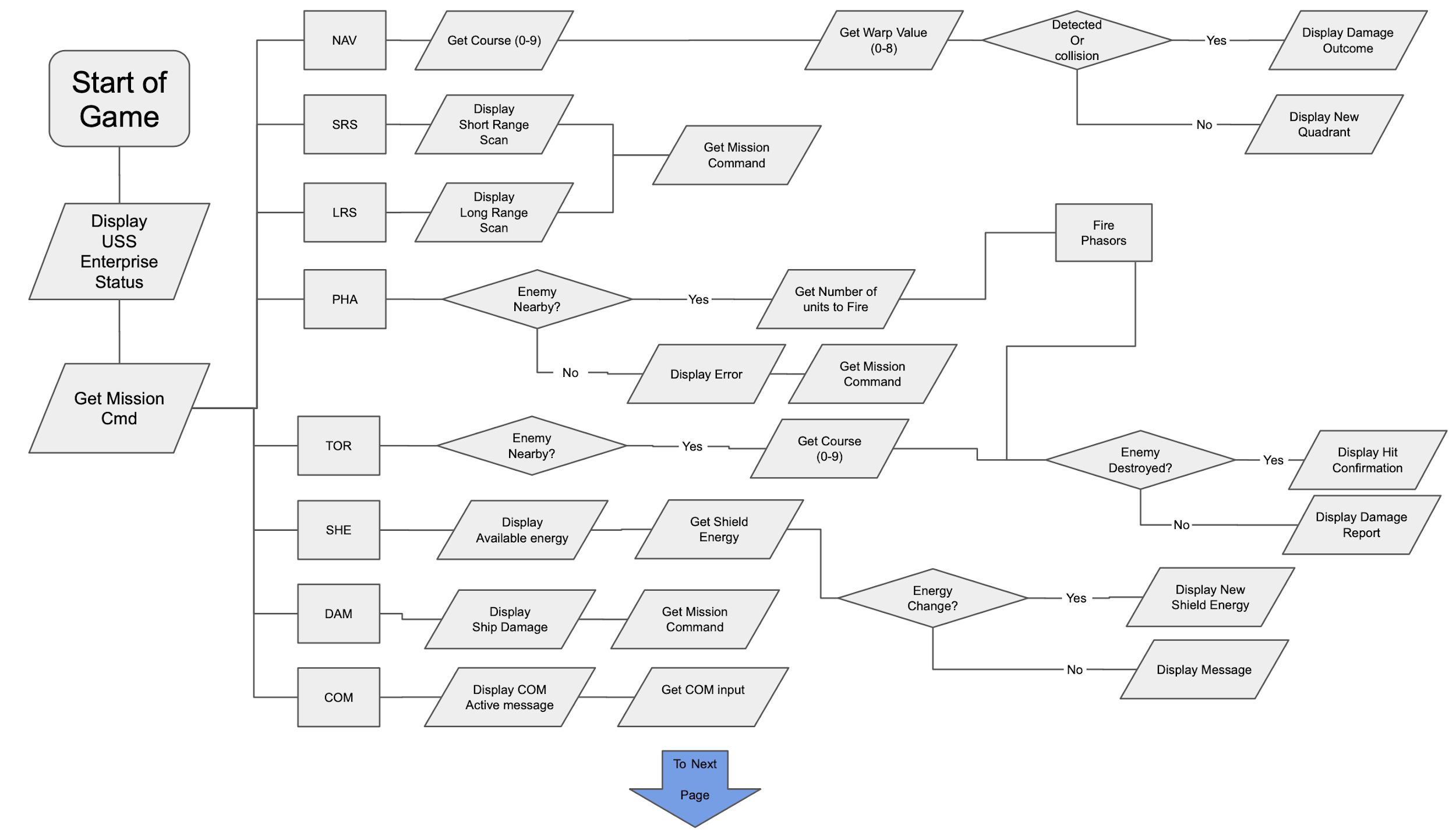
## Analysis

One of the primary challenges in reading the BASIC code is in identifying what the variables are meant to contain. The variable names are non-descriptive and the code is not commented well enough to follow without effort. In addition, the code isn’t structured in a way that makes it easy to read. For example, code is all blocked together and isn’t indented to separate loops from the rest of the code. It was also harder to follow when viewing it in a simple text editor, versus viewing it in the emulator. This was because the text editor shows all text as the same color, while the emulator differentiates between comments, print statements, commands, and outputs.

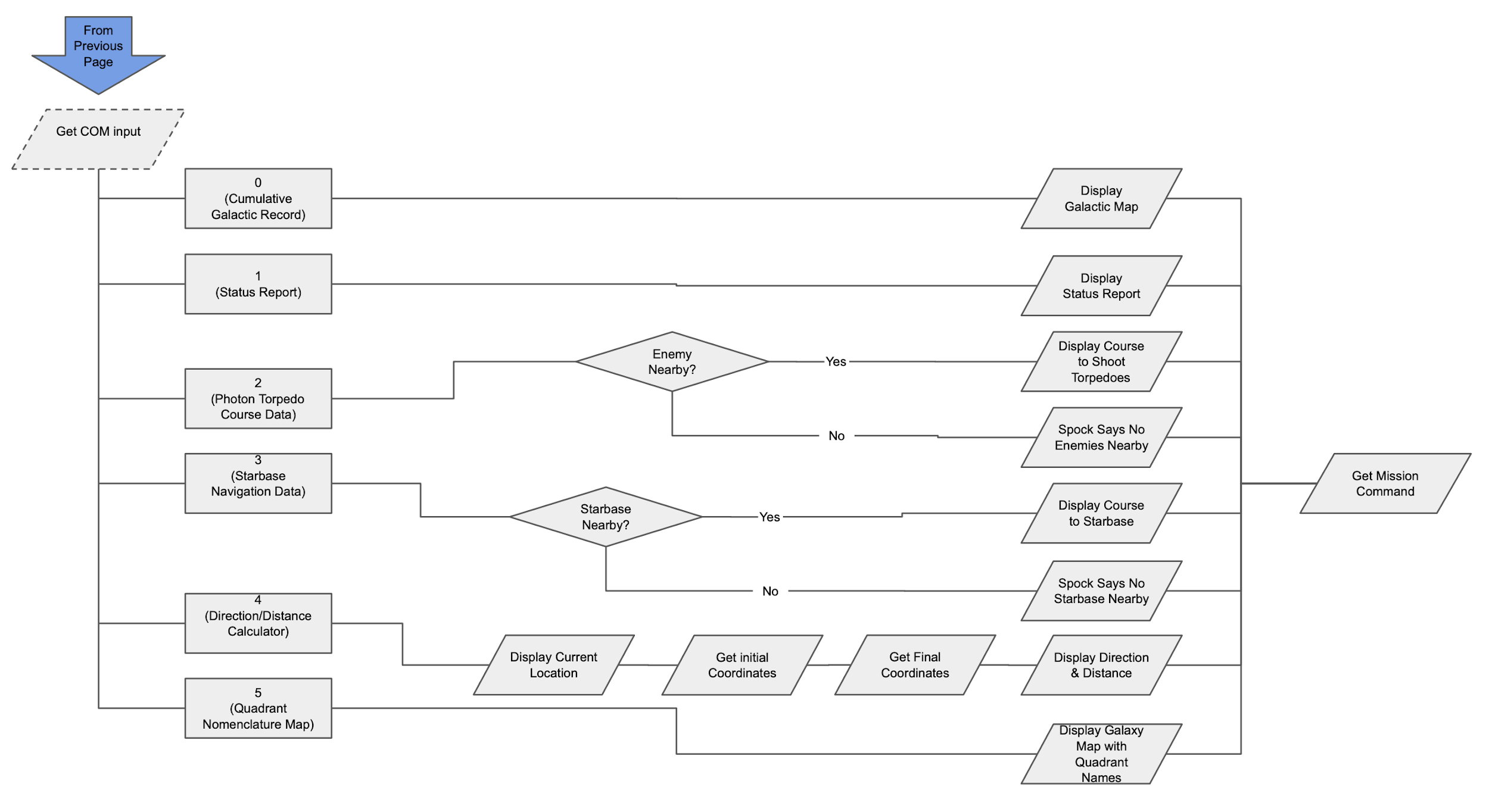
Some of the game was easier to reverse engineer through play than from reviewing the code (i.e. the game board is set as an 8x8 grid dividend into quadrants identified by name and associated roman numeral. Each of these quadrants appear to be subdivided into another 8x8 grid). What was challenging though, was figuring out how the USS Enterprise moved through these quadrants. It took a few attempts at the game to determine that the ship jumps from quadrant to quadrant rather than moving within each quadrant and even further play to find that we could use fractional numbers to maneuver within the quadrant.

Identifying the game variables proved such a challenge that we reviewed several versions of the game that had previously been ported to other languages (C#, Python, and one other). These were very useful in that they better commented the variables.

While the workload wasn't assigned in any formal fashion, we each picked an aspect of the project (or task) to evaluate. Once a specific task was complete, it was presented to the other members for review and feedback. We are currently working through the code and breaking it down into smaller sections. This will help with readability as well as translating the BASIC code into C when the time comes.



*Figure 1- Page 1 of the Game Play Flowchart*



*Figure 2- Page 2 of the Game Play Flowchart*

*Table 1- Game Play Variables*

|  |  |  |  |
| --- | --- | --- | --- |
| **BASIC Variable Name** | **Purpose** | **C Type** | **C Name** |
| d0 | Docked Flag | int | dockFlag |
| d1 | Damage Repair Flag | int | damRepFlag |
| e | Current Energy | int | currEnergy |
| e0 | Starting Energy | int | startEnergy |
| g[8,8] | Galaxy | int array | galaxy |
| g5 | Quadrant Name Flag | int | quadName |
| k[3,3] | Klingon Data | int | klingData |
| k3 | Klingon at Start | int | klingStart |
| k7 | Klingon in Quadrant | int | klingQuad |
| k9 | Klingons Remaining | int | klingLeft |
| p | Photon Torpedos Remaining | int | torpLeft |
| p0 | Photon Torpedos Capacity | int | torpCap |
| q1, q2 | Quadrant Position of Enterprise | int | entQuad1, entQuad2 |
| r1, r2 | Temp Location Coordinates | int | tempPos1, tempPos2 |
| s | Current Shields | int | shields |
| s3 | Stars in Quadrant | int | stars |
| s8 | Quadrant Index | int | quadIndex |
| s9 | Klingon Power | int | klingPow |
| t0 | Starting Stardate | int | stardateStart |
| t9 | End Time | int | stardateDone |
| z[8,8] | Galaxy Record | int array | galaxyRecord |
| z1, z2 | Temp Sector Coordinates | int | tempSectCoord1, tempSectCoord2 |
| z3 | String Compare Value | int | compare |
| z4, z5 | Temp Quadrant Coordinates | int | tempQuadCoord1, tempQuadCoord2 |
|  |  |  |  |
| d[8] | Damage | double array | damage |
| d4 | Compute repair time | double | repairTime |
| s1, s2 | Current Sector Position of Enterprise | double | entSect1, entSect2 |
| t | Current Stardate | double | stardateCurr |
| w1 | Warp Factor | double | warpFactor |
| x, y, x1, y1 | Navigation Coordinates | double | navX, navY, navX1, navX2 |

## Plan (part 2)

The major tasks to complete for Part 2 of this project are Translate, Test, Document, and Demonstrate. These tasks each contain subsections described in more detail in Figure 2 of the Appendix. The goal is to complete the Translate task by July 25th to ensure enough time for testing and bug correction. Testing for all major components of this project should be complete by August second to allow time for report writing and extra credit. Documentation is set to be concluded by the 8th allotting a week buffer before the due date for any emergencies. Finally, the demonstration is scheduled for the last week before the due date.

## Signature Paragraph

* *Ed Rees* 
  + *Began work on Game Play Flow Chart*
  + *Reviewed Game Play and BASIC functionality*
  + *Identified BASIC variables*
  + *Established C variable naming conventions*
* *Tom Otero*
  + *Researched Apple II BASIC*
  + *Evaluated logic from original game play*
  + *Contributed to Part 2 plan*
* *Kevin DeLeon*
  + *Reviewed Game Play for a basic understanding of operation*
  + *Created draft documents for team use*
  + *Reviewed BASIC code and developed a list of subroutines and code blocks.*
  + *Updated trello with Cards and Tasks*
  + *Expanded upon Ed’s top level Game Play Flow Chart*

*All of the students listed at the top of this report have read it and agree with its content.*

## References

* *List your sources of information here (books, articles, web sites, etc.)*
* [*https://en.wikipedia.org/wiki/Interpreted\_language#:~:text=Many%20languages%20have%20been%20implemented,%2Dmachine%2Dfriendly%20interpreted%20language.*](https://en.wikipedia.org/wiki/Interpreted_language#:~:text=Many%20languages%20have%20been%20implemented,%2Dmachine%2Dfriendly%20interpreted%20language.)
* [*https://en.wikipedia.org/wiki/Compiled\_language*](https://en.wikipedia.org/wiki/Compiled_language)
* [*https://data-flair.training/blogs/advantages-and-disadvantages-of-c/*](https://data-flair.training/blogs/advantages-and-disadvantages-of-c/)
* [*https://www.dartmouth.edu/basicfifty/commands.html*](https://www.dartmouth.edu/basicfifty/commands.html)
* [*https://www.complang.tuwien.ac.at/forth/programs/strek.fs*](https://www.complang.tuwien.ac.at/forth/programs/strek.fs)
* [*http://pythonfiddle.com/star-trek-game/*](http://pythonfiddle.com/star-trek-game/)
* [*https://www.codeproject.com/Articles/28228/Star-Trek-1971-Text-Game*](https://www.codeproject.com/Articles/28228/Star-Trek-1971-Text-Game)

## 

## Appendix

### *Gantt Chart*

Upon completion of the Gantt Chart it was discovered that the chart was entirely too big for a standard document format. As such, it has been divided into the following three figures. The first figure is a key describing the charts to follow. Figure 2 contains the schedule for Part 1 of the Super Star Trek project. Figure 3 describes the tasks and due dates for Part 2 of this project.

**

Figure 1. Key for Figures 2 and 3.

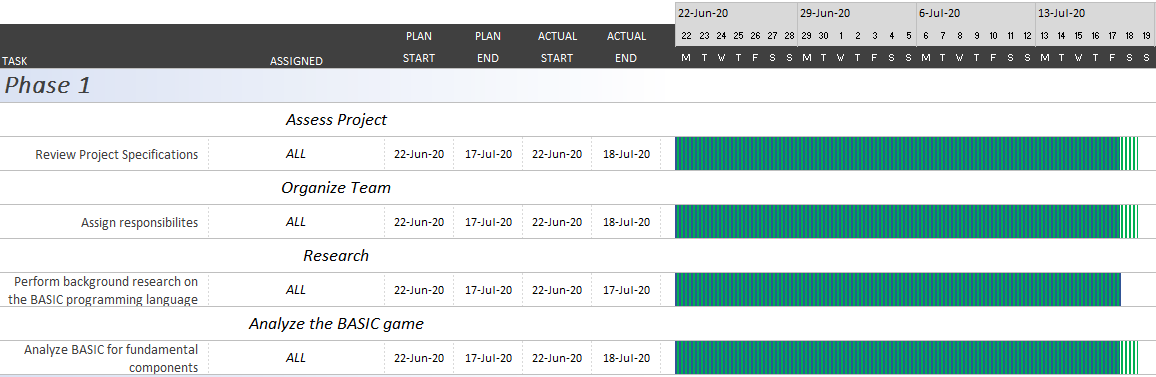
**

Figure 2. Gantt Chart for Part 1 of the Super Star Trek project.

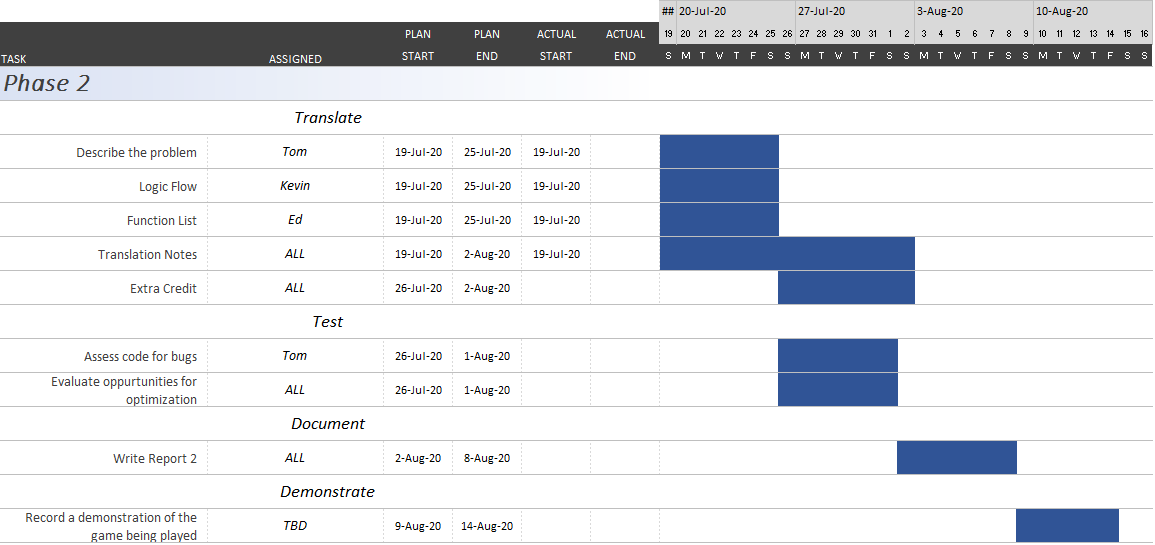
**

Figure 3. Gantt Chart for Part 2 of the Super Star Trek project.

### *Analysis: Tom*

Initially, an attempt was made to start directly writing the Apple II basic code to C. This attempt was quickly cut short when I realized the source code for Super Start Trek could not be read as a structured script. Upon this realization, I started to research Apple II BASIC, which is not necessarily something that can be shown in writing except for [this handy link to the book I’ve been reading](https://apple2online.com/web_documents/Applesoft%20II%202019.pdf). Additionally, I have been researching other people’s efforts in porting this game, so as to gain insight into what the single character variable names might represent and reading the BASIC Computer Games: Microcomputer Edition to better understand the game’s underlying logic. Some of the resources have been [saved to our teams shared Google Drive linked here.](https://drive.google.com/drive/folders/1ouA55ci2hSm4K7soSQhRfUlGOEQacWDl?usp=sharing)

Below is my original attempt to port the Apple II code to C:

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string.h>*

*#include <math.h>*

*/\**

*NEEDED TO ACTIVATE 80-COLUMN TEXT MODE ON APPLE IIE*

*SUPER STARTREK - MAY 16,1978 - REQUIRES 24K MEMORY*

*\*\*\*\* STAR TREK \*\*\*\* \*\*\*\**

*SIMULATION OF A MISSION OF THE STARSHIP ENTERPRISE,*

*AS SEEN ON THE STAR TREK TV SHOW.*

*ORIGINAL PROGRAM BY MIKE MAYFIELD, MODIFIED VERSION*

*PUBLISHED IN DEC'S "101 BASIC GAMES", BY DAVE AHL.*

*MODIFICATIONS TO THE LATTER (PLUS DEBUGGING) BY BOB*

*LEEDOM - APRIL & DECEMBER 1974,*

*WITH A LITTLE HELP FROM HIS FRIENDS . . .*

*COMMENTS, EPITHETS, AND SUGGESTIONS SOLICITED --*

*SEND TO : R. C. LEEDOM*

*WESTINGHOUSE DEFENSE & ELECTRONICS SYSTEMS CNTR.*

*BOX 746, M.S. 338*

*BALTIMORE, MD 21203*

*CONVERTED TO MICROSOFT 8 K BASIC 3/16/78 BY JOHN GORDERS*

*LINE NUMBERS FROM VERSION STREK7 OF 1/12/75 PRESERVED AS*

*MUCH AS POSSIBLE WHILE USING MULTIPLE STATEMENTS PER LINE*

*SOME LINES ARE LONGER THAN 72 CHARACTERS; THIS WAS DONE*

*BY USING "?" INSTEAD OF "PRINT " WHEN ENTERING LINES*

*\*/*

*int main()*

*{*

*// Print the Enterprise*

*printf(" ,------\*------,\n");*

*printf(" ,------------- '--- ------'\n");*

*printf(" '-------- --' / /\n");*

*printf(" ,---' '-------/ /--,\n");*

*printf(" '----------------'\n");*

*printf(" THE USS ENTERPRISE --- NCC-1701\n");*

*// INIT variables*

*char Z\_char[96] = " ";*

*int G[8][8], C[9][2], K[3][3], N[3], Z[8][8], D[8];*

*int T = (rand()\*20+20)\*100;*

*int T0 = T;*

*int T9 = 25 + rand()\*10;*

*int D0 = 0;*

*return EXIT\_SUCCESS;*

*}*

### *Analysis: Kevin*

After reading through the project guide and downloading the BASIC program, I did a quick review in a text editor. While the program syntax looked different, there were quite a few similarities to other programming languages I’ve used in the past. I then pasted the code into the emulator and immediately noticed the difference in how the code looked. Instead of all white text, the code was colored to allow for ease of readability. After spending some time reading through the guide book, and attempting to play the game, I started working on a spreadsheet that outlined the game commands as a quick reference. While this quickly became obsolete due to increased game play, I eventually added more sheets for the BASIC commands used in the code, a list of line numbers identifying the sub routines, and finally a sheet for the game variables. The latter was then further populated by Ed, making the game code easier to understand.

In order to further break the code into smaller sections, I imported the BASIC code into XCODE and started commenting out the lines of code containing REM (remarks) and adding whitespace between the different subroutines. While this may not be the ideal way to convert the code, it provided a familiar environment, and allowed me to directly work on creating a C code.

### *Analysis: Ed*

I started by reviewing the game play dynamics to try to get a handle on the end results the code generated. After several iterations of game play I reviewed the code while playing through the game. Looking at the code using <https://www.calormen.com/jsbasic/>, an Applesoft BASIC emulator, helped a great deal as it color coded the code allowing me to get even a basic idea of what was going on. With the code color coded I was able to identify what I would identify as functions, each headed by a REM line describing their purpose. When I read deeper into the code, it became painfully clear that trying to track one and two character variables through the code to find what they were for was an exercise in futility. I searched the web for versions of the game ported to other languages, Python & C#, and reviewed their naming conventions and associated them with the BASIC variables.