Casey Copeland

CSC-11

Survival RPG

12/12/2015

Introduction

Title: Survival RPG

The project is an assembly implementation of very basic coordinate system and battle system. These systems are key elements of a huge range of RPGs. Once set up, these systems are easy to build upon. For example, a shop could be added later or a more complex map system to make exploring a much more diverse world possible.

Summary

The c code version is about 200 lines of code, while the assembly version is about 500 lines. This 500 lines does not include the modulus function imported from a previous homework. The most challenging aspect was managing the string that was the visual aspect of the map. Registers are designed to deal with 4 bytes at a time, so writing to just one byte requires some byte manipulation to not change the values of the surrounding 3 bytes. As such, the use of bitwise operators were required for that. To not change the values of the surrounding 3 bytes, they were copied and combined with the single byte that was to be written. This causes the surrounding 3 bytes to be overwritten with the same value they started with before the write. For a non-threaded program, this is equivalent to writing 1 byte. For the combination of the 3 bytes and 1 byte, the inclusive or bitwise operator was used after ensuring the other bytes of each registered were zeroed out. Zeroing out the other bytes of the register was achievable with bit shifting. The one useful aspect of the 4 byte register dealing with an array of bytes is that multiple bytes of the array can be written at once. This is used during the initialization of the map since they are all set to the same character. For the entire project, it took a few days for the assembly code. The use of memory addresses to access structures as memory blocks complicated the assembly code greatly in comparison to c code. While working with the floating point processor, the minor nuisance of a bus error occurred for vldr. This caused a little confusion at first. Had it been a

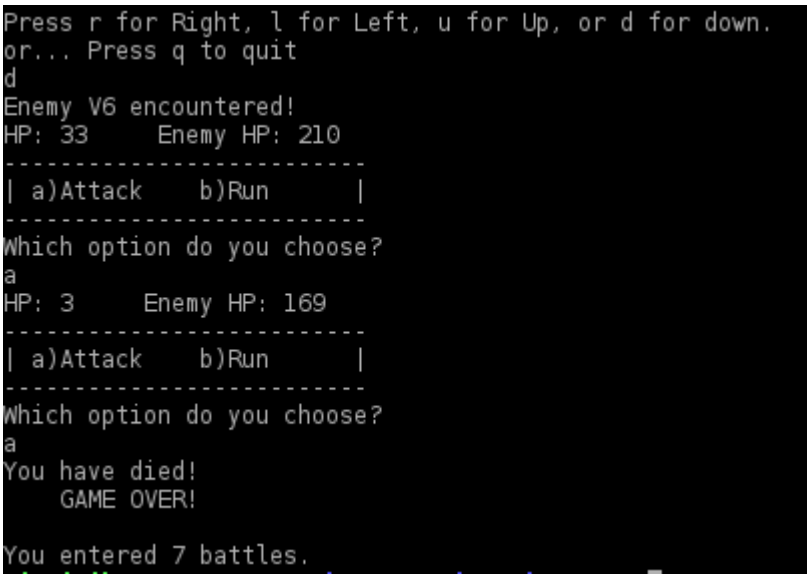
regular ldr operation, it would've just segfaulted. Gdb easily found where it occurred once the backtrace was looked at for the bus error. In addition, I had to push onto the stack for a call to printf once i had switched the program to using floats. This is due to both printf require doubles and how each double must start in an even numbered register. Since register 5 is outside the calling c calling convention, this leaves register 4 as ineligible for storing doubles for c function calls. As a result, the float had to be pushed onto the stack.

Description

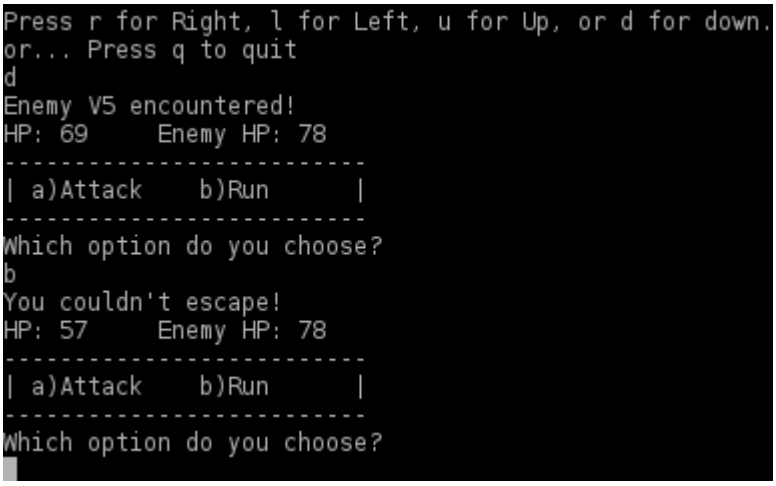
The program was developed piece by piece. The first piece created was the overall control flow. I checked to make sure that the while loop in main worked and would exit. After the general flow of it was laid out, more and more logic would be implemented. It went from creating battlers, to creating the battle function, and then finally to the text visual of seeing yourself traveling through the map. When errors occurred, gdb was used a lot to find where the segfault occurred. To make the code manageable, commenting was used to provide additional information. In particular, structures were commented to keep track of which order the variables were in an a structure. For functions, arguments required were typically commented to keep in mind which input it's meant to work with.

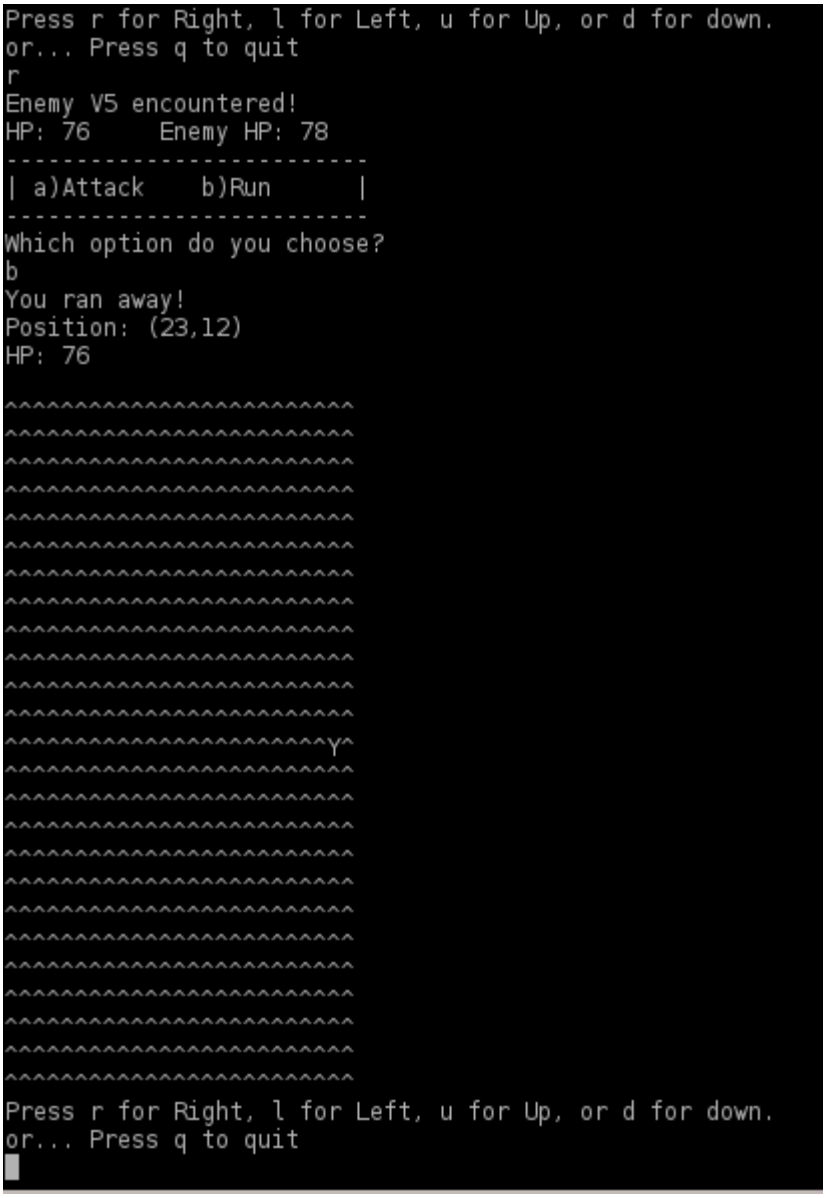
Screenshots

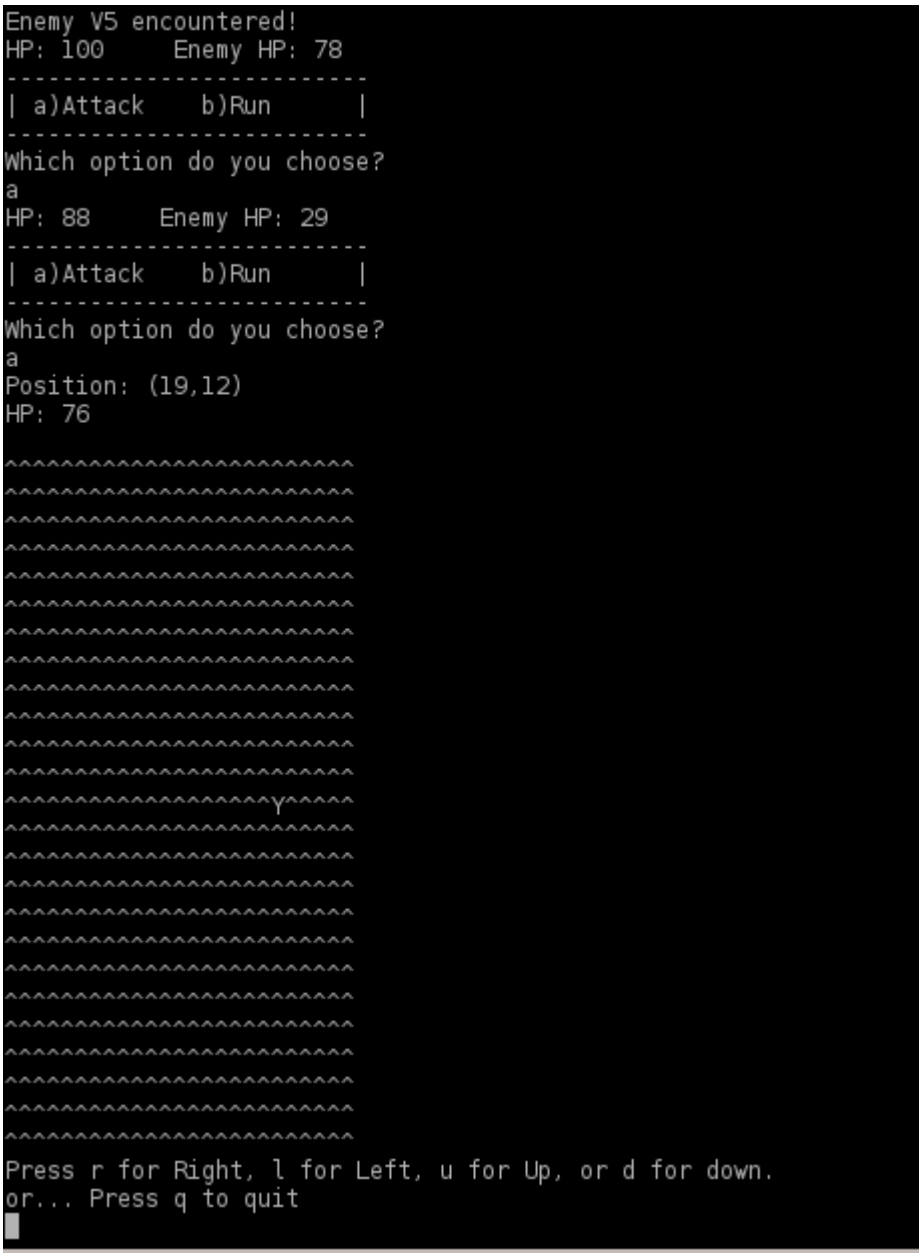
```
Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit
r
Position: (18,12)
HP: 100

^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^Y^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^
Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit
r
Enemy V5 encountered!
HP: 100     Enemy HP: 78
-------------------------
| a)Attack    b)Run      |
-------------------------
Which option do you choose?
```


Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit
d
Enemy V6 encountered!
HP: 33 Enemy HP: 210
| a)Attack b)Run |
Which option do you choose?
a
HP: 3 Enemy HP: 169
| a)Attack b)Run |
Which option do you choose?
a
You have died!
GAME OVER!
You entered 7 battles.

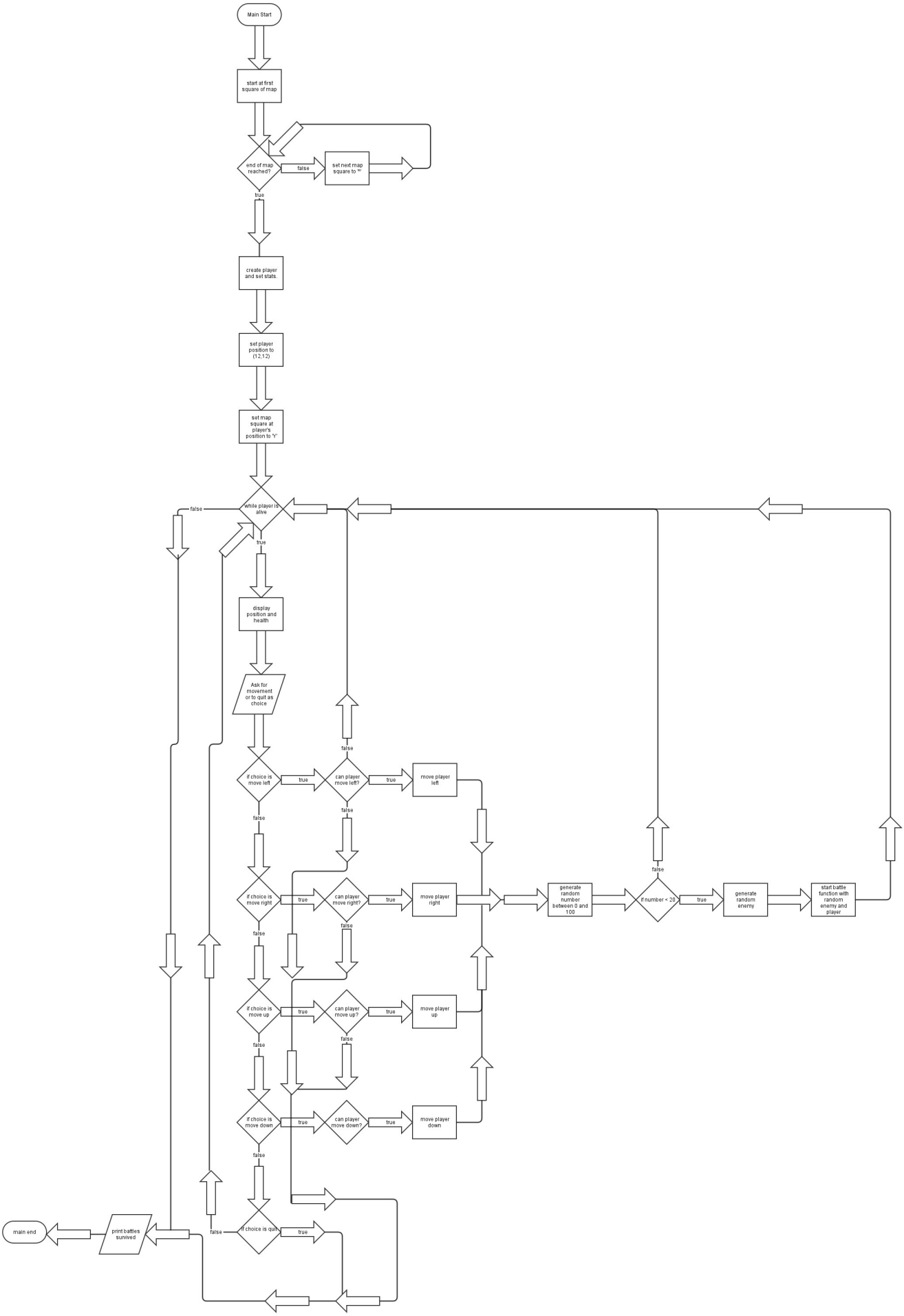
pi@minibun ~/CSC11/raspberry/Proj/Project 1 $ ./game
Position: (12,12)
HP: 100
Y
Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit
Y
Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit

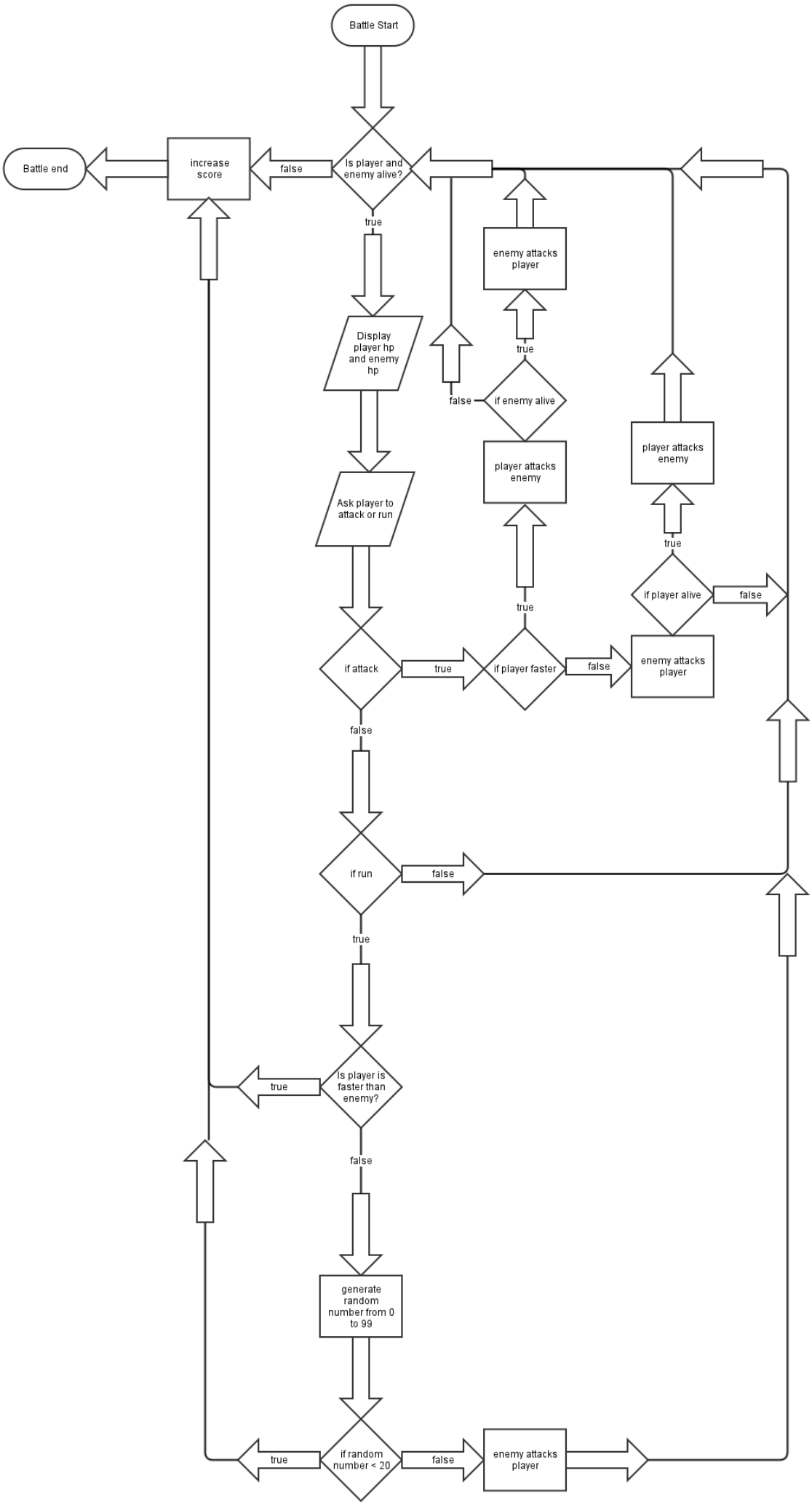

Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit
d
Enemy V5 encountered!
HP: 69 Enemy HP: 78
| a)Attack b)Run |
Which option do you choose?
b
You couldn't escape!
HP: 57 Enemy HP: 78
| a)Attack b)Run |
Which option do you choose?


Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit
r
Enemy V5 encountered!
HP: 76 Enemy HP: 78
| a)Attack b)Run |
Which option do you choose?
b
You ran away!
Position: (23,12)
HP: 76
Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit


Enemy V5 encountered!
HP: 100 Enemy HP: 78
| a)Attack b)Run |
Which option do you choose?
a
HP: 88 Enemy HP: 29
| a)Attack b)Run |
Which option do you choose?
a
Position: (19,12)
HP: 76
Press r for Right, l for Left, u for Up, or d for down.
or... Press q to quit

Flow Chart


Main Start
start at first square of map
end of map reached?
false
set next map square to '0'
true
create player and set stats.
set player position to (12,12)
set map square at player's position to 'Y'
while player is alive
false
true
display position and health
Ask for movement or to quit as choice
false
If choice is move left
true
can player move left?
true
move player left
false
false
If choice is move right
true
can player move right?
true
move player right
false
false
generate random number between 0 and 100
If number < 20
false
true
generate random enemy
start battle function with random enemy and player
If choice is move up
true
can player move up?
true
move player up
false
false
If choice is move down
true
can player move down?
true
move player down
false
If choice is quit
false
true
print battles survived
main end


Battle Start
Is player and enemy alive?
false
increase score
Battle end
true
Display player hp and enemy hp
Ask player to attack or run
if attack
true
if player faster
false
enemy attacks player
true
if player alive
false
player attacks enemy
true
if enemy alive
false
enemy attacks player
false
if run
false
true
Is player is faster than enemy?
true
false
generate random number from 0 to 99
if random number < 20
true
false
enemy attacks player

Pseudo Code

```
map is an array of ASCII characters
position is a point (x,y)
stats are hp, attack, defense, and speed
battlers are a combination of stats, position, and a name
hit (attacker, receiver) {
        if attacker's attack not greater than the receiver's defense:
                receiver lost 1 hp
        else:
                receiver's hp drops by the difference between the two
}
battle (player, enemy) {
print enemy encountered
while enemy and player are alive:
        ask player whether to run or attack
        if attack:
                if player faster:
                        have player hit enemy
                        if enemy alive:
                                have enemy hit player
                else:
                        have enemy hit player
```

```
                    if player alive:
                        have player hit enemy
        if run:
            if run successful
                print run successful
                quit battle
            else have enemy hit player
                print run failed
                have enemy hit player
}
display_map (map) {
for each square in map, print square
}
main() {
Both player and enemy are battlers.
while player is alive:
        display position, hp, and map
        ask player which option he/she chooses
        if player choses to move left, move player left
        if player chooses to move right, move player right
        if player chooses to move up, move player up
        if player chooses to move down, move player down
        if player chooses to quit, print battles encountered and quit
        if player has chosen any of the movements above,
            if player would've moved outside the map
```

prevent the move

else

randomly decide if to generate a random enemy.

If random enemy generated,

start battle between player and random enemy

if player died

print game over

print battles encountered

quit

}

Major Variables

Type	Variable Name	Description	Location
Byte array	map	Visual representation of Map as text	main.s
word	score	Number of battles encountered	main.s
Battler Array	battlers	Array of all battlers needed to be allocated at once. One for the player and one for the enemy.	main.s

Structures

Coordinate: Contains the position of something on the map.

word X

word Y

Stats: Contains all the stats necessary for calculations in battle.

float HP

float Attack

float Defense

float Speed

Battler: A structure containing all the data necessary for the main player as well as enemies.

Stats stats

Coordinate position

String name

Language Constructs

C-level view of assembly

for loop: used to initialize the map

while loop: used for main loop and battle loop

if else: used in many places, such as checking if a player is faster than the enemy during attempting to run

switch-case: used for choices in main loop and battle

function calls: used in many places, such as for printing, and for doing battling

assembly level view

branching: used everywhere, including main loop

printf: used for printing, such as in main loop

scanf: used for getting input, such as in reply to asking what to do in main loop

mov: used everywhere, including main loop

.balign 4: used for most data to align it

.global: used to make a function or data accessible to other assembly files during linking

push: used for functions that required more than 4 arguments, such as initBattler

pop: used for same reason as pop, but to take out the arguments instead of placing them

ldr: used in many places to load from memory or to load values too big for a mov

add: used for adding, such as pointers offset

sub: used for subtracting, such as decreasing the X coordinate of the player

lsl: used to shift bits around for dealing with the map, which was a byte arrray

lsr: same as lsl, but shifts bits to the right instead of left

orr, used to combine the bits to two registers when dealing with the map, one register holding 3 bytes that are already in the map, and the other holding the byte to write. Mnemonic for bitwise operator inclusive or.

.macro/.endm: used to create a copy and paste macro similar to what the c preprocessor can do.

labels: used to name sections of code to jump to in a branch

.asciiz: used to make null terminated strings

.ascii: used to make a string

.byte: used to put my name into the executable as seen in the end of main.s. Also used for data such as inputChar

mul: used for multiplication for finding which map index an (x,y) coordinate is in mapCoordinateToIndex

vcmp: used to compare speeds in the float point processor

vcvt: used to convert integer values for the initialization of the main player. Also used to convert floats into doubles for use in a printf call.

vldr: used to load speed into the float point processor

vmrs: used to move status flags from speed comparison to the regular status flags so they can be used for later branching

vsub: used for subtracting, such as decreasing hp

memory addresses: used to refer to allocated variables with label names

.text: used to denote code section and constants

.data: used to denote data section

.include: used to include other assembler files that contain macros and symbols that the linker doesn't deal with

.skip: used to allocate map as a byte array

References

Code from the modulus homework was placed into the project to use it as part of the range for random number generation and for the display of the map to make sure each row gets printed on it's own line. In addition to that, a few functions from the standard c library were used. The c functions used were printf, scanf, getchar, srand, and rand. printf and scanf were imported for the purpose of output and input respectively. The fucntion getchar was necessary to keep the input and output working as expected by clearing out any newlines left by scanf. For random values, srand, rand, and time was imported so that I could make enemy encounters random.

Program

attack.s

```
.balign 4
.text

.include "battler.s"

.global attack
/*args
   R0 is pointer to attacker
   R1 is pointer to receiver*/
attack:
```

```
        push {R2, LR}
        /*HP*/
        vldr S0, [R1]
        /*Defense*/
        vldr S1, [R1,#8]
        /*Attack*/
        vldr S2, [R0,#4]
        vcmp.f32 S2, S1
        vmrs APSR_nzcv, FPSCR
        blt oneDamage
        /*normal damage calculation*/
        vsub.f32 S3, S2, S1
        vsub.f32 S0, S0, S3
        b attackReturn
oneDamage:
        /*subtract 1.0 health*/
        mov R2, #0x3f800000
        vmov S3, R2
        vsub.f32 S0, S0, S3
attackReturn:
        vstr S0, [R1]
        ldr R2, [R1,#(sizeOfStats+sizeOfCoord)]
        ldr R1, [R0,#(sizeOfStats+sizeOfCoord)]
        ldr R0, =attackMsg
```

```
        bl printf
        pop {R2, LR}
        bx LR


attackMsg:
.asciz "%s attacked %s!\n"
```

battle.s

```
.balign 4
.text
.global battle

.include "battler.s"
.include "usefulMacros.s"

/* args:
        R0 is pointer to controllable battler
        R1 is pointer to AI battler */
battle:
        /*save return address*/
        push {IP, LR}

        mov R5, R0
        mov R6, R1
        ldr R0, =encounterMessage
        add R1, #(sizeOfStats+sizeOfCoord)
```

```
        ldr R1, [R1]
        bl printf
battleLoop:
        /*check if enemy and player are still alive*/
        vldr S1, [R6]
        vcmp.f32 S1, #0
        vmrs APSR_nzcv, FPSCR
        ble battleLoopEnd
        vldr S0, [R5]
        vcmp.f32 S0, #0
        vmrs APSR_nzcv, FPSCR
        ble battleLoopEnd
        /*display battle menu*/
        ldr R0, =battleLoopMessage
        /*push enemy hp onto stack for printf due to calling
         convention only using up to R4*/
        vldr S0, [R6]
        vcvt.f64.f32 D1, S0
        vmov R2, R3, D1
        push {R2, R3}
        vldr S0, [R5]
        vcvt.f64.f32 D1, S0
        vmov R2, R3, D1
        bl printf
        pop {R2, R3}
```

```
        ldr R0, =inputFormat
        sub SP, #8
        mov R1, SP
        bl scanf
        bl getchar
        /* compare input to A */
        compareBothCase 0x41 attackEnemy
        /* compare input to B */
        compareBothCase 0x42 run
        add SP, #8
        b battleLoop
attackEnemy:
        /*compare speed*/
        vldr S0, [R5,#12]
        vldr S1, [R6,#12]
        vcmp.f32 S0, S1
        vmrs APSR_nzcv, FPSCR
        /*player attacks first if faster*/
        movge R0, R5
        movge R1, R6
        /*else enemy attacks first*/
        movlt R1, R5
        movlt R0, R6
        push {R0, R1}
        bl attack
```

```
        pop {R0, R1}
        /*checks if receiver of attack died*/
        vldr S0, [R1]
        vcmp.f32 S0, #0
        vmrs APSR_nzcv, FPSCR
        ble battleLoopEnd
        /*counter attack if alive*/
        mov R2, R0
        mov R0, R1
        mov R1, R2
        bl attack
        b battleLoop
run:
        /*compare speed*/
        mov R0, R5
        vldr S0, [R0,#12]
        mov R1, R6
        vldr S1, [R1,#12]
        vcmp.f32 S0, S1
        vmrs APSR_nzcv, FPSCR
        bgt runSuccess
        bl rand
        mov R1, #100
        bl mod
        /* 20 percent chance of running when slower*/
```

```
        cmp R0, #20
        blo runSuccess
        ldr R0, =runFailMsg
        bl printf
        mov R1, R5
        mov R0, R6
        bl attack
        b battleLoop
runSuccess:
        ldr R0, =runSuccessMsg
        bl printf
battleLoopEnd:
        ldr R0, =score
        ldr R1, [R0]
        add R1, R1, #1
        str R1, [R0]
        /*return*/
        pop {IP, LR}
        bx LR

.balign 4
encounterMessage:
        .asciz "%s encountered!\n"
.balign 4
battleLoopMessage:
```

```
        .ascii "HP: %f     Enemy HP: %f\n"
        .ascii "-------------------------\n"
        .ascii "| a)Attack    b)Run      |\n"
        .ascii "-------------------------\n"
        .asciz "Which option do you choose?\n"
.balign 4
inputFormat: .asciz "%c"
runFailMsg: .asciz "You couldn't escape!\n"
runSuccessMsg: .asciz "You ran away!\n"

.global printf
.global scanf
.global getchar
.global rand
```

battler.s

```
/* File for inclusion to make symbols available to assembler
 (.global is for making symbols available to linker) */

/* STRUCTURES
--------------------------------*/
/* coordinate structure
        1 word X
        1 word Y*/
sizeOfCoord=8
/* stats structure
```

```
        1 float HP
        1 float Attack
        1 float Defense
        1 float Speed */
sizeOfStats=16
/* battler structure layout
   1 stats
   1 coordinate position
   1 word (address to string) name */
sizeOfBattler=sizeOfStats+sizeOfCoord+4
```

battlerInit.s

```
.balign 4
.text
.include "battler.s"

/* Init Functions */
.global initCoord
.global initBattler
.global initStats

/*args:
        R0 is memory pointer to coordinate structure
        R1 is X value
        R2 is Y value
*/
```

```
initCoord:
        /* X = 0 */
        str R1, [R0]
        /* Y = 0 */
        str R2, [R0,#4]
        bx LR

/*args:
        R0 is memory pointer to stats structure
        R1 is HP
        R2 is Attack
        R3 is Defense
        on stack: Speed
*/
initStats:
        /*set HP*/
        str R1, [R0]
        /*set Attack*/
        str R2, [R0,#4]
        /*set Defense */
        str R3, [R0,#8]
        mov R5, R0
        pop {R0, IP}
        /*set Speed */
        str R0, [R5,#12]
```

```
        bx LR


/*args:
        R0 is memory pointer to battler structure
        R1 is coordinate X
        R2 is coordinate Y
        R3 is pointer to name
        on stack: Speed, HP, Attack,Defense
*/
initBattler:
        /* save return address */
        push {IP, LR}

        add R0, R0, #sizeOfStats
        bl initCoord
        add R0, R0, #sizeOfCoord
        str R3, [R0]
        sub R0, R0, #(sizeOfStats+sizeOfCoord)
        mov R5, R0
        pop {IP,LR}
        pop {R0, R1, R2, R3}
        push {IP,LR}
        push {R0, IP}
        mov R0, R5
        bl initStats
```

```
	/* return */
	pop {IP, LR}
	bx LR
```

main.s

```
.data

.global score
.balign 4
score: .word 0

mapBoundHigh = 25

.balign 4
/*map is a square that's top left tile or coordinate is (0,0)*/
map: .skip mapBoundHigh*mapBoundHigh

/* memory allocation for battlers */
.balign 4
battlers: .skip sizeOfBattler*2

.balign 4
.text
.include "battler.s"
.include "usefulMacros.s"
	.global main
```

```
/*returns back to main loop after battle*/
.macro initBattle HP, Attack, Defense, Speed, NameAddress
        ldr R1, =\HP
        ldr R2, =\Attack
        ldr R3, =\Defense
        ldr R0, =\Speed
        push {R0, R1, R2, R3}
        ldr R0, =(battlers+sizeOfBattler)
        mov R1, #0
        mov R2, #0
        ldr R3, =\NameAddress
        bl initBattler
        ldr R1, =(battlers+sizeOfBattler)
        ldr R0, =battlers
        ldr LR, =mainLoop
        b battle
.endm
.macro moveSwapMapBytes difference
        ldr R0, =(battlers+sizeOfStats)
        mov R1, #mapBoundHigh
        bl mapCoordinateToIndex
        ldr R1, =map
        add R0, R0, R1
        add R2, R0, #\difference
```

```
        #swap characters in map
        ldr R1, [R0]
        ldr R3, [R2]
        str R1, [R2]
        str R3, [R0]
.endm

main:
        /* save return address*/
        push {IP, LR}

        /*initialization*/
        mov R0, #0
        bl time
        bl srand
        /* initialize map*/
        ldr R0, =map
        /* ASCII '^' */
        mov R1, #0x5E
        ldr R2, =(mapBoundHigh*mapBoundHigh)
        bl mapInit
        /* initializes main player*/
        /*HP*/
        mov R1, #100
        vmov S1, R1
```

```
vcvt.f32.u32 S0, S1

vmov R1, S0

/*Attack*/

mov R2, #51

vmov S1, R2

vcvt.f32.u32 S0, S1

vmov R2, S0

/*Defense*/

mov R3, #5

vmov S1, R3

vcvt.f32.u32 S0, S1

vmov R3, S3

/*Speed*/

mov R0, #6

vmov S1, R0

vcvt.f32.u32 S0, S1

vmov R0, S0

push {R0, R1, R2, R3}

ldr R0, =battlers

/*Position*/

mov R1, #12

mov R2, #12

ldr R3, =playerName

bl initBattler

ldr R0, =(battlers+sizeOfStats)
```

```
        mov R1, #mapBoundHigh
        bl mapCoordinateToIndex
        ldr R1, =map
        add R1, R1, R0
        /*shifts map pointer over 3 bytes so write places
          'Y' in correct byte*/
        sub R1, R1, #3
        /*load map bytes for later inclusive or*/
        ldr R2, [R1]
        lsl R2, #8
        lsr R2, #8
        /* player represented by 'Y' */
        mov R3, #0x59
        lsl R3, #24
        /*keep map bytes since registers are 4 bytes*/
        orr R3, R2
        str R3, [R1]

mainLoop:
        ldr R6, =battlers
        vldr S0, [R6]
        vcmp.f32 S0, #0
        vmrs APSR_nzcv, FPSCR
        ble gameOver
        ldr R0, =mainLoopStatusMessage
```

```
vldr S0, [R6]
vcvt.f64.f32 D1, S0
vmov R4, R5, D1
/* R5 outside standard calling convention,
   so push onto stack for printf*/
push {R4, R5}
add R6, R6, #sizeOfStats
ldr R1, [R6]
add R6, R6, #(sizeOfCoord/2)
ldr R2, [R6]
bl printf
pop {R4, R5}
ldr R0, =map
mov R1, #mapBoundHigh
ldr R2, =(mapBoundHigh*mapBoundHigh)
bl mapDisplay
ldr R0, =mainLoopControlsMessage
bl printf
ldr R0, =mainInputFormat
sub SP, #8
mov R1, SP
bl scanf
/* remove newline still in buffer
(newline in buffer will printf to print twice when loop repeats)*/
bl getchar
```

```
        /* compare to ASCII L */
        compareBothCase 0x4C, moveLeft
        /* compare to ASCII R */
        compareBothCase 0x52, moveRight
        /* compare to ASCII U */
        compareBothCase 0x55, moveUp
        /* compare to ASCII D */
        compareBothCase 0x44, moveDown
        /* compare to ASCII Q*/
        compareBothCase 0x51, endMainLoop
        add SP, #8
        b mainLoop
moveLeft:
        ldr R2, =(battlers+sizeOfStats)
        ldr R1, [R2]
        sub R3, R1, #1
        /*branch on overflow (subtracting from 0)*/
        cmp R3, R1
        bhi mainLoop
        str R3, [R2]
        moveSwapMapBytes 1
        b chanceEncounter
moveRight:
        ldr R2, =(battlers+sizeOfStats)
```

```
        ldr R1, [R2]
        add R3, R1, #1
        cmp R3, #(mapBoundHigh-1)
        bhi mainLoop
        str R3, [R2]
        moveSwapMapBytes (-1)
        b chanceEncounter
moveDown:
        ldr R2, =(battlers+sizeOfStats+sizeOfCoord/2)
        ldr R1, [R2]
        add R3, R1, #1
        cmp R3, #(mapBoundHigh-1)
        bhi mainLoop
        str R3, [R2]
        moveSwapMapBytes (-mapBoundHigh)
        b chanceEncounter
moveUp:
        ldr R2, =(battlers+sizeOfStats+sizeOfCoord/2)
        ldr R1, [R2]
        sub R3, R1, #1
        /*branch on overflow (subtracting from 0)*/
        cmp R3, R1
        bhi mainLoop
        str R3, [R2]
        moveSwapMapBytes mapBoundHigh
```

```
        b chanceEncounter
chanceEncounter:
        bl rand
        mov R1, #100
        bl mod
        /* 5 percent chance of enemy encounter */
        cmp R0, #15
        blo genEnemy
        b mainLoop
genEnemy:
        bl rand
        mov R1, #6
        bl mod
enemyOne:
        cmp R0, #0
        bne enemyTwo
        /*50.0, 11.5, 2.0, 5.0*/
        initBattle 0x42480000 0x41380000 0x40000000 0x40A00000 enemyOneName
enemyTwo:
        cmp R0, #1
        bne enemyThree
        /*55.0, 12.5, 2.0, 5.1*/
        initBattle 0x425C0000 0x41480000 0x40000000 0x40A33333 enemyTwoName
enemyThree:
        cmp R0, #2
```

```
        bne enemyFour
        /*60.0, 13.5, 3.0, 5.2*/
        initBattle 0x42700000 0x41580000 0x40400000 0x40A66666 enemyThreeName
enemyFour:
        cmp R0, #3
        bne enemyFive
        /*67.0 15.5 5.0 6.0*/
        initBattle 0x42860000 0x41780000 0x40a00000 0x40c00000 enemyFourName
enemyFive:
        cmp R0, #4
        bne enemySix
        /*78.0, 17.5, 2.0, 7.0*/
        initBattle 0x429C0000 0x418C0000 0x40000000  0x40E00000 enemyFiveName
enemySix:
        /*210.0, 35.0, 10.0, 8.0*/
        initBattle 0x43520000 0x420C0000 0x41200000 0x41000000  enemySixName
gameOver:
        ldr R0, =deathMessage
        bl printf
endMainLoop:
        ldr R0, =scoreMessage
        ldr R1, =score
        ldr R1, [R1]
        bl printf
        /*return*/
```

```
        pop {IP, LR}
        bx LR


/*constants*/
.balign 4
mainLoopStatusMessage:
        .ascii "Position: (%d,%d)\n"
        .asciz "HP: %f\n"
mainLoopControlsMessage:
        .ascii "\nPress r for Right, l for Left, u for Up, or d for down.\n"
        .asciz "or... Press q to quit\n"
deathMessage:
        .asciz "You have died!\n    GAME OVER!\n\n"
scoreMessage:
        .asciz "You entered %d battles.\n"
.balign 4
mainInputFormat: .asciz "%c"


.global printf
.global scanf
.global getchar
.global srand
.global rand
.global time
```

```
.balign 4
playerName: .asciz "Hero"
.balign 4
enemyOneName: .asciz "Enemy V1"
.balign 4
enemyTwoName: .asciz "Enemy V2"
.balign 4
enemyThreeName: .asciz "Enemy V3"
.balign 4
enemyFourName: .asciz "Enemy V4"
.balign 4
enemyFiveName: .asciz "Enemy V5"
.balign 4
enemySixName: .asciz "Enemy V6"


.byte 0x43, 0x61, 0x73, 0x65, 0x79, 0x20, 0x20, 0x43
.byte 0x6F, 0x70, 0x65, 0x6C, 0x61, 0x6E, 0x64, 0x00
```

map.s

```
.balign 4
.text
.global mapInit
/* args:
        R0 = address to char array of map to fill
        R1 = character to fill array of map with
        R2 = size of map*/
```

```
mapInit:
        mov R3, R0
        add R3, R3, R2
        mov R4, R1
        /*R1 is 4 bytes, so fill all 4 bytes with same pattern*/
        lsl R4, #8
        orr R1, R4
        mov R4, R1
        lsl R4, #16
        orr R1, R4
mapInitLoop:
        cmp R0, R3
        bhs mapInitEnd
        str R1, [R0]
        add R0, R0, #4
        b mapInitLoop
mapInitEnd:
        bx LR


.global mapDisplay
/* args:
        R0 is address to char array of map
        R1 is map width
        R2 is map size */
mapDisplay:
```

```
        push {IP, LR}
        /* stack:
          pointer [SP,#0],
          start of map #4,
          end #8,
          width #12,
          map size #16 */
        sub SP, #24
        str R0, [SP]
        str R0, [SP,#4]
        add R0, R0, R2
        str R0, [SP,#8]
        str R1, [SP,#12]
        str R2, [SP,#16]
        ldr R0, [SP]
mapDisplayLoop:
        /*if pointer reached end, exit*/
        ldr R3, [SP,#8]
        cmp R0, R3
        beq mapDisplayEnd
        /* if end of row reached, print new line */
        ldr R1, [SP,#4]
        sub R0, R0, R1
        ldr R1, [SP,#12]
        bl mod
```

```
        cmp R0, #0
        bne printMapSquare
        ldr R0, =newLineMessage
        bl printf
printMapSquare:
        ldr R0, =mapSquareMessage
        ldr R1, [SP]
        ldr R1, [R1]
        bl printf
        /*increment pointer*/
        ldr R1, [SP]
        add R1, R1, #1
        str R1, [SP]
        mov R0, R1
        b mapDisplayLoop
mapDisplayEnd:
        add SP, #24
        pop {IP, LR}
        bx LR

.global mapCoordinateToIndex
/*args:
        R0 is pointer to coordinate
        R1 is map width*/
mapCoordinateToIndex:
```

```
        mov R2, R0
        add R2, R2, #4
        ldr R0, [R0]
        ldr R2, [R2]
        mul R3, R2, R1
        add R0, R0, R3
        bx LR

.global mapSwapCoordinates
/*args:
        R0 is pointer to coordinate
        R1 is pointer to other coordinate
        R2 is map width*/
mapSwapCoordinates:
        push {IP, LR}
        sub SP, #8
        str R2, [SP]
        mov R4, R1
        mov R1, R2
        bl mapCoordinateToIndex
        mov R3, R0
        mov R0, R4
        mov R4, R3
        ldr R1, [SP]
        bl mapCoordinateToIndex
```

```
        /*perform swap based on indexes*/
        ldr R2, [R4]
        ldr R1, [R0]
        str R2, [R0]
        str R1, [R4]
        add SP, #8
        pop {IP, LR}
        bx LR

mapSquareMessage: .asciz "%c"
newLineMessage: .asciz "\n"
```

mod.s

```
.text
.global mod

/* args:
        R0 unsigned value being divided, remainder should be in here at end
        R1 unsigned divisor, assumed to be greater than 1 */
mod:
        mov R2, R1
        cmp R1, R0
        bgt end
        b mod_start
/*bit shift left*/
modLoop:
```

```
        cmp R1, R0
        bhi result
mod_start:
        sub R0, R0, R1
        lsl R1, #1
        b modLoop
/*subtract and bit shift right*/
result:
        lsr R1, #1
        cmp R1, R2
        blo end
        cmp R1, R0
        bhi result
        sub R0, R0, R1
        b result
end:
        bx LR
```

usefulMacros.s

```
/* assumes char to compare has been put on the stack
   pops off the stack for you if it jumps to branch */
.macro compareBothCase upperChar, trueBranch
        ldr R0, [SP]
        cmp R0, #\upperChar
        beq \trueBranch
        cmp R0, #(\upperChar+0x20)
```

```
        addeq SP, #8
        beq \trueBranch
.endm
```

main.c

```
#include<stdio.h>
#include<time.h>

#define MAP_WIDTH 25
#define MAP_SIZE 625

char map [MAP_SIZE];
int score = 0;

struct Coord {
        int x;
        int y;
};

struct Stats {
        float hp;
        float attack;
        float defense;
        float speed;
};
```

```
struct Battler {
	struct Stats stats;
	struct Coord pos;
	const char* name;
};


void attack (struct Battler* attacker,struct Battler* receiver) {
	if (attacker->stats.attack > receiver->stats.defense) {
		receiver->stats.hp-=attacker->stats.attack-receiver->stats.defense;
	} else {
		receiver->stats.hp-=1.0f;
	}
}
void battle (struct Battler* player,struct Battler* enemy) {
	char choice;
	int run;
	int odds;
	printf("%s encountered!\n",enemy->name);
	while (1) {
		if (player->stats.hp <= 0) {
			score+=1;
			return;
		} else if (enemy->stats.hp <= 0) {
			score+=1;
			return;
```

```
}
printf("HP: %f     Enemy HP: %f\n",player->stats.hp,enemy->stats.hp);
printf("------------------------\n");
printf("|  a)Attack    b)Run    |\n");
printf("------------------------\n");
printf("Which option do you choose?\n");
scanf("%c",&choice);
getchar();
switch(choice) {
        case 'a':
        case 'A':
                if (player->stats.speed > enemy->stats.speed) {
                        attack(player,enemy);
                        if (enemy->stats.hp > 0) {
                                attack(enemy,player);
                        }
                } else {
                        attack(enemy,player);
                        if (player->stats.hp > 0) {
                                attack(player,enemy);
                        }
                }
                break;
        case 'b':
        case 'B':
```

```
                        if (player->stats.speed > enemy->stats.speed) {
                                printf("You ran away!\n");
                                run = 1;
                                score+=1;
                        } else {
                                odds = rand() % 100;
                                if (odds < 20) {
                                        printf("You ran away!\n");
                                        run = 1;
                                        score+=1;
                                } else {
                                        printf("You couldn't escape!\n");
                                        attack(enemy,player);
                                }
                        }
                        break;
                }
                if (run == 1) {
                        break;
                }
        }
}

void initBattle (struct Battler* player, float hp, float attack,
                float defense, float speed, const char* name) {
```

```
	struct Battler enemy;
	enemy.stats.hp = hp;
	enemy.stats.attack = attack;
	enemy.stats.defense = defense;
	enemy.stats.speed = speed;
	enemy.name = name;
	battle(player,&enemy);
}

void chanceEncounter (struct Battler* player) {
	int odds = rand() % 100;
	if (odds < 15) {
		odds = rand() % 6;
		switch(odds) {
			case 0:
				initBattle(player,50.0f,11.5f,2.0f,5.0f,"Enemy V1");
				break;
			case 1:
				initBattle(player,55.0f,12.5f,2.0f,5.1f,"Enemy V2");
				break;
			case 2:
				initBattle(player,60.0f,13.5f,3.0f,5.2f,"Enemy V3");
				break;
			case 3:
				initBattle(player,67.0f,15.5f,5.0f,6.0f,"Enemy V4");
```

```
                        break;
                case 4:
                        initBattle(player,78.0f,17.5f,2.0f,7.0f,"Enemy V5");
                        break;
                case 5:
                        initBattle(player,210.0f,35.0f,10.0f,8.0f,"Enemy V6");
                        break;
            }
        }
}

int main() {
        struct Battler player;
        char choice;
        int i;
        srand(time(NULL));
        for (i=0;i<MAP_SIZE;++i) {
                map[i] = '^';
        }
        player.stats.hp = 100;
        player.stats.attack = 51.0f;
        player.stats.defense = 5.0f;
        player.stats.speed = 6.0f;
        player.pos.x = 12;
        player.pos.y = 12;
```

```
map[player.pos.x+MAP_WIDTH*player.pos.y]='Y';
player.name = "Hero";
while (1) {
	if (player.stats.hp <= 0) {
		printf("You have died!\n    GAME OVER!\n\n");
		printf("You entered %d battles.\n",score);
		return;
	}
	printf("Position: (%d,%d)\n",player.pos.x,player.pos.y);
	printf("HP: %f\n",player.stats.hp);
	for (i=0;i<MAP_SIZE;++i) {
		if (i%MAP_WIDTH == 0) {
			printf("\n");
		}
		printf("%c",map[i]);
	}
	printf("\n\nPress r for Right, l for Left, u for Up, or d for down.\n");
	printf("or... Press q to quit\n");
	scanf("%c",&choice);
	getchar();
	switch(choice) {
		case 'r':
		case 'R':
			player.pos.x+=1;
			if (player.pos.x >= MAP_WIDTH) {
```

```
                player.pos.x=MAP_WIDTH-1;
                break;
        }
        map[player.pos.x-1+MAP_WIDTH*player.pos.y]='^';
        map[player.pos.x+MAP_WIDTH*player.pos.y]='Y';
        chanceEncounter(&player);
        break;
case 'l':
case 'L':
        player.pos.x-=1;
        if (player.pos.x < 0) {
                player.pos.x=0;
                break;
        }
        map[player.pos.x+1+MAP_WIDTH*player.pos.y]='^';
        map[player.pos.x+MAP_WIDTH*player.pos.y]='Y';
        chanceEncounter(&player);
        break;
case 'u':
case 'U':
        player.pos.y-=1;
        if (player.pos.y < 0) {
                player.pos.y=0;
                break;
        }
```

```
                        map[player.pos.x+MAP_WIDTH*(player.pos.y+1)]='^';
                        map[player.pos.x+MAP_WIDTH*player.pos.y]='Y';
                        chanceEncounter(&player);
                        break;
                case 'd':
                case 'D':
                        player.pos.y+=1;
                        if (player.pos.y >= MAP_WIDTH) {
                                player.pos.y=MAP_WIDTH-1;
                                break;
                        }
                        map[player.pos.x+MAP_WIDTH*(player.pos.y-1)]='^';
                        map[player.pos.x+MAP_WIDTH*player.pos.y]='Y';
                        chanceEncounter(&player);
                        break;
                case 'q':
                case 'Q':
                        printf("You entered %d battles.\n",score);
                        return 0;
        }
    }
}
```

Makefile

```
all:
    gcc -g -o game main.s battlerInit.s mod.s battle.s attack.s map.s
```

```
	gcc -g -o cgame main.c

clean:

	rm cgame game
```

run.sh

```
read -p "running c version... Press enter to continue"

./cgame

read -p "running asm version... Press enter to continue"

./game
```