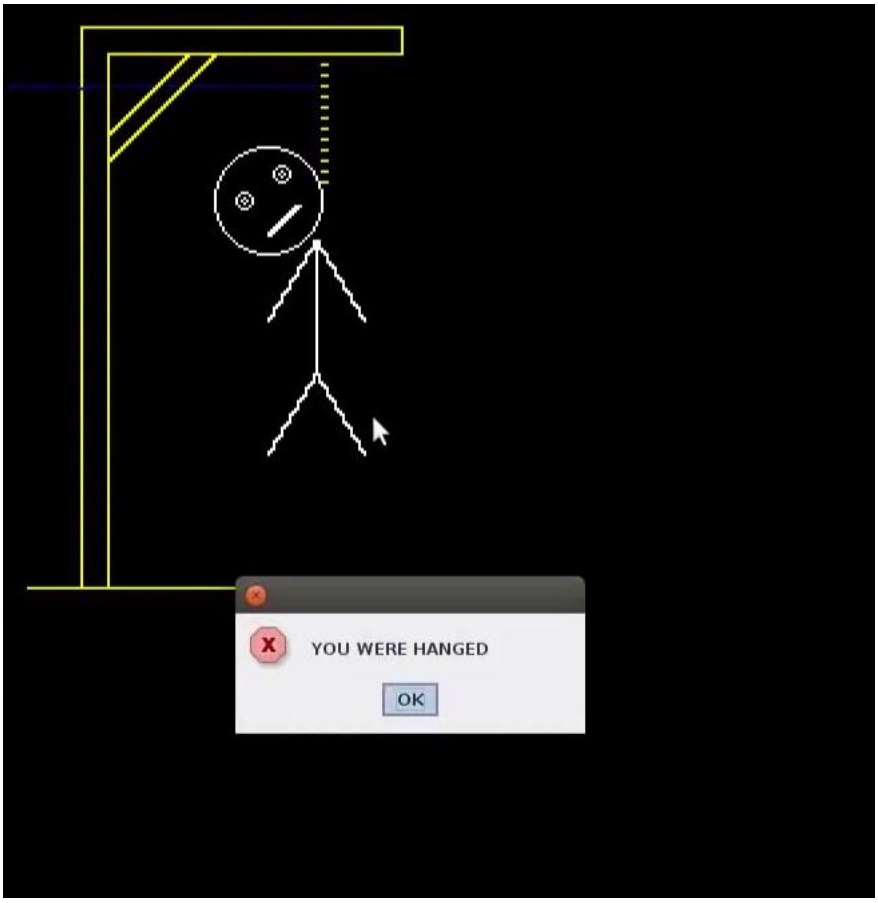

HANGMAN

Implemented in MIPS Assembly

The project implements the Hangman Game in MIPS Assembly. Bitmap display tool is used to display Hangman in MARS

Submitted By:
Akash Chaturvedi
Grant Yetter
Fabian Soriano
Kushal Khanal

Table of Contents

Task Done

We were a team of 4: Akash Chaturvedi, Grant Yetter, Fabian Soriano, Kushal Khanal.
Most of my work was on display part. i.e. Hangman display using Bitmap Display tool in MARS.
I used global segment of memory to draw pixels on particular co-ordinates specified by the arguments in the registers.

```
##    Function that will colour in a pixel from the X and Y coor and the
color asked
      setPixel:
            li    $t3, 0x10000000        #loading first pixel into a temp
registar, using the data segament
            #li   $t3, 0x10010000
            sll   $t0, $a1, 9      #Multiply the Y coord. by 512 (means
$a1 has Y coordinate)
            addu  $t1, $t0, $a0          #Add X and Y together  (means $a0
has X coordinate)
            sll   $t1, $t1, 2      #Multiply the X+Y by 4!
            #srl  $t1, $t1, 2
            addu  $t2, $t3, $t1          #Add to the first pixel
            sw    $a2, ($t2)       #Display Pixel
            jr    $ra              # Jump back to main()
            nop
```

The Draw Line function:

DrawLine function, will draw a line on two points.

```
      drawLine:

            addiu $sp, $sp, -24          # save all the s values
            sw    $s0, 20($sp)
            sw    $s1, 16($sp)
            sw    $s2, 12($sp)
            sw    $s3, 8($sp)
            sw    $s4, 4($sp)
            sw    $ra, ($sp)
```

```
            subu  $s0, $a2, $a0          #Subtract X1 - x0 and storing it
            in dx
            abs   $s0, $s0         #ABS dx
            subu  $s1, $a3, $a1          #subtract y1 - y0
            abs   $s1, $s1         #storing dy
            sub   $s4, $s0, $s1          #making err
            blt   $a0, $a2, settingSX    #check for SX
            li    $s2, -1                #not greater, my SX  -1
            j     checkSY                #check SY
```

The DrawCircle function

```
#     Function that will draw a circle, taking in x, y, radius and a
colour
      drawCircle:
            addiu $sp, $sp, -4
            sw    $ra, ($sp)

            li    $t0, 1                 #storing 1 so it can be used

            sub   $s0, $t0, $a3          #setting F

            li    $s1, 1                 #setting ddF_X
            mul   $s2, $a3, -2           #setting ddF_y

            li    $s3, 0                 #setting X (not x0)
            move  $s4, $a3         #setting Y (not y0)

            #     setPixel(x0, y0 + radius);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9         # move the colour into a2 ready to set
            the pixel
            add   $a1, $a1, $a3          # y0 + radius
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0, y0 - radius);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
```

```
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            sub   $a1, $a1, $a3           # y0 - radius
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 + radius, y0);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            add   $a0, $a0, $a3           # x0 + radius
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16
#
            #     setPixel(x0 - radius, y0);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            sub   $a0, $a0, $a3           # x0 - radius
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            j     circleLoop        # jump to the circle loop
            nop

      circleLoop:
            blt   $s3, $s4, keepGoingCircle    # if x < y then keep going
...
```

```
            nop

            lw    $ra, ($sp)              # if NOT then load ra and go back
to main
            addiu $sp, $sp, 4

            jr    $ra
            nop

      keepGoingCircle:
            bgez  $s0, circleFLoop        # if f >= 0)
            nop

            j     circleMainLoop                # if not, carry onto main
drawing loop
            nop

      circleFLoop:                              # if statement
            sub   $s4, $s4, 1             # y--
            add   $s2, $s2, 2             # ddf_y +=2
            add   $s0, $s0, $s2                 # f+= ddf_y

            j     circleMainLoop                # go to main drawing loop
            nop

      circleMainLoop:
            add   $s3, $s3, 1       #x++;
            add   $s1, $s1, 2       #ddF_x += 2;
            add   $s0, $s0, $s1           #f += ddF_x;

            #     setPixel(x0 + x, y0 + y);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            add   $a0, $a0, $s3           # x0 + x
            add   $a1, $a1, $s4           # y0 + y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 - x, y0 + y);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
```

```
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            sub   $a0, $a0, $s3           # x0 - x
            add   $a1, $a1, $s4           # y0 + y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 + x, y0 - y);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            add   $a0, $a0, $s3           # x0 + x
            sub   $a1, $a1, $s4           # y0 - y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 - x, y0 - y);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            sub   $a0, $a0, $s3           # x0 - x
            sub   $a1, $a1, $s4           # y0 - y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 + y, y0 + x);
            addiu $sp, $sp, -16
```

```
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            add   $a0, $a0, $s4           # x0 + x
            add   $a1, $a1, $s3           # y0 + y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 - y, y0 + x);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            sub   $a0, $a0, $s4           # x0 - x
            add   $a1, $a1, $s3           # y0 + y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            #     setPixel(x0 + y, y0 - x);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            add   $a0, $a0, $s4           # x0 + x
            sub   $a1, $a1, $s3           # y0 - y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16
```

```
            #     setPixel(x0 - y, y0 - x);
            addiu $sp, $sp, -16
            sw    $a0, 12($sp)
            sw    $a1, 8($sp)
            sw    $a2, 4($sp)
            sw    $a3, ($sp)
            move  $a2, $t9          # move the colour into a2 ready to set
the pixel
            sub   $a0, $a0, $s4           # x0 - x
            sub   $a1, $a1, $s3           # y0 - y
            jal   setPixel
            nop
            nop
            lw    $a3, ($sp)
            lw    $a2, 4($sp)
            lw    $a1, 8($sp)
            lw    $a0, 12($sp)
            addiu $sp, $sp, 16

            j     circleLoop
```

These two are the basic functions that I used to draw various shapes in Bitmap Display tool. The following are the abstract functions / subroutines that I used to display shapes.

```
drawWalls:
            li    $t9, 0x00FFFF00         #yellow
#pillar1
            li    $a0, 30
            li    $a1, 10
            li    $a2, 30
            li    $a3, 220
            jal   drawLine
            nop
            nop
#pillar2
            li    $a0, 40
            li    $a1, 20
            li    $a2, 40
            li    $a3, 220
            jal   drawLine
            nop
            nop
#knob 1
            li    $a0, 30
            li    $a1, 10
            li    $a2, 150
            li    $a3, 10
            jal   drawLine
            nop
            nop
#knob 2
```

```
            li    $a0, 40
            li    $a1, 20
            li    $a2, 150
            li    $a3, 20
            jal   drawLine
            nop
            nop
#knob corner
            li    $a0, 150
            li    $a1, 10
            li    $a2, 150
            li    $a3, 20
            jal   drawLine
            nop
            nop

            li    $a0, 40
            li    $a1, 60
            li    $a2, 80
            li    $a3, 20
            jal   drawLine

            li    $a0, 40
            li    $a1, 50
            li    $a2, 70
            li    $a3, 20
            jal   drawLine

#base1
            li    $a0, 10
            li    $a1, 220
            li    $a2, 200
            li    $a3, 220
            jal   drawLine
            nop
            nop

#base2
            li    $a0, 150
            li    $a1, 10
            li    $a2, 150
            li    $a3, 20
            jal   drawLine
            nop
            nop

j hangmanExit
############################################################STEP 1 ENDS

######################## STEP 2 starts ###################################

#rope
drawRope:
```

```
            li    $a0, 120
            li    $a1, 20
            li    $a2, 120
            li    $a3, 70
            jal   dashLine
            nop
            nop
            li    $a0, 121
            li    $a1, 20
            li    $a2, 121
            li    $a3, 70
            jal   dashLine
            li    $a0, 122
            li    $a1, 20
            li    $a2, 122
            li    $a3, 70
            jal   dashLine
j hangmanExit
############################################################STEP 2 ENDS

drawFace:
######################## STEP 3 starts ###################################

#     # hangman face
            li    $t9, 0x00FFFFFF
            #li   $t9, 0x00FFFF00
            li    $a0, 100
            li    $a1, 75
            li    $a3, 20                 #radius
            jal   drawCircle
            nop
            nop
      #lefteye
            li    $a0, 91
            li    $a1, 75
            li    $a3, 1                  #radius
            jal   drawCircle
            nop
            nop
      #righteye
            li    $a0, 105
            li    $a1, 65
            li    $a3, 1                  #radius
            jal   drawCircle
            nop
            nop
      #nose
      #     li    $a0, 101
      #     li    $a1, 76
      #     li    $a2, 105
      #     li    $a3, 80
      #     jal   drawLine
      #     nop
      #     nop
```

```
      #mouth
            li    $a0, 100
            li    $a1, 87
            li    $a2, 111
            li    $a3, 77
            jal   drawLine

j hangmanExit
#############################################################STEP 3 ENDS

######################## STEP 4 starts ###################################
drawBody:
      #hangman body
            li $a0, 118
            li $a1, 90
            li $a2, 118
            li $a3, 140
            jal drawLine
j hangmanExit
#############################################################STEP 4 ENDS

######################## STEP 5 starts ###################################

drawLeftHand:
      #hangman left hand
            li $a0, 118
            li $a1, 90
            li $a2, 100
            li $a3, 120
            jal drawLine
j hangmanExit
#############################################################STEP 5 ENDS

######################## STEP 6 starts ###################################
drawRightHand:
      #hangman right hand
            li $a0, 118
            li $a1, 90
            li $a2, 136
            li $a3, 120
            jal drawLine
j hangmanExit
#############################################################STEP 6 ENDS

######################## STEP 7 starts ###################################
drawLeftLeg:
      #hangman left leg
            li $a0, 118
            li $a1, 140
            li $a2, 100
            li $a3, 170
            jal drawLine
j hangmanExit
#############################################################STEP 7 ENDS
```

```
######################## STEP 8 starts ################################
      #hangman right leg
drawRightLeg:
            li $a0, 118
            li $a1, 140
            li $a2, 136
            li $a3, 170
            jal drawLine

j hangmanExit
#################################################################STEP 8 ENDS

######################## Hangman gonnna die
###################################
      #HANGMAN Dies
hangmanDies:

      ################################################
            li $v0, 33
      li $a0, 60  # pitch, C#
      li $a1, 2000      #duration in milisecond
      li $a2, 111 #instrument (0 - 7 piano)
      li $a3, 100 #volume
      syscall
      ################################################

            #leftbigeye
            #li   $t9, 0x00FF00FF        # Colour - Blue
            li    $a0, 91
            li    $a1, 75
            li    $a3, 3                #radius
            jal   drawCircle
            nop
            nop
      #rightbigeye
            li    $a0, 105
            li    $a1, 65
            li    $a3, 3                #radius
            jal   drawCircle
            nop
            nop
      #toung comesout
            li    $a0, 100
            li    $a1, 88
            li    $a2, 112
            li    $a3, 77
            jal   drawLine
#########################################################################
##
      hangmanExit:
      lw $ra, 0($sp)
      addi $sp, $sp, 4
      jr $ra
```

Additionally, I also worked on merging various code sections i.e Fabian's work of sound syscalls, Kushal's work of File reading and grant's work of String manipulation into my code.

the WHAT (Project Description)

The project required us to build a functional Hangman Game using MIPS Assembly Language. This project must implement sound and generate a random word. In our case a random word was generated using a file hardcoded with words. The program must ask the user to enter characters, in the event were a mismatch happens the hang man begins to display on the bitmap display and course one body part at a time. In the event where the user enters a correct or incorrect character a sound must generate letting the user know whether the character was correct or incorrect. If the user is not able to correctly guess the word, the hangman will draw all body parts and be hanged. In the event were the user is able to guess the word correctly, he wins the game and a final sound will alert the player that he was saved.

The HOW (How the program was implemented)

Our game uses a dictionary file that contains name of the 50 states. Every time the game is run, dictionary file is read and one state name is seleted randomly. This selected random state name is used in the game and the user has to guess the game.
The following syscalls were used for this implementation:

1)File open

```
        #opening the file
        li      $v0, 13              #13 for opening the file
        la      $a0, fin      #fin is string that contains file name
        li      $a1, 0        #0 is read modde
        li      $a2, 0
        syscall
This file open syscall returns a file descriptor into register $v0
```

2)File read

```
File read uses the file descriptor to read the file.
        move    $s6, $v0        #save the file descriptor

        #now read the file just opened and store all of its content into
buffer
        li      $v0, 14              #syscall for reading a file
        move    $a0, $s6      #pass the file descriptor in $a0
```

```
        la      $a1, buffer   #buffer is the space allocated in data
segment
        li      $a2, 1024     #maximum length of characters reading from
file
        syscall
```
This syscall returns the number of characters read into $v0 and save all the characters read into buffer

3) Random Number Generator

We use random generator syscall to generate a random number between 0 to the length of buffer.
We make use of the generated number to read from buffer at random location.

```
        #$s7 contains the length of buffer - 10, Hence this function
generates a random number used in selecting a state form buffer string
         move $a1, $s7   #Range set from 0 to (length of buffer - 10)
         li $v0, 42      #generates random number and put it in $a0
         syscall
```

4) Read a state name from buffer randomly

```
la $t0, buffer
add $t0, $t0, $a0            #add generated random number, currently $a0
contains the random number between 0 to 44

recur:
        lb $t2, 0($t0)          #Load the first byte from address in $t2
        beq $t2, 0x2a, pointS   #if it encounters a * while reading
characters, jump to storeWord function
        addi $t0, $t0, 1        #else increment the address unless it
encounters a *
        j recur


         ###################################
         pointS:
        addi $t0, $t0, 1
         jal storeWord           #storeWord function is used for storing a
random selecter word from buffer into testWord
        ####################################################
```

5) Store the read word into testWord string (which is used further in the program)

```
storeWord:
         la $t1, testWord
         whileSW:
         lb $t2, 0($t0)
         beq $t2, 0x2a, exitStoreWord    #exit if reaches to the end of
current word (another *)
         beqz $t2, exitStoreWord         #exit if reached end of buffer
string
         sb $t2, 0($t1)
```

```
        addi $t0, $t0, 1
        addi $t1, $t1, 1
        j whileSW

        exitStoreWord:
        li $t3, 0x00
        sb $t3, 0($t1)       #storing null at the end of testWord string
inorder to make it a valid string
        jr $ra
```

By the end of this function, testWord contains a state name of US selected randomly.

6) The strlen subroutine:

```
###### THIS FUNCTION CALCULATES THE STRING LENGTH OF TESTWORD AND RETURNS
IT IN $v0
## Argument passed $a3, contains address of the string whose length is to
calculate
## Return value $v0, length of string
strlen:
        li $v0, 0
        loopStrLen:
        lb $t2, 0($a3)  # Load the first byte from address in $t0
        beqz $t2, endStrlen   # if $t2 == 0 then go to label end
        addi $a3, $a3, 1      # else increment the address
        addi $v0, $v0, 1 # and increment the counter of course
        j loopStrLen      # finally loop

        endStrlen:    jr $ra
```

7) Initializing guessedString

Throughout the program, we maintain a guessedString that is double in length of testWord string. The reason behind that is because it has an space between each character in the testWord string.
for e.g. if testWord = "texas" (length of testWord is 5)
initial guessedString will be "_ _ _ _ _ " (length of guessedString is 10)

We wrote the following code for the implementation of this part:

```
        ##calling strlen function
        la $a3, testWord        #$a3 is passed argument in strlen function
        jal strlen      #it returned testWord string length in $v0


 #loop for initializing guessedString with '_' and ' ' alternatively
        la $t0, guessedString
        li $t1, 0
        li $t2, 0x5f         #initialize $t2 with ascii value of a '_'
(underscore)
        li $t3, 0x20         #initialize $t3 with ascii value of a ' '
(space)
        move $t4, $v0   #saving the returned length of string into $t4
```

```
        whileA:
                beq $t1, $t4, exitA     #$t1 is counter that increments by
1 everytime loop runs, run the loop equal to length of testWord string
                sb $t2, 0($t0)          #store '_ ' in gussedString
                addi $t0, $t0, 1        #increment address pointer by 1
                sb $t3, 0($t0)          #store a ' ' i.e. an space in
gussedString
                addi $t0, $t0, 1        #increment address pointer by 1
                addi $t1, $t1, 1        #increment the counter used for
iterating the loop
                j whileA

                exitA:
                li $t1, 0x0
                sb $t1, 0($t0)
```

8) promptChar subroutine

```
###### THIS FUNCTION USED TO TAKE INPUR FROM THE USER
############################
## return value, read character in $v0

promptChar:
        la $t3, charInputHistory
        move $a3, $t3                   #going to call strlen, so storing
address of string in $a3(argument for strlen function)
        addi $sp, $sp, -4               #storing current $ra (i.e some
line # of main), on stack
        sw $ra, 0($sp)                  #because going to call strlen,
that will overwrite $ra

        jal strlen                      #this will return the length of
charInputHistory string into $v0
        move $t0, $v0                   #$t0 has length of
charInputHistory

        promptCharLoop:                 #if the char has already been
input (present in history) then come here again
                li $v0, 54
                la $a0, charInputPrompt
                la $a1, charInput
                li $a2, 2
                syscall

                la $t1, charInput
                lb $t2, 0($t1)          #now $t2 has the user input
character
                #there is nothing in charInputHistory array
                beqz $t0, checkHistoryLoopExit

                #check whether $t2 has previously been inputed by running
a loop on charInputHistory, $t0 times
                checkHistoryLoop:
                        lb $t4, 0($t3)
```

```
                        beqz $t4, checkHistoryLoopExit  #have reached end
of 'charInputHistory' string
                        beq $t4, $t2, promptCharLoop    #means user has
previously inputed the same char and hence need to input again
                        addi $t3, $t3, 1
                        j checkHistoryLoop


        checkHistoryLoopExit:
#       addi $t3, $t3, 1        #store the new input in charInputHistory
char array
        sb $t2, 0($t3)

        lw $ra, 0($sp)                  #restoring $ra from stack to get
back to somewhere in main
        addi $sp, $sp, 4
        move $v0, $t2                   #returns the read char in $v0
        jr $ra
```

9)matchSound subroutine

This subroutine is called from our main loop when there is a match

```
matchSound:
li $v0, 33     #for midi sound output
li $a0, 60     # pitch, C#
li $a1, 80     #duration in milisecond
li $a2, 124    #instrument (0 - 7 piano)
li $a3, 60     #volume
syscall
jr $ra
```

10)drawHangman subroutine:

This subroutine is also called from our main loop everytime there is a mismatch. We call the hangman subroutine with an error number passed in $a0, so that we know which part of hangman to draw based on the error number passed.

```
drawHangman:
        addi $sp, $sp, -4
        sw $ra, 0($sp)
        move $s2, $a0         #error number is saved in ragister $s2 now

#we first play a mismatch sound in case of mismatch, we could have made a
separate function for that. But this is also fine
        ################################################
        ############### MISMATCH SOUND #################
        li $v0, 33
        li $a0, 60     # pitch, C#
        li $a1, 100    #duration in milisecond
        li $a2, 111    #instrument (0 - 7 piano)
        li $a3, 100    #volume
        syscall
        ################################################
```

```
        beq $s2, 0, hangmanExit
        beq $s2, 1, drawWalls
        beq $s2, 2, drawRope
        beq $s2, 3, drawFace
        beq $s2, 4, drawBody
        beq $s2, 5, drawLeftHand
        beq $s2, 6, drawRightHand
        beq $s2, 7, drawLeftLeg
        beq $s2, 8, drawRightLeg
        beq $s2, 9, hangmanDies

                #li     $t9, 0x00FFFFFF             # Colour - White
                #li     $t9, 0x00FF00FF             # Colour -
Blue
Additionally, we use set pixel subroutine in all of the above branch
instructions to set pixels on specified coordinates.
So as to draw a geomatrical figure on bitmap Display tool.
```

11) setChar subroutine:

setChar subroutine is called for putting a matched character in 'guessedWord' string so as to display the current status of guesses.

```
################################################################################
#########
#This function is used to set matched character into 'guessedString'
buffer,
# The character read is passed in $a0
################################################################################
#########
setChar:
        move $t4, $a0
        la $t0, matchPositions
        la $t1, guessedString
        li $t2, 0
        la $a3, testWord      #$a3 is passed argument in strlen function

        addi $sp, $sp, -4
        sw $ra, 0($sp)
        jal strlen     #it returned testWord string length in $v0

        lw $ra, 0($sp)
        addi $sp, $sp, 4
        move $t3, $v0  #saving the returned length of string into $t3

        whileB:
                beq $t2, $t3, exitB   #$t2 is counter that increments by 1
everytime loop runs
                lb $t5, 0($t0)                #if()
                lb $t6, 0($t1)        #read char from guessedString, and
only insert if read char($t6) is '_'
                beqz $t5, dontSet
```

```
                beq $t6, 0x5f, set
                j dontSet

                set:
                sb $t4, 0($t1)

                dontSet:
                addi $t0, $t0, 1      #increment address pointer by 1
                addi $t1, $t1, 2
                #sb $t3, 0($t0)              #store a space in gussedString
                #addi $t0, $t0, 1     #increment address pointer by 1
                addi $t2, $t2, 1      #increment the counter used for
iterating the loop
                j whileB

        exitB:

        jr $ra
```

12) printGuessedString

This is also called from our main loop to print the current status of the string.

```
printGuessedString:
        li $v0, 55
        li $a1, 1
        la $a0, guessedString
        syscall

        #li $v0, 56
        #la $a0, errorPrompt
        #move $a1, $t7
        #syscall

        jr $ra
```

13) The main loop

We have used a main loop that is the core of string logic in our code. We maintain an errorcount in this loop and keep on calling the promptChar function in the main loop unless we reach to a maximum value of errorCount that we have set as 9 in our program. So we can say that user gets 8 chances of mispredictions, on 9th chance we hang the hangman and terminate the program. Here is the code for this implementation.

```
mainLoop:
                #if (errorcount == MAX) --> exit mainLoop
                beq $s5, 9, exitMainLoop     #$t7 acts as error count

                ############ Counting the number of remaining '_' in
guessedString ###################
                la $t0, guessedString
                li $t1, 0
```

```
                li $s4, 0x5f

                loopx:
                lb $t2, 0($t0)
                beqz $t2, exitLoopx
                addi $t0, $t0, 1
                bne $t2, $s4, loopx
                addi $t1, $t1, 1
                j loopx
                exitLoopx:

                beqz $t1, exitMainLoop

        ##########################################################################
#####################
                jal promptChar
                move $t0, $v0  #$t0 contains the character read

                la $t4, matchPositions
                li $t5, 1
                #so let's just first calculate the length of testWord
                la $t1, testWord
                li $t3, 0      #used in set mark in 'matchPositions'
                #move $a3, $t1
                #jal strlen
                #move $t2, $v0 #now $t2 contains length of testWord

                #this loop will check if this char($t0) is part of correct
answer/string
                li $t6, 0
                shortLoop:
                        lb $t2, 0($t1)        # Load the first byte from
address in $t1(testWord)
                        beqz $t2, endl        #done with checking this input
character, now accept another character from user (i.e. go to mainLoop
again)
                        bne $t2, $t0, nextShortLoop
                        sb $t5, 0($t4)                #if character match
than set mark in 'matchPositions'

                        ############ call matchSound
                        jal matchSound
                        ############################
                        li $t6, 1                     #used as flag to check

                nextShortLoop:
                        addi $t1, $t1, 1      # else increment the address
                        #add $t3, $t3, 1
                        add $t4, $t4, 1               #increment mark for
matchpositions
                        j shortLoop
```

```
            endl:                   #it's done with checking this
particular char and have set 'matchPositions' array also
                    beqz $t6, errCount          # if($t6==0)--> $t7++
                    j noError
                    errCount:
                    addi $s5, $s5, 1      #error count is stored in $s5
                    move $a0, $s5         #passing this error count
argument in drawHAngman subroutine
                    jal drawHangman
                           beq $s5, 9, hangPrompt1


                    noError:
                    move $a0, $t0
                    jal setChar    #now call setChar function for
putting this character in 'guessedWord' at positions specified in
matchPositions

            jal printGuessedString
            j mainLoop

                    hangPrompt1:
                           li $v0, 55
                           la $a0, hangPrompt
                           li $a1, 0
                           syscall
                    ################################################
        li $v0, 59
        la $a0, correctMsg
        la $a1, testWord
        syscall
        ################################################



            exitMainLoop:
            blt $s5, 9, success
            j main_exit

            success:

                    ##play success sound
                    li $v0, 33
                    li $a0, 60     # pitch, C#
                    li $a1, 1000   #duration in milisecond
                    li $a2, 119    #instrument (0 - 7 piano)
                    li $a3, 300    #volume
                    syscall

                    ##display success sound
                    ##display succcess message
                    li $v0, 55
                    la $a0,  successPrompt
                    li $a1, 1
```

```
syscall
```

These 13 parts are the main components in our implementation of Hangman game.

The Process

We decided to do a divide and conquer method collaborating as we worked on separate aspects of the game in order to put together the program. One person would work on sound, display, string logic, or the file library and would send out reports of their progression and explain how they coded each part.

CHALLENGES

We were unable to meet up on a regular basis, but we were able to email one another for help on certain sections in their program. It was also difficult to understand one another's code or logic, but we collaborated and assisted one another to piece together the project.

WHAT I HAVE LEARNT

I have learnt a lot in this project. Not only the technical work that I have learn (MIPS assembly, various sound and File I/O syscalls), But also the way work is done in team project.

It’s not easy to work with a team since everyone has a different process and logic behind their codes, but I was able to adapt and learn how to organize and explain my code and work in a team setting in order to contribute and get the work done.