Final Report

Jules Hajjar & Anthony Luna

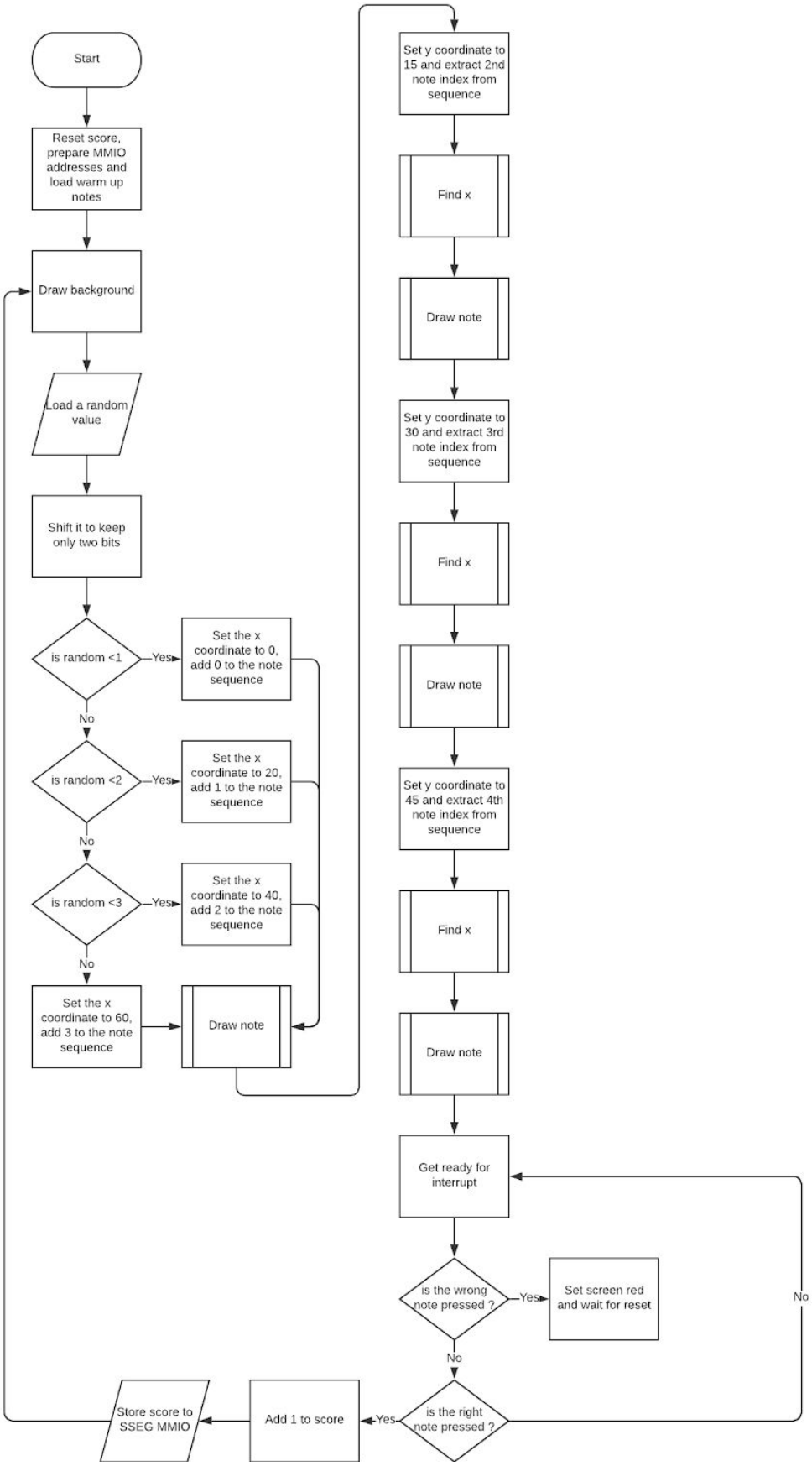
Introduction

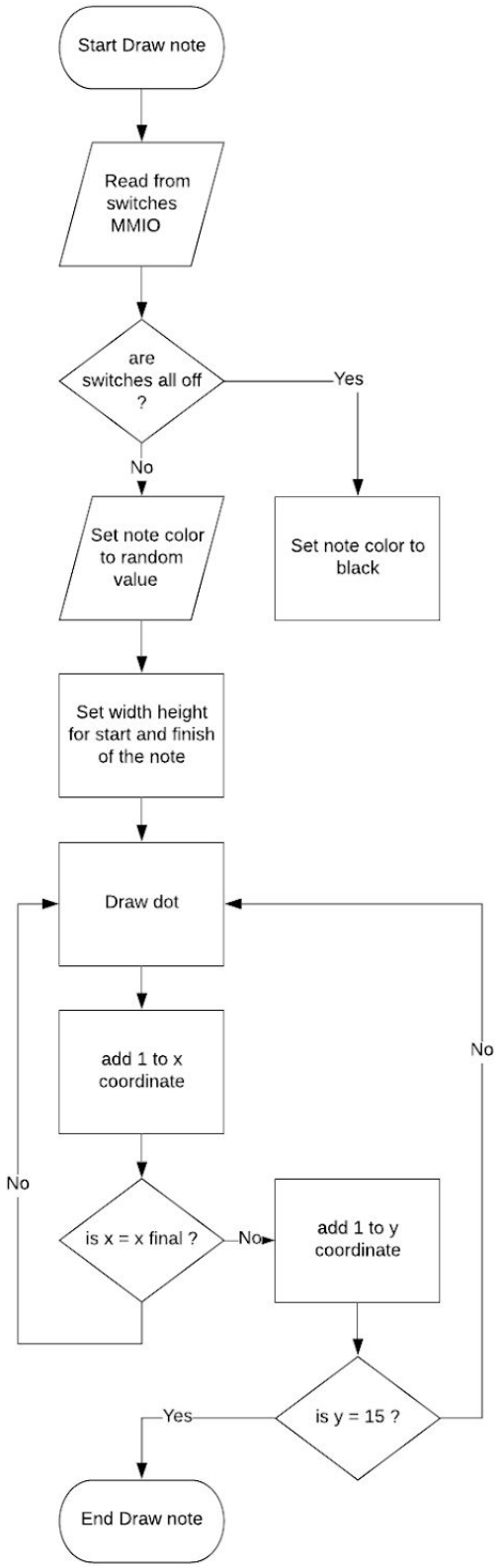
The game is essentially a rhythm game, where you have to time tapping keys with “notes” coming down from the top of the screen. It is similar to popular modern games such as Guitar Hero, or the mobile games where you tap on the screen in time with the music. It does not have levels, but each game is different because the notes come down randomly every time. To play the game, a keyboard is required as well as a vga display. The game is lightweight and operates on a 50MHz OTTER processor.

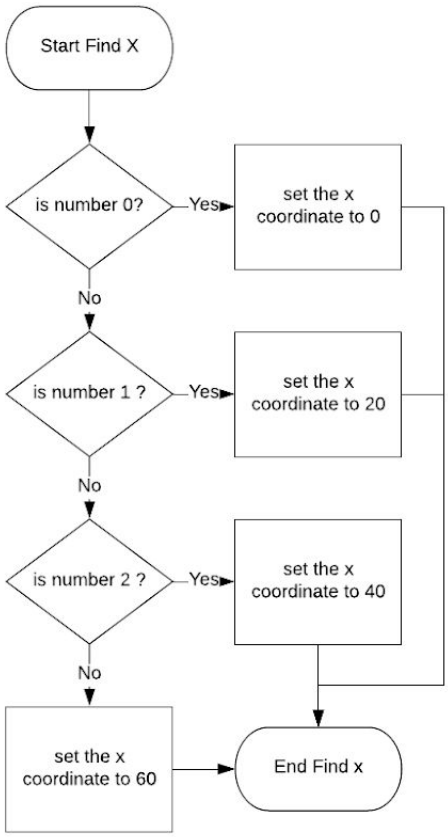
Software Design

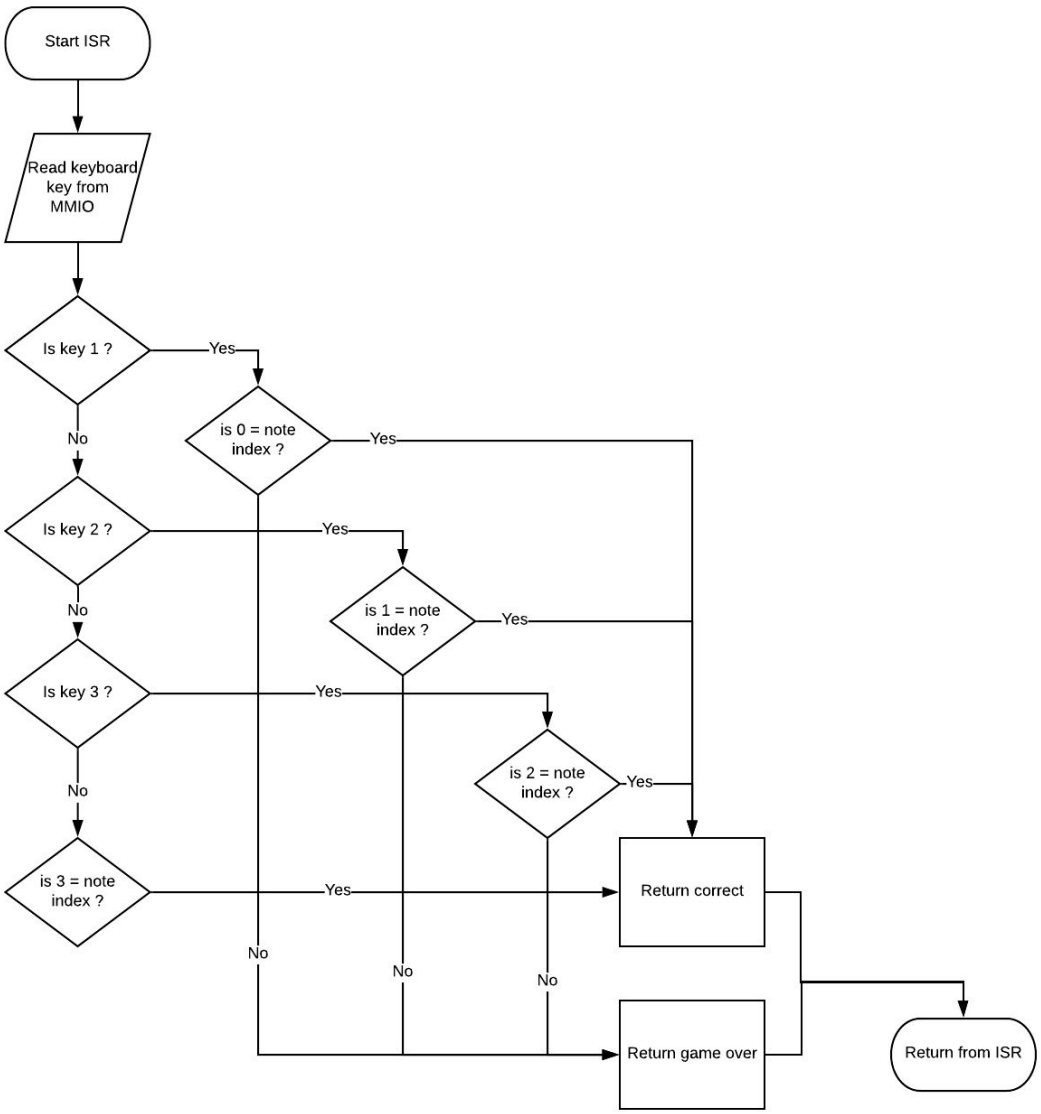
During the initialization, three fixed notes are generated and drawn on screen as a “warm up” sequence. The four notes displayed on screen are stored in a word that shifts everytime a new note gets loaded.

The program runs in a loop, it draws a background, generates a new random note, draws it, and draws the three preceding notes after it. Once everything is drawn, the program waits for an interrupt to be triggered. This interrupt checks to see if the key pressed corresponds to the right note. Once this is determined, the program either proceeds to add one to the score and go back up the loop, or goes to a game over state, waiting to be reset.


Start
Reset score, prepare MMIO addresses and load warm up notes
Draw background
Load a random value
Shift it to keep only two bits
is random <1
Yes
Set the x coordinate to 0, add 0 to the note sequence
No
is random <2
Yes
Set the x coordinate to 20, add 1 to the note sequence
No
is random <3
Yes
Set the x coordinate to 40, add 2 to the note sequence
No
Set the x coordinate to 60, add 3 to the note sequence
Draw note
Set y coordinate to 15 and extract 2nd note index from sequence
Find x
Draw note
Set y coordinate to 30 and extract 3rd note index from sequence
Find x
Draw note
Set y coordinate to 45 and extract 4th note index from sequence
Find x
Draw note
Get ready for interrupt
is the wrong note pressed ?
Yes
Set screen red and wait for reset
No
No
is the right note pressed ?
Yes
Add 1 to score
Store score to SSEG MMIO


Start Draw note
Read from switches MMIO
are switches all off ?
Yes
No
Set note color to random value
Set note color to black
Set width height for start and finish of the note
Draw dot
add 1 to x coordinate
is x = x final ?
No
add 1 to y coordinate
is y = 15 ?
No
Yes
End Draw note


Start Find X
is number 0?
Yes
set the x coordinate to 0
No
is number 1 ?
Yes
set the x coordinate to 20
No
is number 2 ?
Yes
set the x coordinate to 40
No
set the x coordinate to 60
End Find x


Start ISR
Read keyboard key from MMIO
Is key 1 ?
Yes
No
is 0 = note index ?
Yes
Is key 2 ?
Yes
No
is 1 = note index ?
Yes
Is key 3 ?
Yes
No
is 2 = note index ?
Yes
is 3 = note index ?
Yes
Return correct
No
No
No
Return game over
Return from ISR

Appendix

```
.eqv RANDNUM_ADDR, 0x11240000
.eqv BG_COLOR, 0xFF                  #  (7/7 red, 7/7 green, 3/3 blue)
.eqv NOTE_COLOR, 0x00
.eqv VG_ADDR, 0x11100000
.eqv VG_COLOR, 0x11140000
.eqv VG_READ_AD, 0x11040000
.eqv KEYBOARD_AD, 0x11200000
.eqv SSEG, 0x110C0000

init:
      li a0, SSEG
      addi t2, x0, 0                 #reset the score
      sw t2, (a0)

      li s9, BG_COLOR
  li sp, 0x10000                     #initialize stack pointer
      li s2, VG_ADDR           #load MMIO addresses
      li s3, VG_COLOR
      li t6, 1                       #gotta be different than zero
      li a5, 258                     # load some notes

main:
# get ready for new random note to be loaded in the sequence and shifts
# the other notes for next displaying
slli a5, a5, 8
# fill screen using default color
call draw_background                 # must not modify s2, s3

draw_new_rect:
# needs the random 32bit number from memory and maps
# it down to 4 values before drawing that new note
      lw s11, RANDNUM_ADDR
      srli s11, s11, 30              #read only two bits of random number
      li s5, 15
      li s6, 1                       #get ready for comparisons
      add s7, s6, s6
      add s8, s6, s7
      blt s11, s6, draw_firstpos
      blt s11, s7, draw_secondpos
      blt s11, s8, draw_thirdpos

# the following branches make sure the right note is drawn
draw_fourthpos:
      addi a5, a5, 3
      li a0, 60                # X coordinate
      li a1, 0                 # Y coordinate
```

```
      call draw_note          # must not modify s2, s3
      j next_main
draw_thirdpos:
      addi a5, a5, 2
      li a0, 40               # X coordinate
      li a1, 0                # Y coordinate
      call draw_note          # must not modify s2, s3
      j next_main
draw_secondpos:
      addi a5, a5, 1
      li a0, 20               # X coordinate
      li a1, 0                # Y coordinate
      call draw_note          # must not modify s2, s3
      j next_main
draw_firstpos:
      li a0, 0                # X coordinate
      li a1, 0                # Y coordinate
      call draw_note          # must not modify s2, s3
next_main:

li a0, 0
# Here we draw the next notes in the sequence, which requires finding
# their starting coordinate from the note value in the sequence
li a1, 15
srli a6, a5, 8                      #go to index of that note
andi a6, a6, 3                      #extract the value
call findX
call draw_note

li a1, 30
srli a6, a5, 16
andi a6, a6, 3
call findX
call draw_note

li a1, 45
srli a6, a5, 24
andi a6, a6, 3
call findX
call draw_note

li t5, 1                            #reset the game over flag
la t0, ISRingame
csrrw x0, mtvec, t0                 #setup ISR address

wait_for_userinput:                 #wait for keypress
    li s10, 1
```

```
    csrrw x0, mie, s10              #enable interrupts
    beqz t6, goodnote          #if right key is pressed
    beqz t5, gameover          #if wrong key is pressed
    j wait_for_userinput

gameover:
    li s9, 0xE0                #display red background
    call draw_background
    j gameover

goodnote:
    li t6, 1                   #reset goodnote flag
    li a0, SSEG                #get ready to show score
    addi t2, t2, 1             #add one to score
    sw t2, (a0)                #display score
    j main

############################################################

# Fills the 60x80 grid with one color using successive calls to
# draw_horizontal_line
# Modifies (directly or indirectly): t0, t1, t4, a0, a1, a2, a3
draw_background:
       addi sp,sp,-4
       sw ra, 0(sp)
       addi a3, s9, 0          #use default color
       li a1, 0                #a1= row_counter
       li t4, 60               #max rows
start:
   li a0, 0
       li a2, 79               #total number of columns
       call draw_horizontal_line     # must not modify: t4, a1, a3
       addi a1,a1, 1
       bne t4,a1, start        #branch to draw more rows
       lw ra, 0(sp)
       addi sp,sp,4
       ret

# draws a horizontal line from (a0,a1) to (a2,a1) using color in a3
# Modifies (directly or indirectly): t0, t1, a0, a2
draw_horizontal_line:
       addi sp,sp,-4
       sw ra, 0(sp)
       addi a2,a2,1                  #go from a0 to a2 inclusive
draw_horiz1:
       call draw_dot                 # must not modify: a0, a1, a2, a3
       addi a0,a0,1
       bne a0,a2, draw_horiz1
```

```
        lw ra, 0(sp)
        addi sp,sp,4
        ret

# draws a dot on the display at the given coordinates:
#     (X,Y) = (a0,a1) with a color stored in a3
#     (col, row) = (a0,a1)
# Modifies (directly or indirectly): t0, t1
draw_dot:
          andi t0,a0,0x7F       # select bottom 7 bits (col)
          andi t1,a1,0x3F       # select bottom 6 bits  (row)
          slli t1,t1,7                #  {a1[5:0],a0[6:0]}
          or t0,t1,t0                 # 13-bit address
          sw t0, 0(s2)                # write 13 address bits to register
          sw a3, 0(s3)                # write color data to frame buffer
          ret
# draws a note on the display at the given coordinates:
#     (X,Y) = (a0,a1) with a color stored in a3
#     (col, row) = (a0,a1)
# Modifies (directly or indirectly): a3,s4,s5,s6,s7,t0,t1
draw_note:
          li a3, 0x11000000           #check the switches for easter egg
          lw a3, (a3)
          beqz a3, normalcolor  #if no switch up then no easter egg

          lw a3, RANDNUM_ADDR   #if switch up then do random color
          j eastereggend

       normalcolor:
          li a3, 0x00                 #black

       eastereggend:
          addi s4, a0, 20             #width
          addi s5, a1, 14             #height
          addi s6, a0, 0              #offset by note position
          addi s7, a1, 0
       draw_note_main:
          andi t0,s6,0x7F       # select bottom 7 bits (col)
          andi t1,s7,0x3F       # select bottom 6 bits  (row)
          slli t1,t1,7                #  {a1[5:0],a0[6:0]}
          or t0,t1,t0                 # 13-bit address
          sw t0, 0(s2)                # write 13 address bits to register
          sw a3, 0(s3)                # write color data to frame buffer
          addi s6, s6, 1        #go to next pixel in line
          beq s6, s4, nextline  #if the end of the line is reached go to the next
one
          j draw_note_main
       nextline:
```

```
         beq s7, s5, end_draw_note   #if the end of the note is drawn return to
loop
         addi s7, s7, 1        #otherwise add one to the row
         addi s6, a0, 0              #and reset the column to the first position
         j draw_note_main
      end_draw_note:
         ret

# finds the x coordinate from the stored 0-3 position given by a6 returning it
in a0
# Modifies (directly or indirectly): a0, s5, s6, s7
findX:
      li s5, 0
      li s6, 1
      li s7, 2
      beq a6, s5, firstpos
      beq a6, s6, secondpos
      beq a6, s7, thirdpos
      fourthpos:
         li a0, 60            # X coordinate
         ret
      thirdpos:
         li a0, 40            # X coordinate
         ret
      secondpos:
         li a0, 20            # X coordinate
         ret
      firstpos:
         li a0, 0             # X coordinate
         ret

##################################################
# when a key is pressed this ISR checks if the key pressed corresponds to a
note by
# checking if the note has the right index. Ignores all other keypresses.
# Modifies (directly or indirectly): s0, t6, t5, t3

ISRingame:
      li s0, KEYBOARD_AD
      lw s0, 0(s0)
      srli a0, a5, 24
      li t6, 0x25             #load key to be tested against
      beq s0, t6, third_note_test
      li t6, 0x26
      beq s0, t6, second_note_test
      li t6, 0x1E
      beq s0, t6, first_note_test
      li t6, 0x16
```

```
beq s0, t6, zeroeth_note_test
mret
zeroeth_note_test:
   beqz a0, correct            #jump to the correct section
   li t5, 0                    #otherwise prepare for game over
   mret
first_note_test:
   li t3, 1
   beq t3, a0, correct
   li t5, 0
   mret
second_note_test:
   li t3, 2
   beq t3, a0, correct
   li t5, 0
   mret
third_note_test:
   li t3, 3
   beq t3, a0, correct
   li t5, 0
   mret
correct:
   li t6, 0              #correct !
   mret
```

Final tip : try flipping a switch ●

Anthony Luna
& Jules Hajjar

Official
RISC V OTTER MCU
Seal of Quality

GUITARISC

Music at your fingertips

↦ What is GuitarRISC

A simple rhythm game, designed to test timing and coordination.

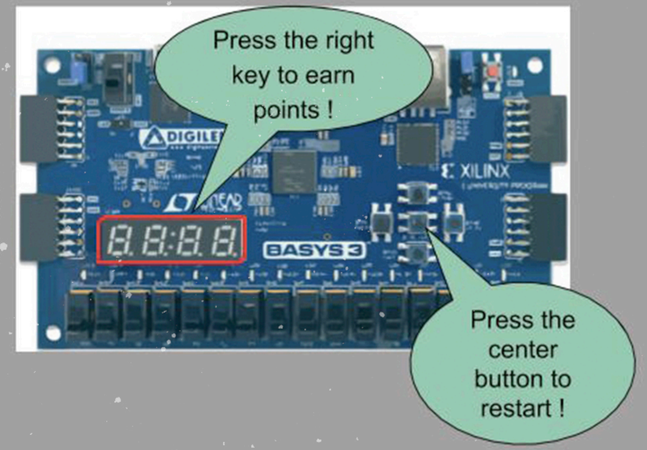
Everytime you start a game, the sequence will be randomized after the three warm up notes, allowing for unpredictable gameplay and endless fun.

If the wrong button is pressed, Game Over !

Play along some of your own music to keep the tempo and become a professional guitarisc ♫

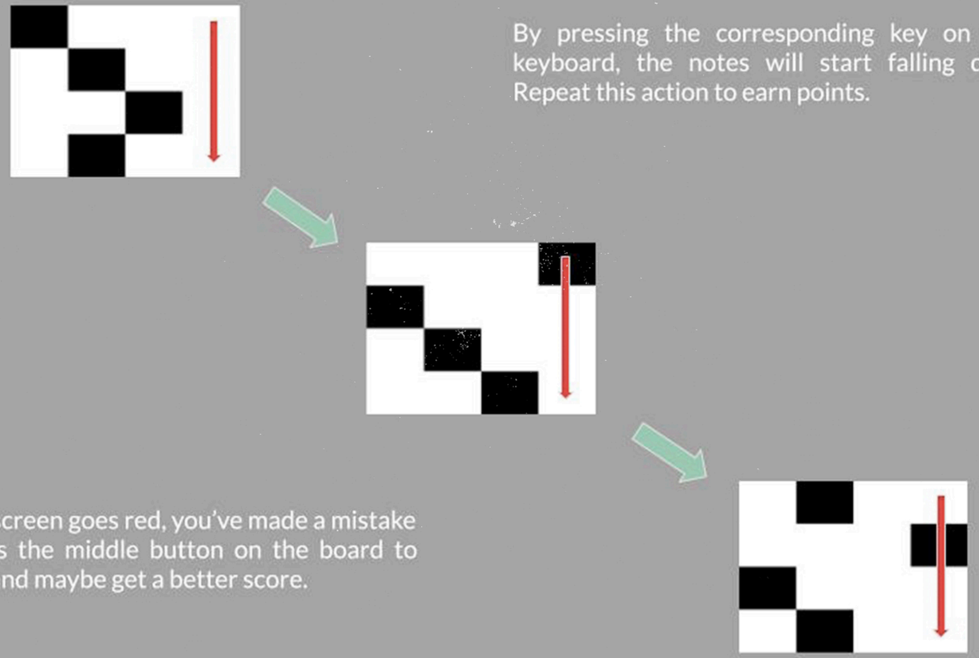
↦ HOW TO PLAY

This game requires no musical knowledge of any kind. It is about speed and precision. Make sure your fingers are warmed up before playing !



Any other on the keyboard will be ignored by the game, so do not worry about setting your wrist on other keys.

⤇ HOW TO PLAY



By pressing the corresponding key on your keyboard, the notes will start falling down. Repeat this action to earn points.

If the screen goes red, you've made a mistake ! Press the middle button on the board to retry and maybe get a better score.

⤇ MEMO