**Project 2**

Title

**Loot for the Slaying – Last Mission**

Author

**Kevin Vo**

Course

**CSC-11**

Section

**48982**

Due Date

**December 16, 2015**

**Introduction**

Title: “Loot for the Slaying – Last Mission”

                “Loot for the Slaying – Last Mission” is the continuation of Project 1 (“Loot for the Slaying”). It continues as an adventure story game where it tests the luck and memory of the player. After defeating the boss from the game of the first project, you are setup to a turn-based puzzle game where you try and figure out said puzzle before your turns are spent. I am doing this because not only is having fun important, but being able to build upon and retain a good memory is essential for the daily life. To start off, “Last Mission” is a game that permits the player three turns to be able to guess which two blocks has the same value underneath them. These blocks have integer numbers 1 through 6 displayed in the center of them for the player to know which blocks to select. In the game, it indicates to you that within the first input it only accepts an identification integer from the first row and the second input only accepts integers from the second row. Moreover, Every time when you as the player enter a value, it would provide you another number that is a hidden value associated to that particular block. There are three different hidden values and each two of the six blocks on the game have one of those hidden values in common. If you identify the two blocks with the same hidden value before your three turns are up, you win!

**Summary:**

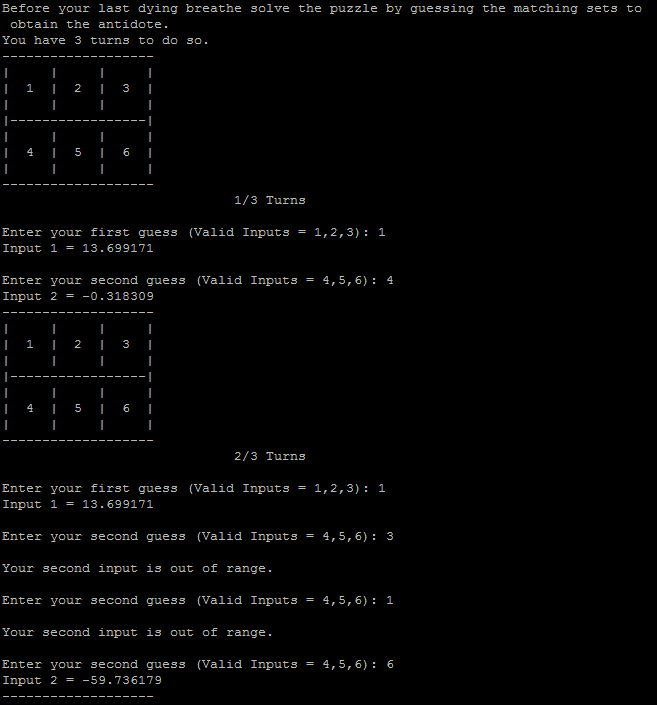
Project Size (C++ version): 375 lines

Number of Variables: ~26

This project meets the criteria for a second project is because it contains a total of greatly 375 in the C++ version which reasonably surpasses the minimum of 250 lines that the criteria suggests. On top of that, it demonstrates the uses of both arrays and floats when looking inside the LastMission.s file. As mentioned in the introduction, this project is an extension of the first project which was what made it fun. On top of the concept of arrays and floats, I used constructs such as do-while, for-loop, if statements, switch, and so on that can be found in all the files. This project was challenge and I felt like I was going to have a panic attack for the first couple of days. I spent a lot of time trying to find help on arrays in assembly and in the end, the best source was the one I had to begin with (which was Thinkingeek.com). Even then it felt kind of fuzzy. However, through luck and looking deeply at an example provided by ThinkInGeek I was able to find out how to store float numbers onto an array. Overall, it took me half a day to figure out what to do, I was thinking and tried Space Invaders for two days as my second project then scraped that since I don’t know anything about making real-time games and the turned-based version of it wasn’t working so well. Finally, I thought of a memory puzzle game that I used to play and figured that was more in my reach. In addition, I figured that since the final product has to be in assembly, it was the best opinion to do a game that I have complete comprehension of unlike blackjack since I have no idea to even play that in real life. For this project unlike the first one, I spent a lot of time writing out the flow of the program on paper so when looking back at the written work and including the C++ code made it a lot easier when transitioning to assembly.

**Description**

**Sample Input/Output**



**Pseudo Code**

*Initialize*

*If player input “y” game will jump to the last mission*

*If the random number > 1 then minus 2 from random number*

*Else display random number*

*If input equals random number add three to player’s damage*

*Else player’s damage remains the same*

*Game ask player to figure out a pattern*

*If player enters the right answer adds ten to player’s damage*

*Else don’t add anything to player’s damager*

*Asks player to do a bonus mission*

*If player agrees to play then game gives player five turns to guess the number*

*If player guess the number correctly then game give ten rounds to fight boss*

*Else the game only gives seven rounds to fight boss*

*Else if player doesn’t want to do bonus then game jumps to boss*

*If player enter the right key then damage is done to boss*

*Else the attack shall miss*

*If boss is dead before round ends player wins*

*Else player losses*

*Unitize last mission of the game*

*Game gives three rounds to figure out puzzle*

*If player is able to match blocks with the same float values game displays win message*

*If three turns are up player will lose*

*Game exits*

**Major Variable**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Variable | Description | Location |
| Floats | globArr[] | Holds the calculated float values to be matched | LastMission.s |
| Single Precision Floats | S8 – S14 | Holds calculated float values to be stored inside array | LastMission.s |
| Double Precision Floats | D15 | Displays float values when needed | LastMission.s |
| Integer | input1 | Accepts int value for first guess | LastMission.s |
|  | input2 | Accepts int value for second guess | LastMission.s |
|  | pDamage | Hold change value of player’s damage | main.cpp |
|  | bot | Stores value for challenge one | main.cpp |
|  | input | Takes player’s input | main.cpp |
|  | bossTurn | Holds number of turns for boss battle | main.cpp |
|  | ansPat | Stores the answer of a generated pattern | Pattern.h |
|  | increment | Keeps track of pattern’s terms | Pattern.h |
|  | patInput | Input for pattern | Pattern.h |
|  | bossHealth | Keeps track of boss’ health | Boss.h |
|  | t | Keeps track of turn | Boss.h |
|  | bossTurn | Maximum turn for boss battle | Boss.h |
| Char | checkA | Stores ‘a’ for comparison | Boss.h |
|  | bossInput | Char input for attacking boss | Boss.h |
|  | checkY | Compares if user agrees to skip to last mission | main.s |
|  | inY | Accepts value for mission selection | main.s |
| Integer | inGuessBonus | Input for the bonus challege | Guess.h |

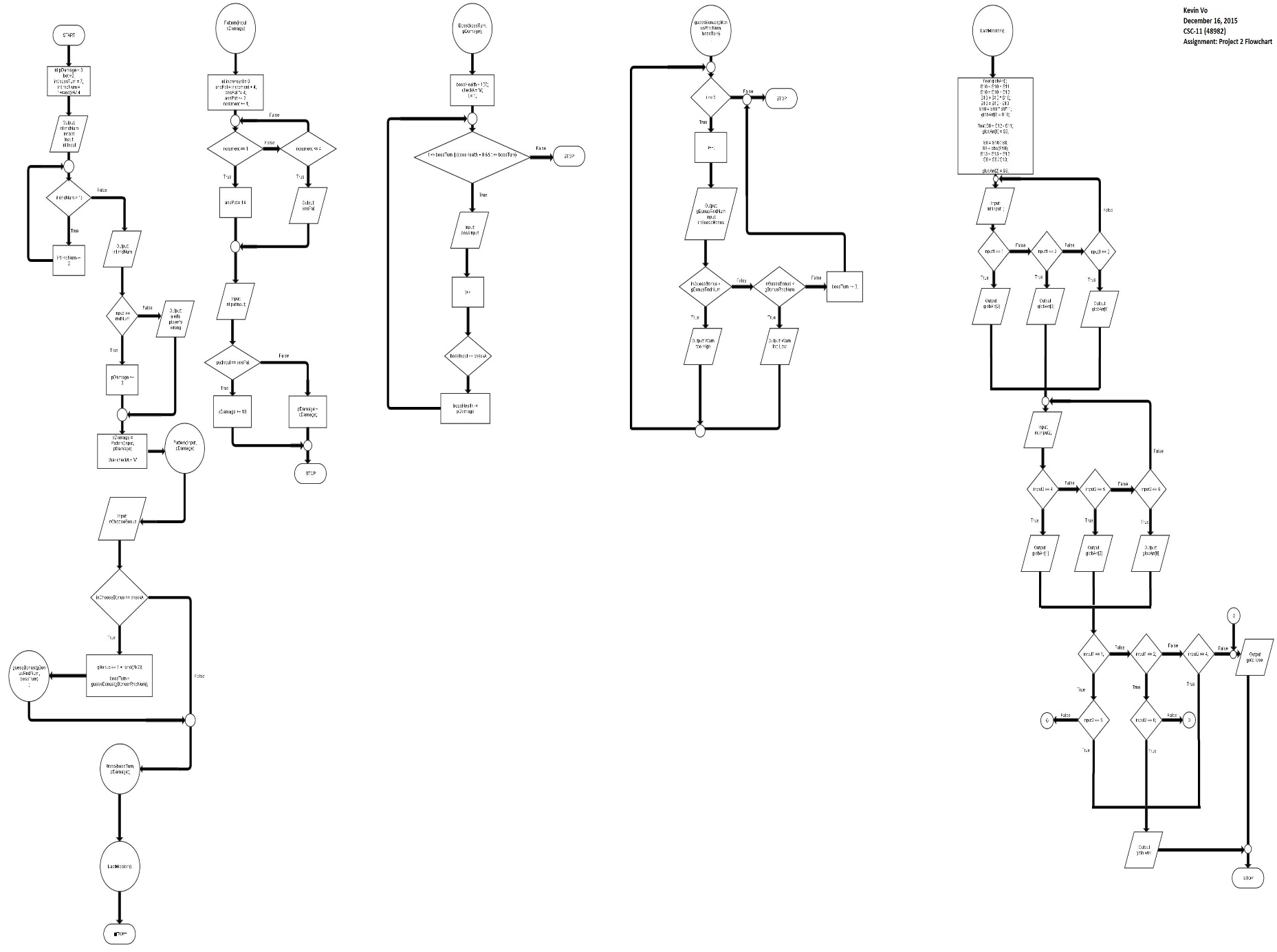
**Language Constructs**

Most of the syntax that I have recently learned were from Chapter 22 in the green book where it discusses the use of floats. It talks of what floats are, how to display it, and most importantly how do arithmetic of it. Some of the two that I was glad to see again were VDIV (on line #94 in LastMission.s) and VSQRT (on line #98 in LastMission.s). On top of that VCVT (on line #159, 167, 175, … in LastMission.s), VLDR(line #166, 174, … LastMission.s) and VSTR (line #88, 92, 100 in LastMission.s), were all the new ones that I have learned. I was figuring out how to store floats into an array and it totally slipped my mind that I need to use vector store. As for array, in thinkgeek.com was helpful on array (line #88, 92, 100 in LastMission.s) in stacks. It reminded me the importance of using the stack pointer to make room in memory and closing it again after when its dynamic activation is complete.

**References**

I borrowed code from the mainDivModFuncV2.s and randTest.s that was present in class on Wednesday into order generation random numbers for Challenge 1 and the bonus challenge. I have also taken commands from Dr. Lehr’s repository for shell files. Eventually, I used the array example from thinkingeek.com and read over about floats in “Raspberry Pi Assembly Language RASPBIAN Beginners: Hands On Guide” where Bruce did a great job on explaining floats and provide a great list of useful vector arithmetic such as VDIV and VSQRT.

**Flowchart:**



**Program (LastMission.s - part of it)**

.data

euler: .float 2.7181281

r: .float 1.318309

one: .float 1.000000

pi: .float 3.14159

st: .float 2.6879

input1e: .asciz "Input 1 = %f\n"

input2e: .asciz "Input 2 = %f\n"

.balign 4

testd: .asciz"R44 = %d\n"

.balign 4

disVal: .asciz "\nThis Input is equal to value: %d\n"

.balign 4

globArr: .skip 12

.balign 4

mIn1: .asciz "\nEnter your first guess (Valid Inputs = 1,2,3): "

.balign 4

mIn2: .asciz "\nEnter your second guess (Valid Inputs = 4,5,6): "

.

.

.

.global LastMission

LastMission:

//PUSH {R4, LR}

STR LR, [SP, #-4]!

SUB SP, SP, #4

MOV R4, #0 //Counter

//Vector loads the single precision values

LDR R1, addr\_st

VLDR S9, [R1]

LDR R1, addr\_euler

VLDR S10, [R1]

LDR R1, addr\_r

VLDR S11, [R1]

LDR R1, addr\_one

VLDR S12, [R1]

LDR R1, addr\_pi

VLDR S13, [R1]

VADD.F32 S10, S11, S10

VADD.F32 S10, S12, S10

//LDR R1, addr\_globArr //R1 <- &globArr

CalAssign:

VMUL.F32 S10, S10, S10 //r^n

VSUB.F32 S10, S13, S10 //(1-r^n)

VMUL.F32 S10, S9, S10 //a \* (1 - r^n)

VSTR S10, [SP, #4]

VSUB.F32 S8, S12, S11 //S8 = 1 - r

VSTR S8, [SP, #8]

VDIV.F32 S8, S10, S8 //S8 = a(1-r^n)/(1-r)

VABS.F32 S8, S8

VSQRT.F32 S8, S8

VSTR S8, [SP, #12]

memGame:

MOV R10, #0 //Set counters

//Displays prompt, continuation of first game

LDR R0, addr\_pLast1

bl printf