Christian Honein

CPE233 Final Project
Piano Tiles

I. Introduction

Piano Tiles is a single player game based on the mobile game piano tiles. There are four columns of black piano keys which are moving down, and the main goal of the game is to hit the correct key before the tile moves out of a designated part of the game canvas. As the player, presses the correct keys, a certain song will be played. The length of the keys is proportional on the length of notes in the actual song. Game interaction is through switches, and there is a horizontal line on the screen which shows the user at what horizontal level they are pressing when they raise the switch.

II. User’s manual

Piano Tiles

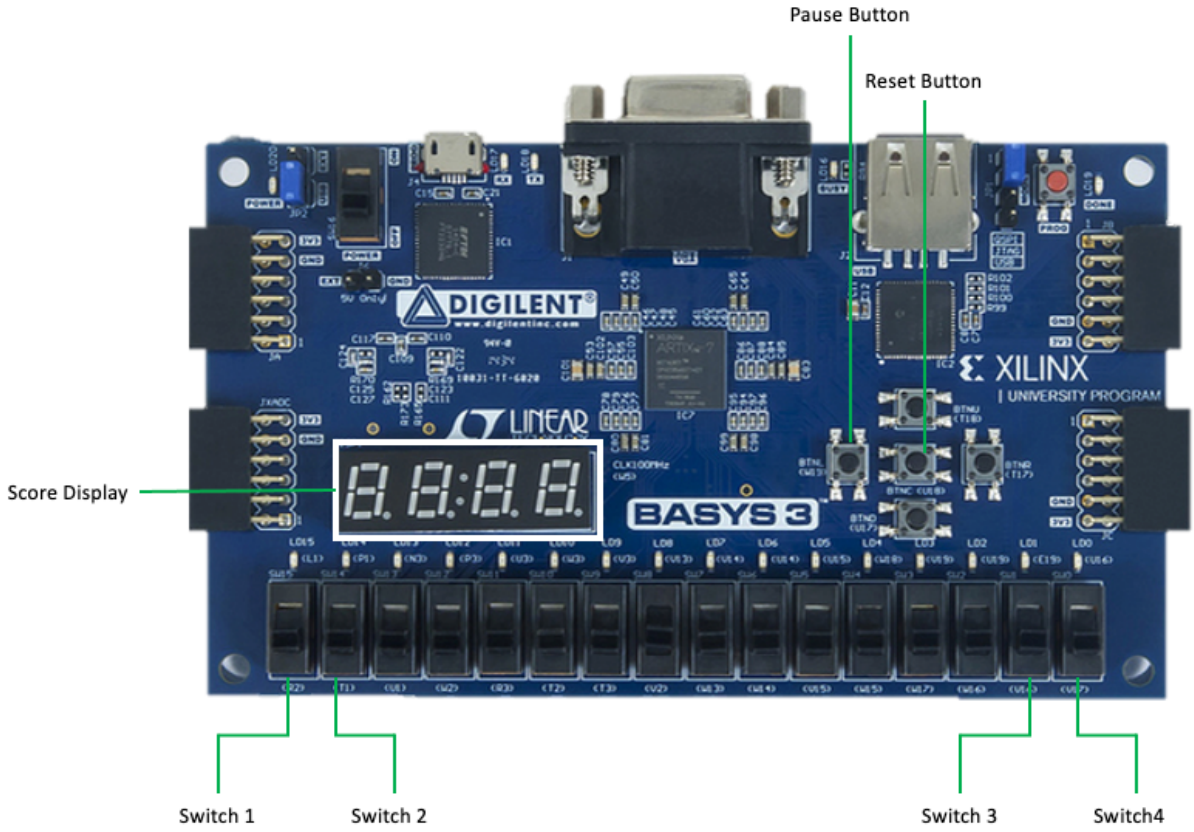
What is Piano Tiles:

Piano Tiles is a music arcade game that tests a player’s reflexes. It is really simply to play, but difficult to master.

The game's aim is to tap the black tiles without making any mistakes. There are 4 columns of black tiles which are moving down the screen and there is a reference red horizontal line on the side of the screen. The player should raise the switch of the corresponding column when a tile in that column becomes on the same level as the reference red line. When the correct switch is raised, a new note of the chosen song will play on the speaker (In the current version, there is only one song: Ode to Joy). Tiles can have different lengths; this is dependent on the length of the notes in the song you are playing. That is why, difficulty is based upon which song is played.

Game Controls:

- **Controller**



- Operation
    - Switch 1: Triggers a tile press on the first column from the left
    - Switch 2: Triggers a tile press on the second column from the left
    - Switch 3: Triggers a tile press on the third column from the left
    - Switch 4: Triggers a tile press on the fourth column from the left
    - Pause Button: pause/resume the game using this button
    - Reset Button: restarts the song from the start and resets score to 0.
- Score Display: continuously shows player score.

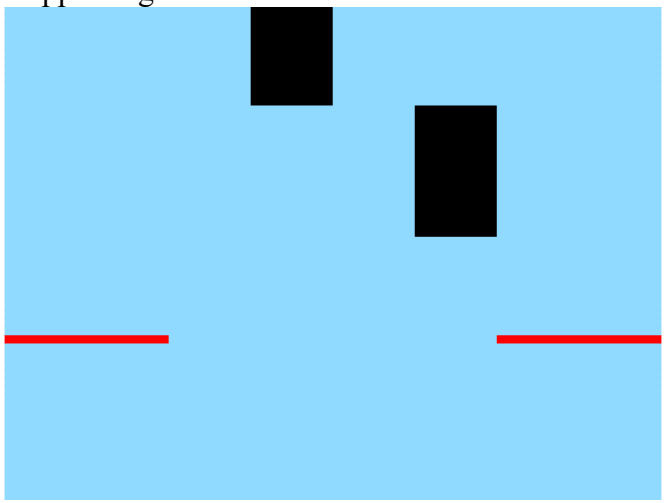
How To Play:

The game is always running and automatically restarts when the player loses the game (This might change in future versions to add the ability to choose songs). The player's score increases by one when the right switch is raised.

Game screens:

1. Game Start, no tiles appear on the screen

2. Tiles appearing on the screen

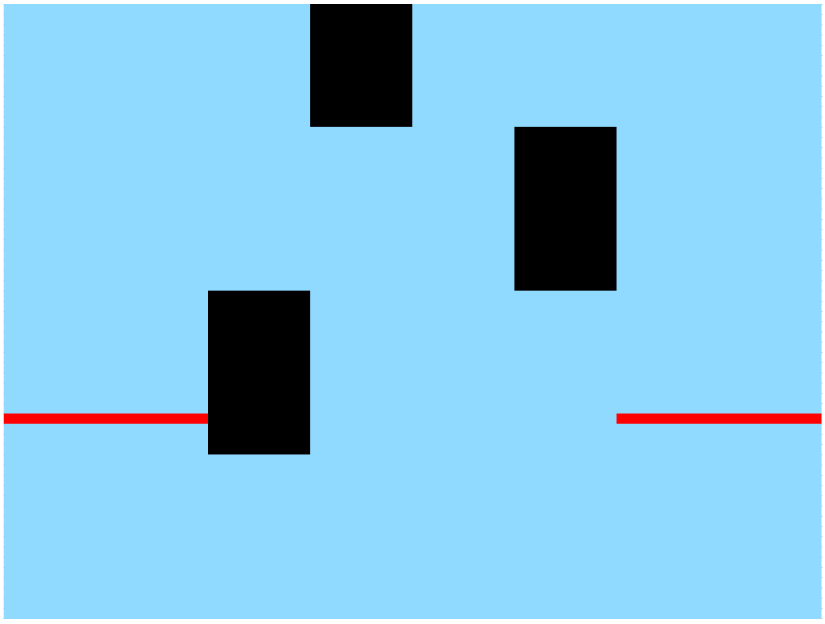


3. Game loss, screen turns red

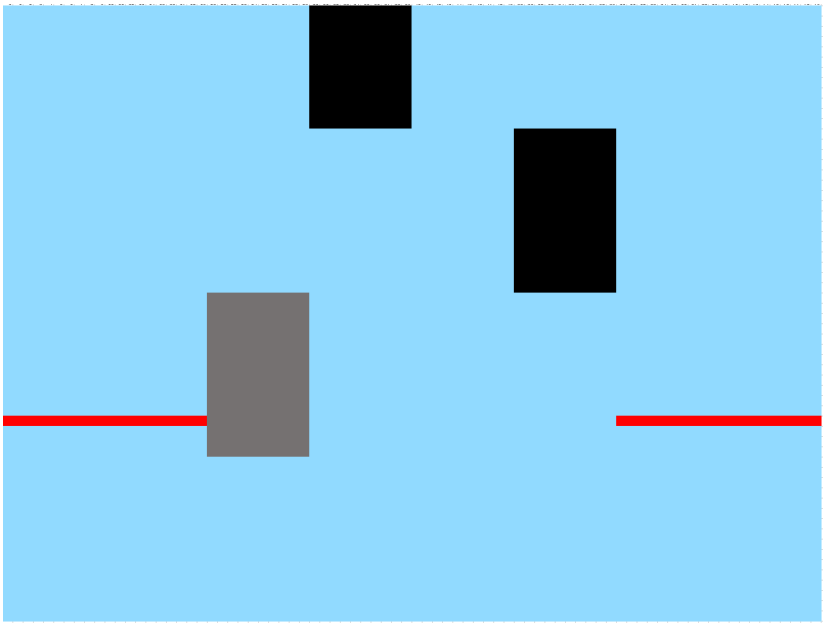
4. Game win, screen turns green. (No preview, only winners can see this screen)

Rules:

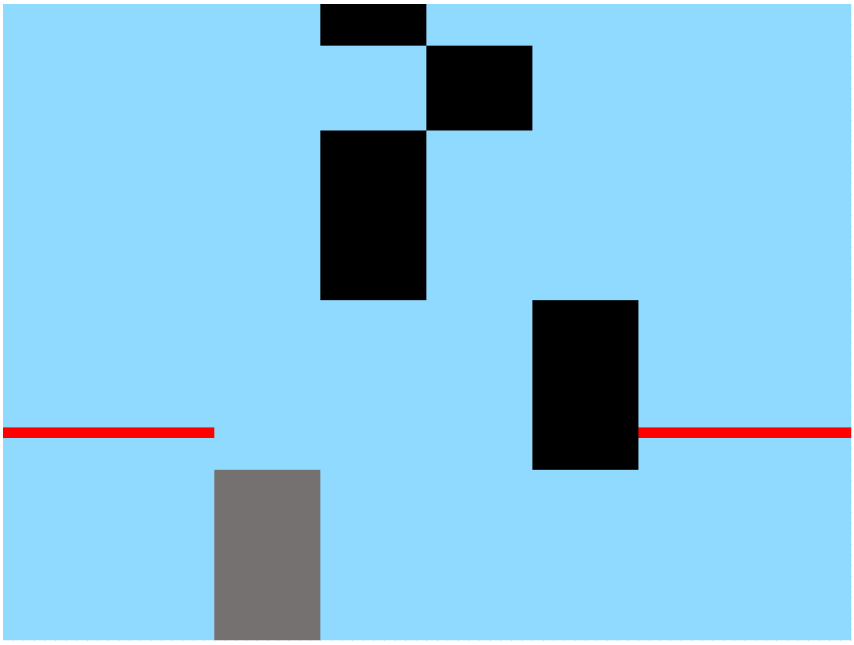
1. Raise the correct switch depending on which column has the black tile that is on the level of the red reference line. In the picture, switch 1 should be raised.



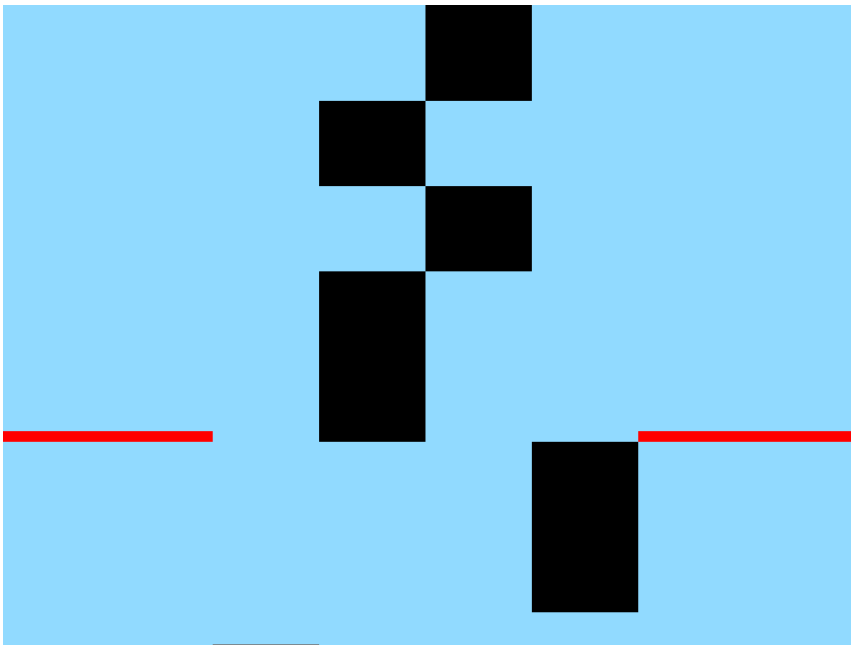
2. When a switch is correctly raised, the key will turn grey. You can keep the switch (in this case switch 1) raised as long as the grey key is on the level of the reference red line.



3. A tile should be grey (pressed) before it goes below the red line.



Game continues since key below red line is grey



Game stops since key below red line is black.

4. Only one switch can be raised at a time. If more than one switch is raised, the game will automatically be lost.

Enjoy the Challenge!

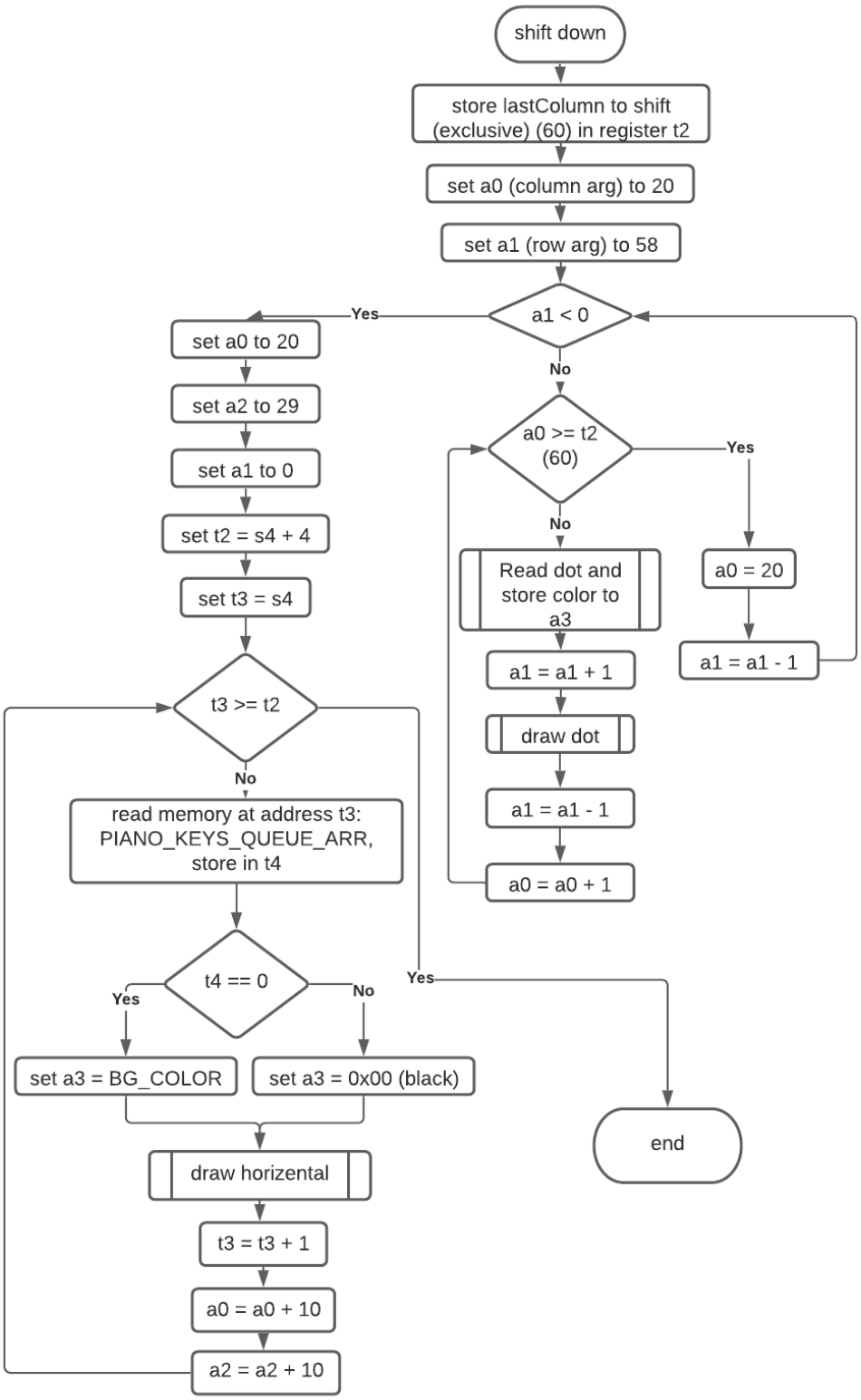
III. Software Design

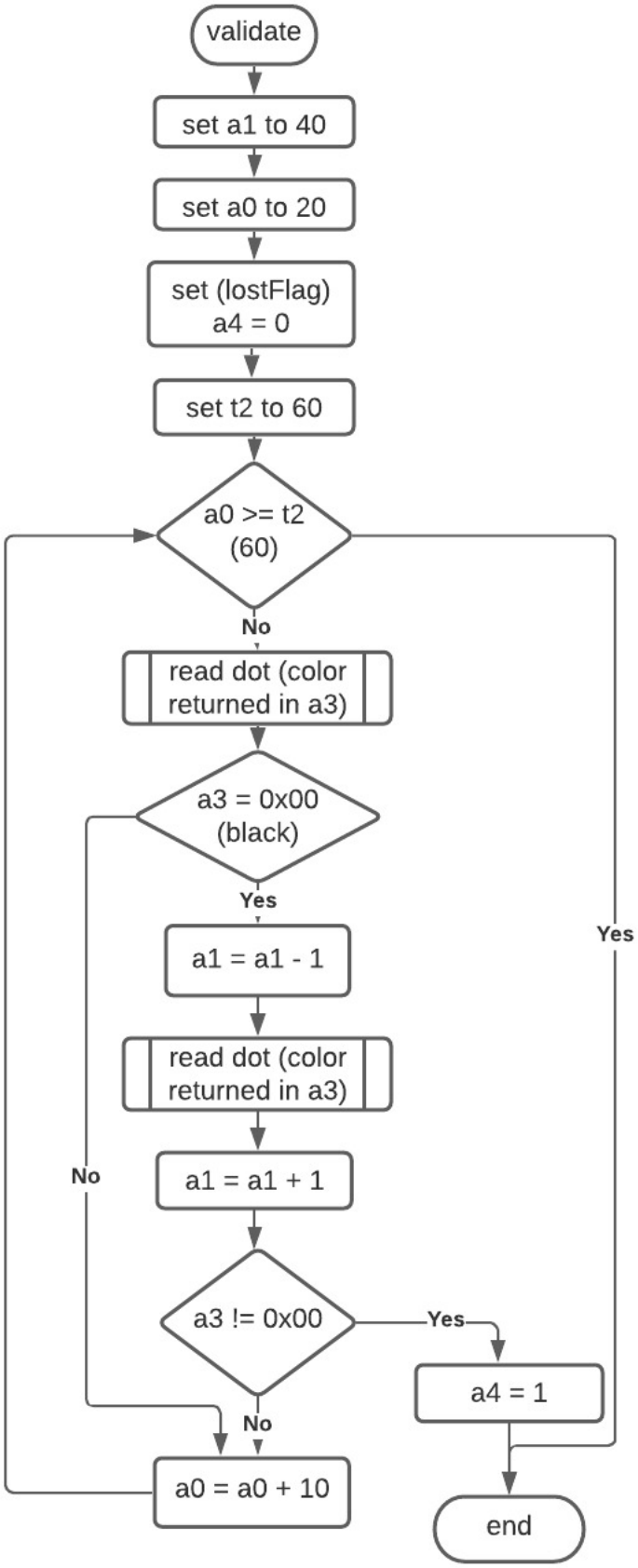
The program has four primary functions to the program:

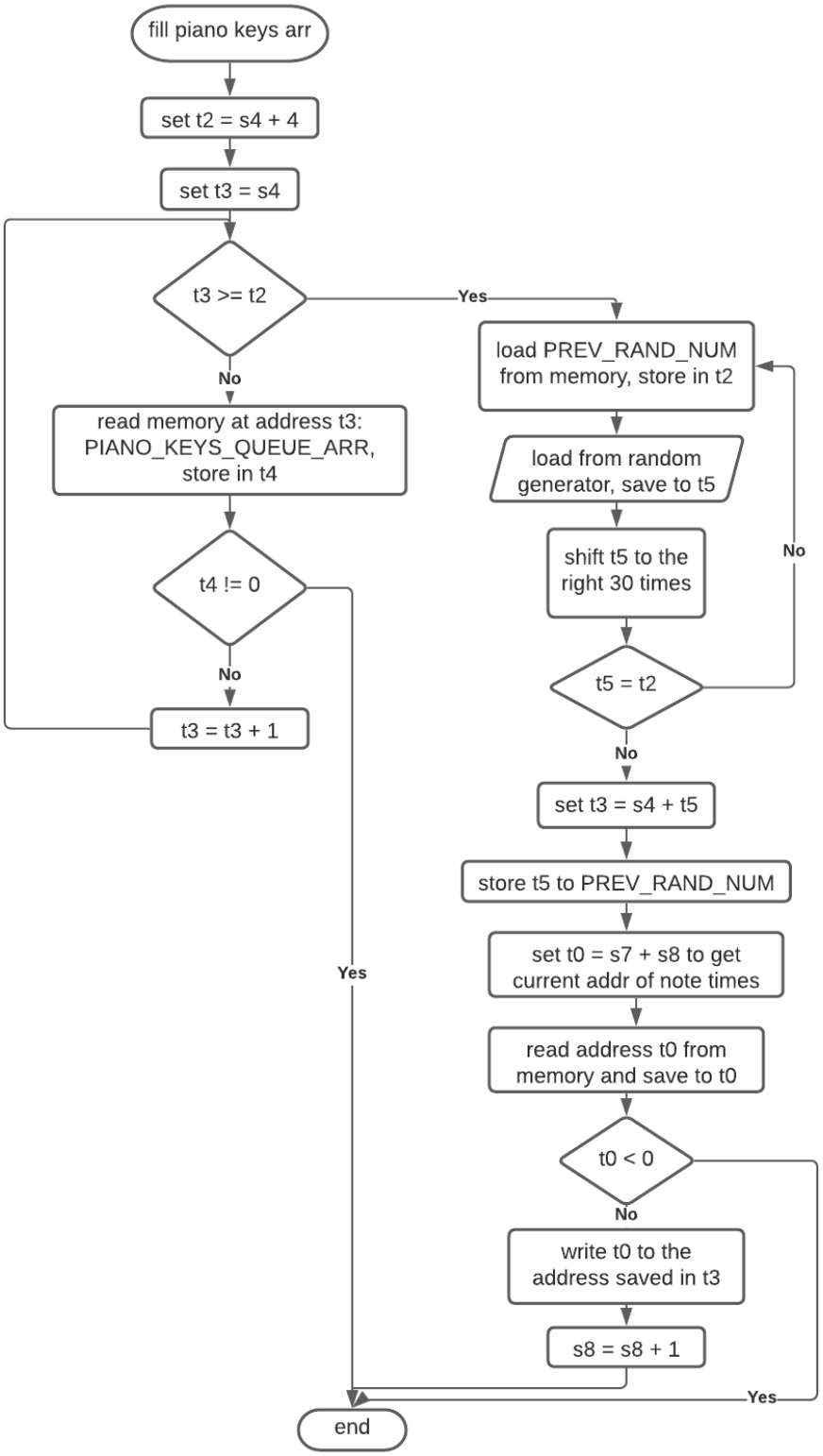
1. shift_down: shifts everything currently being displayed one row down. The top row is filled according to a four-element array in memory named, PIANO_KEYS_QUEUE_ARR.
2. validate: check that all of the tiles that are fully below the reference line are not black. If a black tile is below the reference line, 1 will be returned.
3. Fill_piano_keys_arr: This function fills the array PIANO_KEYS_QUEUE_ARR. This array works as a buffer of the next tile that will be displayed. This function reads from the SONG_TIMES to decide what to put in PIANO_KEYS_QUEUE_ARR. It then uses the random generator to choose at what index to put the data. Moreover, this function has a check to prevent to tiles being placed in the same column consecutively.
4. eval_switches: reads in from the switches and compares what switch is raised with what is currently being display on the screen to identify if the player lost or pressed a new switch. If more than one switch is raised or the wrong switch is raised, 1 will be returned. Function has another return which is 1 when a new tile was pressed. When a new key is pressed, this function will transform this key from black to grey.

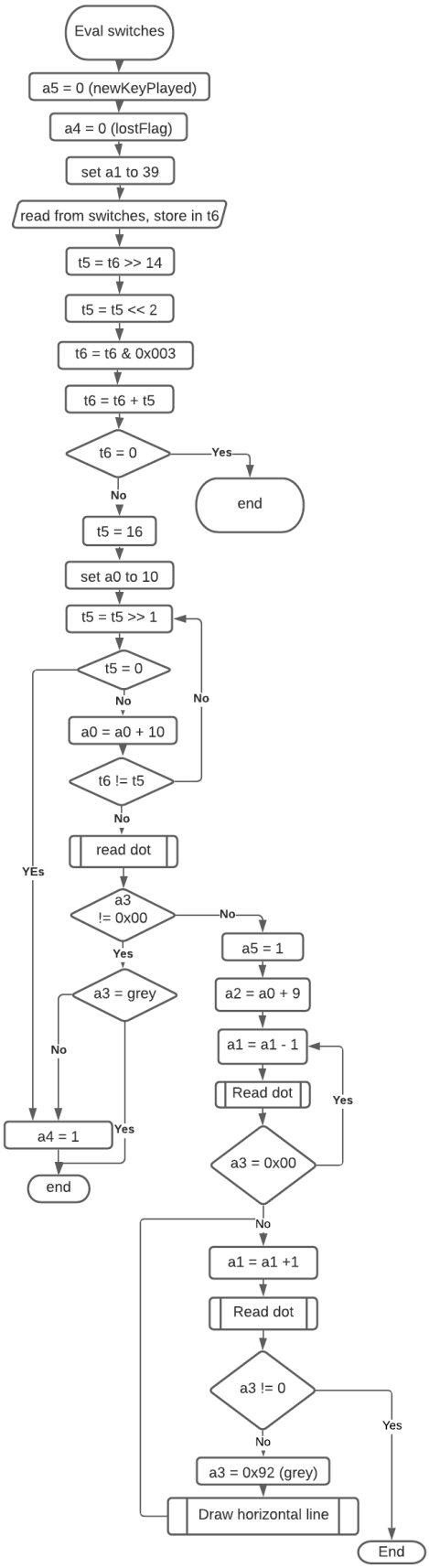
Then, I have the main program which initiates the program, calls the functions above in the correct order, with the desired delay, restarts the game if the player loses, plays the correct note on speaker when correct key is pressed, outputs the score on the 7-segment display (score is converted from HEX to BCD), and pauses the game (in combination with the ISR).

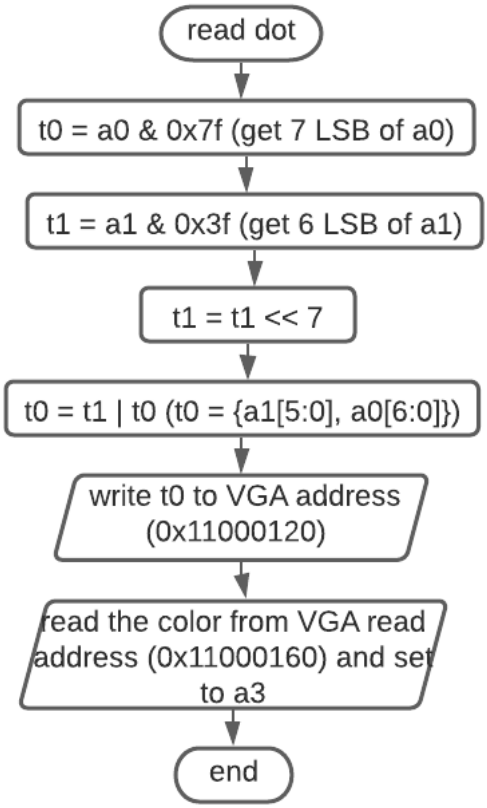
The program has an ISR which is used to pause the game. The ISR communicates with the main program through register s0.

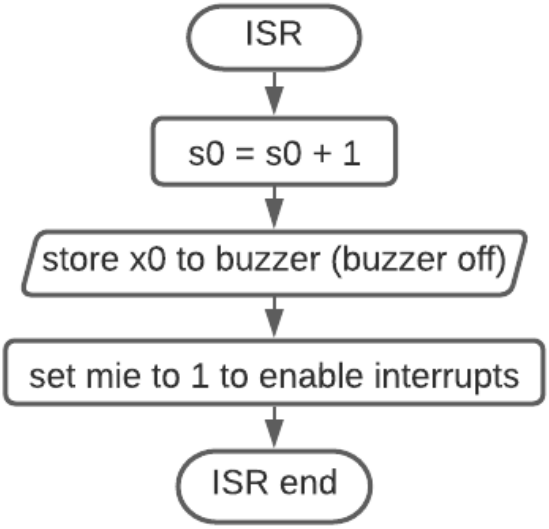

shift down
store lastColumn to shift (exclusive) (60) in register t2
set a0 (column arg) to 20
set a1 (row arg) to 58
a1 < 0
Yes
No
set a0 to 20
set a2 to 29
set a1 to 0
set t2 = s4 + 4
set t3 = s4
t3 >= t2
No
Yes
read memory at address t3: PIANO_KEYS_QUEUE_ARR, store in t4
t4 == 0
Yes
No
set a3 = BG_COLOR
set a3 = 0x00 (black)
draw horizental
t3 = t3 + 1
a0 = a0 + 10
a2 = a2 + 10
a0 >= t2 (60)
Yes
No
Read dot and store color to a3
a1 = a1 + 1
draw dot
a1 = a1 - 1
a0 = a0 + 1
a0 = 20
a1 = a1 - 1
end

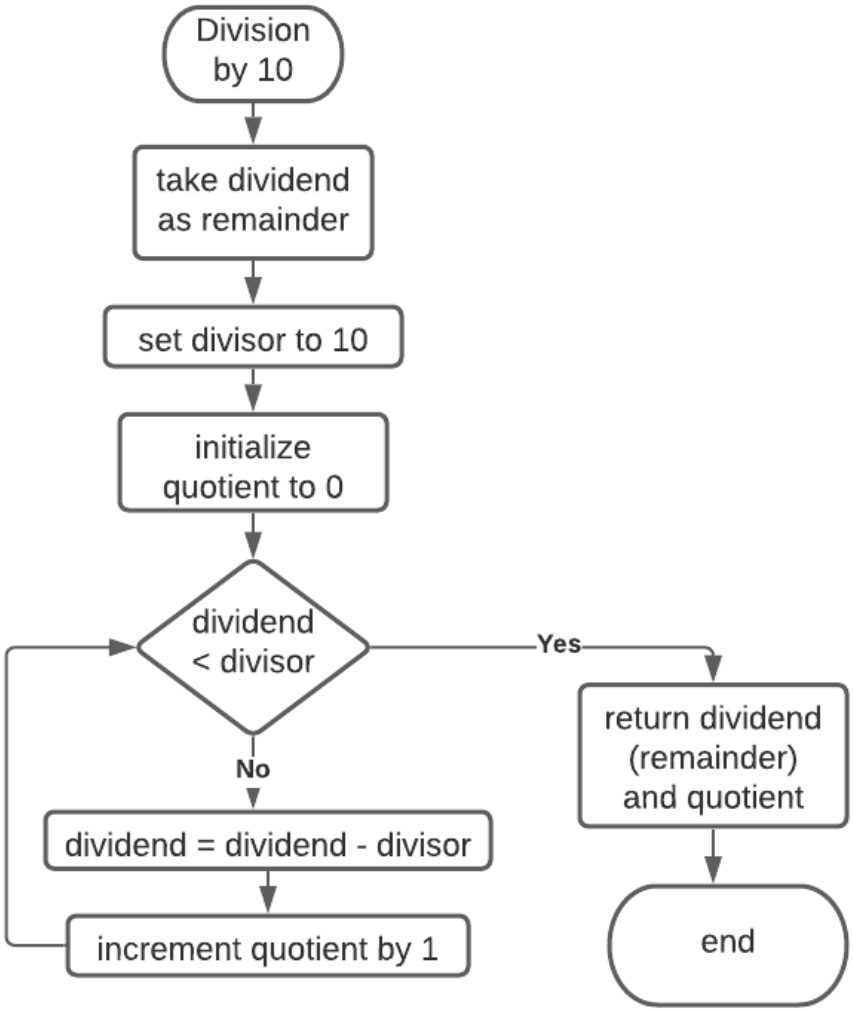

validate
set a1 to 40
set a0 to 20
set (lostFlag)
a4 = 0
set t2 to 60
a0 >= t2
(60)
No
read dot (color
returned in a3)
a3 = 0x00
(black)
Yes
a1 = a1 - 1
read dot (color
returned in a3)
a1 = a1 + 1
No
a3 != 0x00
Yes
a4 = 1
No
a0 = a0 + 10
Yes
end

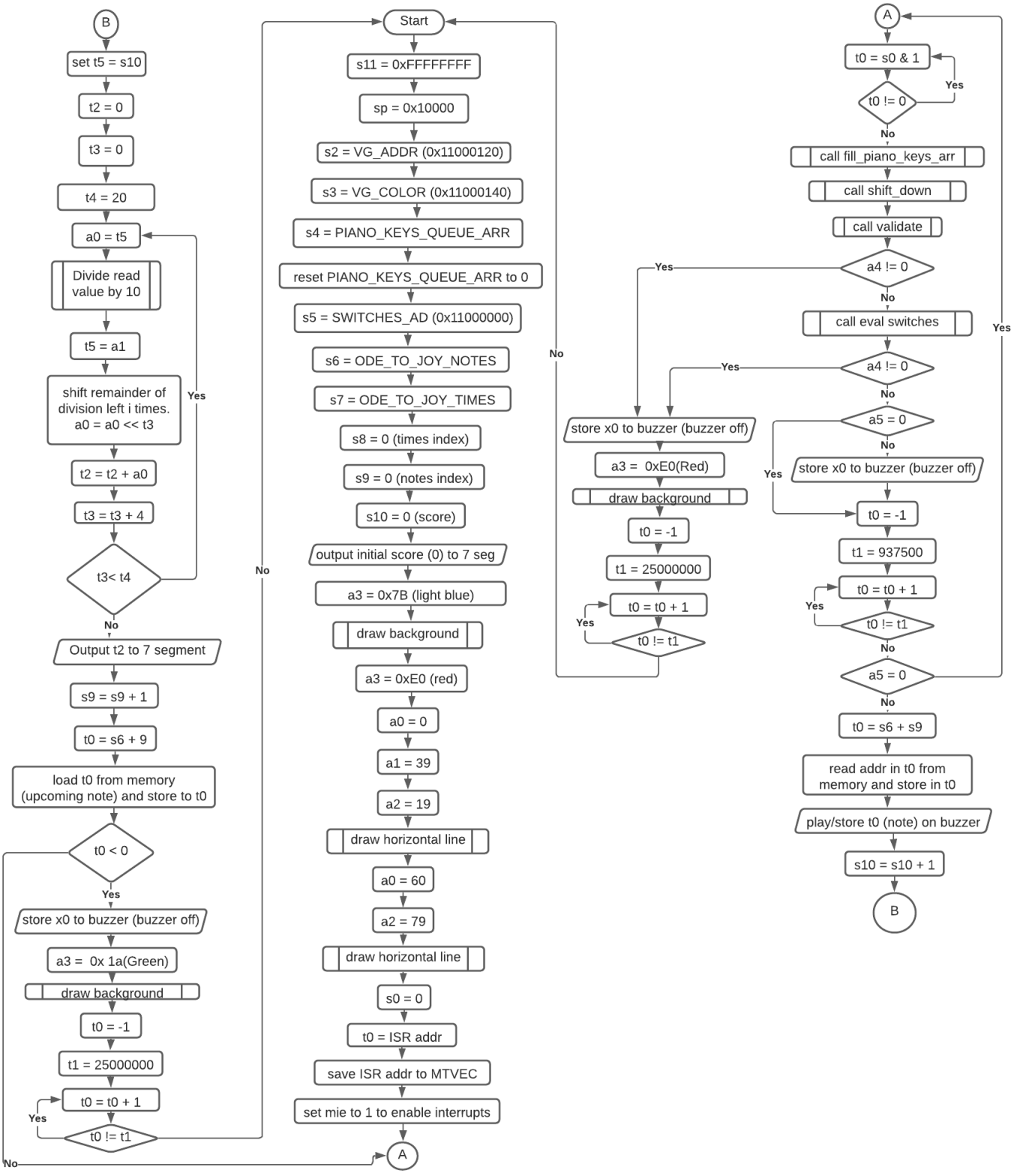

fill piano keys arr
set t2 = s4 + 4
set t3 = s4
t3 >= t2
Yes
No
read memory at address t3: PIANO_KEYS_QUEUE_ARR, store in t4
t4 != 0
No
Yes
t3 = t3 + 1
load PREV_RAND_NUM from memory, store in t2
load from random generator, save to t5
shift t5 to the right 30 times
t5 = t2
No
No
set t3 = s4 + t5
store t5 to PREV_RAND_NUM
set t0 = s7 + s8 to get current addr of note times
read address t0 from memory and save to t0
t0 < 0
No
write t0 to the address saved in t3
s8 = s8 + 1
Yes
end


Eval switches
a5 = 0 (newKeyPlayed)
a4 = 0 (lostFlag)
set a1 to 39
read from switches, store in t6
t5 = t6 >> 14
t5 = t5 << 2
t6 = t6 & 0x003
t6 = t6 + t5
t6 = 0
Yes
end
No
t5 = 16
set a0 to 10
t5 = t5 >> 1
t5 = 0
No
No
a0 = a0 + 10
t6 != t5
No
read dot
YEs
a3
!= 0x00
No
Yes
a5 = 1
a3 = grey
a2 = a0 + 9
No
a1 = a1 - 1
Read dot
Yes
a4 = 1
Yes
a3 = 0x00
end
No
a1 = a1 +1
Read dot
a3 != 0
Yes
No
a3 = 0x92 (grey)
Draw horizontal line
End


read dot
t0 = a0 & 0x7f (get 7 LSB of a0)
t1 = a1 & 0x3f (get 6 LSB of a1)
t1 = t1 << 7
t0 = t1 | t0 (t0 = {a1[5:0], a0[6:0]})
write t0 to VGA address (0x11000120)
read the color from VGA read address (0x11000160) and set to a3
end


ISR
s0 = s0 + 1
store x0 to buzzer (buzzer off)
set mie to 1 to enable interrupts
ISR end


Division by 10
take dividend as remainder
set divisor to 10
initialize quotient to 0
dividend < divisor
Yes
No
return dividend (remainder) and quotient
dividend = dividend - divisor
increment quotient by 1
end


B
set t5 = s10
t2 = 0
t3 = 0
t4 = 20
a0 = t5
Divide read value by 10
t5 = a1
shift remainder of division left i times. a0 = a0 << t3
t2 = t2 + a0
t3 = t3 + 4
t3< t4
Yes
No
Output t2 to 7 segment
s9 = s9 + 1
t0 = s6 + 9
load t0 from memory (upcoming note) and store to t0
t0 < 0
Yes
No
store x0 to buzzer (buzzer off)
a3 = 0x 1a(Green)
draw background
t0 = -1
t1 = 25000000
t0 = t0 + 1
Yes
t0 != t1
No
Start
s11 = 0xFFFFFFFF
sp = 0x10000
s2 = VG_ADDR (0x11000120)
s3 = VG_COLOR (0x11000140)
s4 = PIANO_KEYS_QUEUE_ARR
reset PIANO_KEYS_QUEUE_ARR to 0
s5 = SWITCHES_AD (0x11000000)
s6 = ODE_TO_JOY_NOTES
s7 = ODE_TO_JOY_TIMES
s8 = 0 (times index)
s9 = 0 (notes index)
s10 = 0 (score)
output initial score (0) to 7 seg
a3 = 0x7B (light blue)
draw background
a3 = 0xE0 (red)
a0 = 0
a1 = 39
a2 = 19
draw horizontal line
a0 = 60
a2 = 79
draw horizontal line
s0 = 0
t0 = ISR addr
save ISR addr to MTVEC
set mie to 1 to enable interrupts
A
No
store x0 to buzzer (buzzer off)
a3 = 0xE0(Red)
draw background
t0 = -1
t1 = 25000000
t0 = t0 + 1
Yes
t0 != t1
A
t0 = s0 & 1
Yes
t0 != 0
No
call fill_piano_keys_arr
call shift_down
call validate
a4 != 0
Yes
No
call eval switches
a4 != 0
Yes
No
a5 = 0
Yes
No
store x0 to buzzer (buzzer off)
t0 = -1
t1 = 937500
t0 = t0 + 1
Yes
t0 != t1
No
a5 = 0
Yes
No
t0 = s6 + s9
read addr in t0 from memory and store in t0
play/store t0 (note) on buzzer
s10 = s10 + 1
B