Assembly Project: Columns

(Jeff) Jaehyuk Ryu

November 27, 2025

1 Instruction and Summary

1. Which milestones were implemented?

   Milestone 1 to 5 (Complete).

2. How to view the game:

   (a) unit weight in pixels: 4, for both x and y.

   (b) width: 32 pixels

   (c) height: 16 pixels

   (d) stage size: 12 * 16

   (e) upper middle gray box: next column

   (f) lower middle gray box: save column

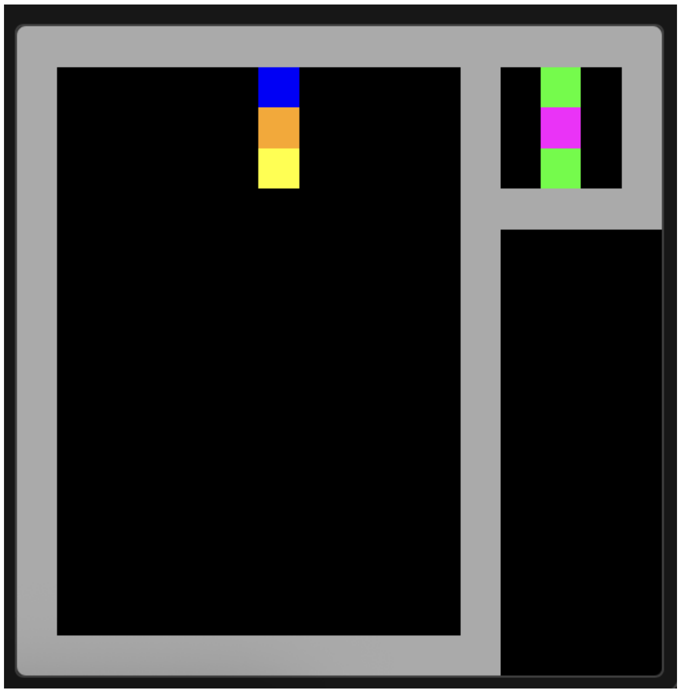


Figure 1: Instructions

3. Game Summary:

   - a typical columns game, with few additional features.
   - game ends if height reaches the top 2 rows, or Q pressed.

- a, d to move the block horizontally, s to move the block downward.
- w shifts the orientation of the block.
- e to save the block for now.
- p pauses the game until p is pressed for the second time.
- at every ’landing’, checks for collisions.
- additional features are implemented, see below.

2 Attribution Table

Jaehyuk Ryu (1009558079)
Pre-build set-up
Stage set-up
Painting structure set-up
Keystroke set-up
Collision detection set-up
Game over detection set-up
MILESTONE 4, 5 set-up

3 MileStones

1. Milestone 1, stage set up.
   - Drew lines in 12*16 boundary, as a game stage.

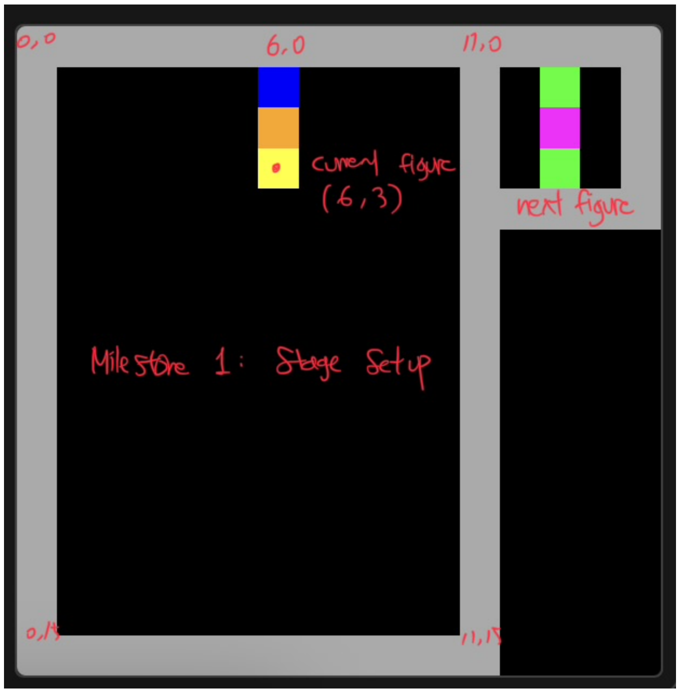
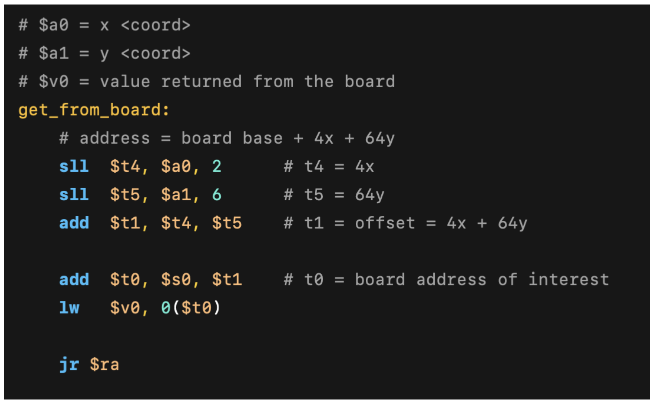


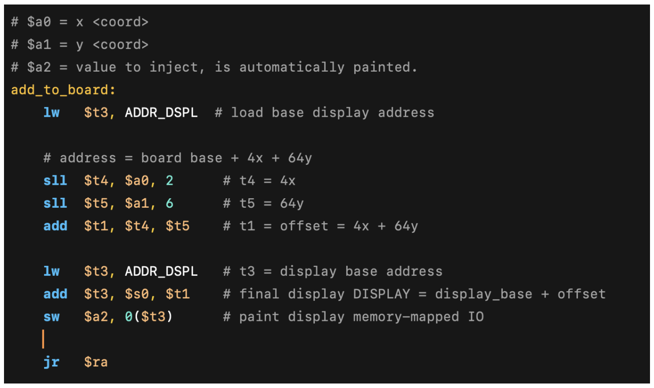
Figure 2: Milestone 1 bitmap snapshot

   - Top-right corner is preserved for displaying the next block.
   - new block starts at (6,3) always, with bottom gem being the standard for its location. If a gem is already filled up in the three spaces when a new block is created, it counts as a game over.



```
# $a0 = x <coord>
# $a1 = y <coord>
# $v0 = value returned from the board
get_from_board:
    # address = board base + 4x + 64y
    sll  $t4, $a0, 2       # t4 = 4x
    sll  $t5, $a1, 6       # t5 = 64y
    add  $t1, $t4, $t5     # t1 = offset = 4x + 64y

    add  $t0, $s0, $t1     # t0 = board address of interest
    lw   $v0, 0($t0)

    jr $ra
```

Figure 3: Getter function



```
# $a0 = x <coord>
# $a1 = y <coord>
# $a2 = value to inject, is automatically painted.
add_to_board:
    lw   $t3, ADDR_DSPL  # load base display address

    # address = board base + 4x + 64y
    sll  $t4, $a0, 2      # t4 = 4x
    sll  $t5, $a1, 6      # t5 = 64y
    add  $t1, $t4, $t5    # t1 = offset = 4x + 64y

    lw   $t3, ADDR_DSPL   # t3 = display base address
    add  $t3, $s0, $t1    # final display DISPLAY = display_base + offset
    sw   $a2, 0($t3)      # paint display memory-mapped IO

    jr   $ra
```

Figure 4: Setter function. Stores + Paints

- Created a getter and setter for the board, calculating the offset inside the functions.

2. Milestone 2, movements and keystrokes set up.
   - A, D for left and right movements
   - S for downward maneuver
   - W for shuffling the orientation
   - Q for quitting the game, manually (gracefully!)

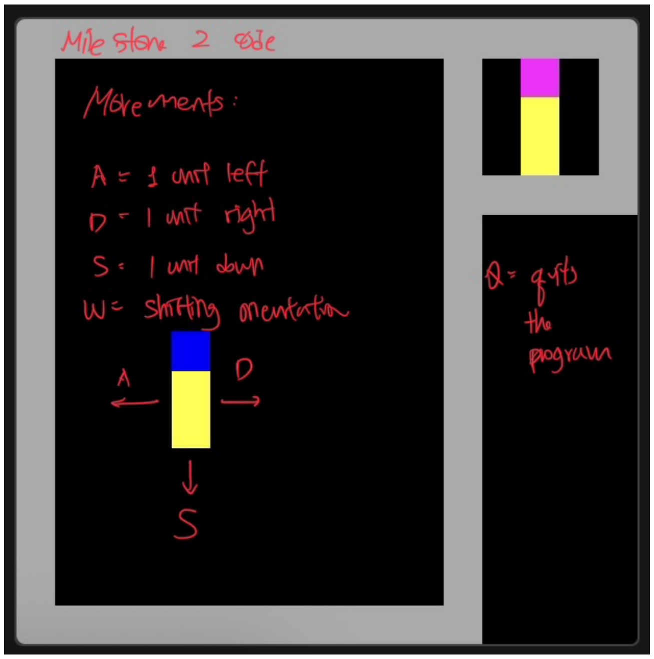
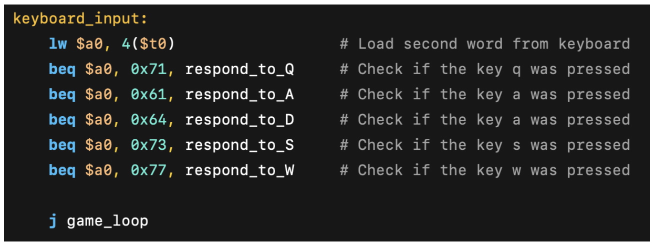


Figure 5: Milestone 2 bitmap snapshot

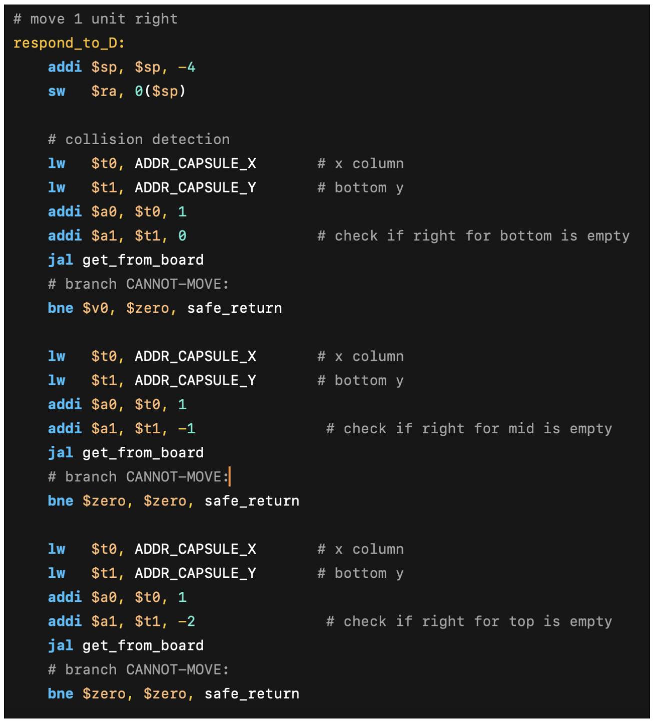


```
keyboard_input:
    lw $a0, 4($t0)                  # Load second word from keyboard
    beq $a0, 0x71, respond_to_Q     # Check if the key q was pressed
    beq $a0, 0x61, respond_to_A     # Check if the key a was pressed
    beq $a0, 0x64, respond_to_D     # Check if the key a was pressed
    beq $a0, 0x73, respond_to_S     # Check if the key s was pressed
    beq $a0, 0x77, respond_to_W     # Check if the key w was pressed

    j game_loop
```

Figure 6: Keyboard Input Mapper

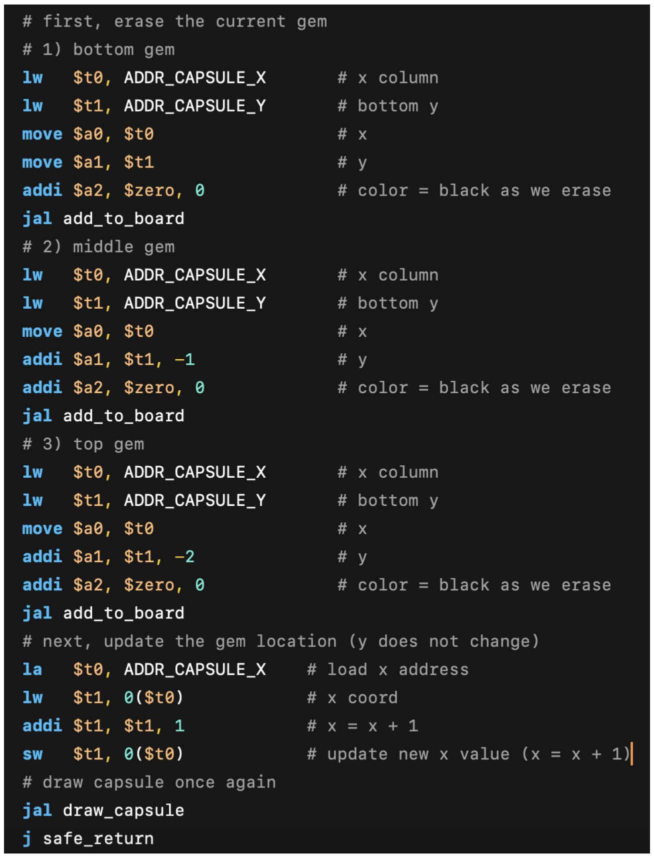
3. Milestone 3, Collision detection.

   - When moving left and right, prevent moving if there is a block to the place the block would go if it were empty.



```
# move 1 unit right
respond_to_D:
    addi $sp, $sp, -4
    sw   $ra, 0($sp)

    # collision detection
    lw   $t0, ADDR_CAPSULE_X       # x column
    lw   $t1, ADDR_CAPSULE_Y       # bottom y
    addi $a0, $t0, 1
    addi $a1, $t1, 0               # check if right for bottom is empty
    jal get_from_board
    # branch CANNOT-MOVE:
    bne $v0, $zero, safe_return

    lw   $t0, ADDR_CAPSULE_X       # x column
    lw   $t1, ADDR_CAPSULE_Y       # bottom y
    addi $a0, $t0, 1
    addi $a1, $t1, -1               # check if right for mid is empty
    jal get_from_board
    # branch CANNOT-MOVE:
    bne $zero, $zero, safe_return

    lw   $t0, ADDR_CAPSULE_X       # x column
    lw   $t1, ADDR_CAPSULE_Y       # bottom y
    addi $a0, $t0, 1
    addi $a1, $t1, -2               # check if right for top is empty
    jal get_from_board
    # branch CANNOT-MOVE:
    bne $zero, $zero, safe_return
```

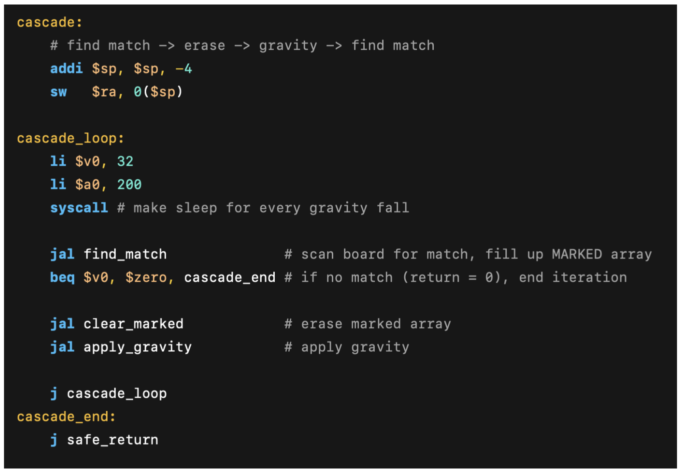
Figure 7: Checker code for the possible future location when D is pressed.



```
# first, erase the current gem
# 1) bottom gem
lw   $t0, ADDR_CAPSULE_X       # x column
lw   $t1, ADDR_CAPSULE_Y       # bottom y
move $a0, $t0                  # x
move $a1, $t1                  # y
addi $a2, $zero, 0             # color = black as we erase
jal add_to_board
# 2) middle gem
lw   $t0, ADDR_CAPSULE_X       # x column
lw   $t1, ADDR_CAPSULE_Y       # bottom y
move $a0, $t0                  # x
addi $a1, $t1, -1              # y
addi $a2, $zero, 0             # color = black as we erase
jal add_to_board
# 3) top gem
lw   $t0, ADDR_CAPSULE_X       # x column
lw   $t1, ADDR_CAPSULE_Y       # bottom y
move $a0, $t0                  # x
addi $a1, $t1, -2              # y
addi $a2, $zero, 0             # color = black as we erase
jal add_to_board
# next, update the gem location (y does not change)
la   $t0, ADDR_CAPSULE_X    # load x address
lw   $t1, 0($t0)            # x coord
addi $t1, $t1, 1            # x = x + 1
sw   $t1, 0($t0)            # update new x value (x = x + 1)
# draw capsule once again
jal draw_capsule
j safe_return
```

Figure 8: Code for cleaning up prev spot, setting up new block, drawing it.

   - When moving downwards, if it hits a block that is not empty (that is, either the stage–rock bottom or another gem that was placed before), check for matches in a loop until there is no match.



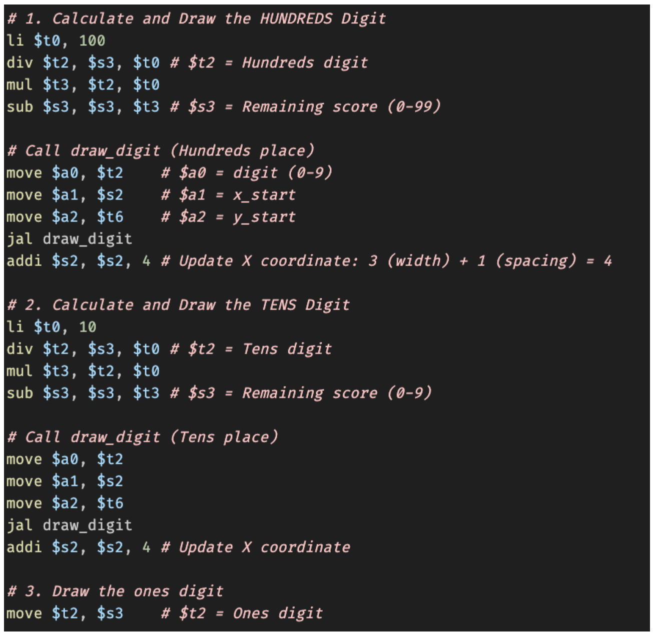
```
cascade:
    # find match -> erase -> gravity -> find match
    addi $sp, $sp, -4
    sw   $ra, 0($sp)

cascade_loop:
    li $v0, 32
    li $a0, 200
    syscall # make sleep for every gravity fall

    jal find_match              # scan board for match, fill up MARKED array
    beq $v0, $zero, cascade_end # if no match (return = 0), end iteration

    jal clear_marked            # erase marked array
    jal apply_gravity           # apply gravity

    j cascade_loop
cascade_end:
    j safe_return
```

Figure 9: Basic code structure for collision detection

   - Here, an array (4*16*16=1024bits) is used to mark the locations of matches. After the entire scan, another iteration through the mark array starts off. The offset from mark base to match cell would be the same as the one from base address of display to the actual cell to erase. This idea was used to find the exact location of display to erase.

4. Milestone 4 + 5: implemented 1 HARD, 7 EASY features.

- 1st HARD feature: Hard(1), display scoreboard.



```
# 1. Calculate and Draw the HUNDREDS Digit
li $t0, 100
div $t2, $s3, $t0 # $t2 = Hundreds digit
mul $t3, $t2, $t0
sub $s3, $s3, $t3 # $s3 = Remaining score (0-99)

# Call draw_digit (Hundreds place)
move $a0, $t2    # $a0 = digit (0-9)
move $a1, $s2    # $a1 = x_start
move $a2, $t6    # $a2 = y_start
jal draw_digit
addi $s2, $s2, 4 # Update X coordinate: 3 (width) + 1 (spacing) = 4

# 2. Calculate and Draw the TENS Digit
li $t0, 10
div $t2, $s3, $t0 # $t2 = Tens digit
mul $t3, $t2, $t0
sub $s3, $s3, $t3 # $s3 = Remaining score (0-9)

# Call draw_digit (Tens place)
move $a0, $t2
move $a1, $s2
move $a2, $t6
jal draw_digit
addi $s2, $s2, 4 # Update X coordinate

# 3. Draw the ones digit
move $t2, $s3    # $t2 = Ones digit
```

Figure 10: Code that calculates each digit by doing modulo operation and draws the digit accordingly.

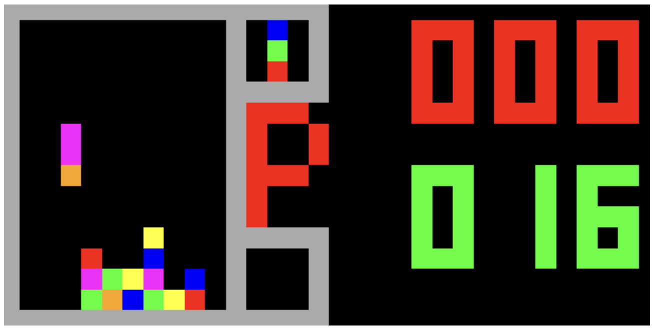
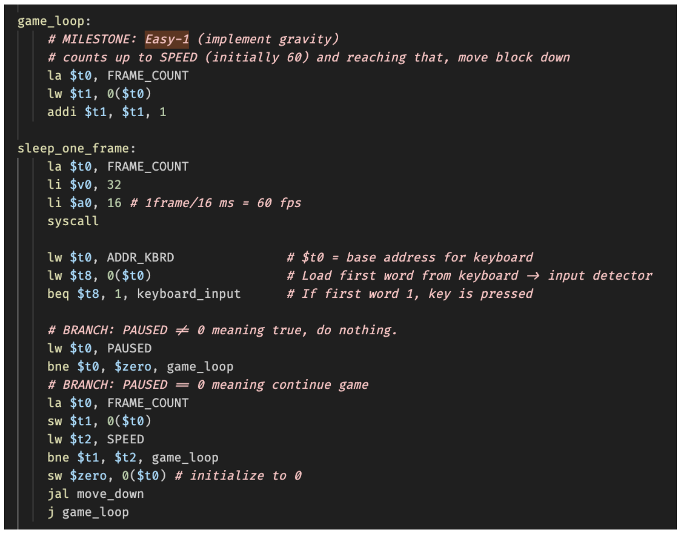


Figure 11: How it looks on the display.

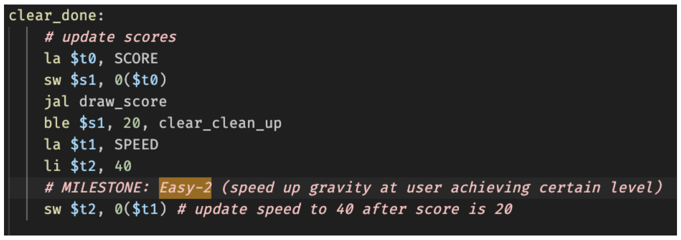
- 1st EASY feature: Easy(1), add gravity.



```
game_loop:
    # MILESTONE: Easy-1 (implement gravity)
    # counts up to SPEED (initially 60) and reaching that, move block down
    la $t0, FRAME_COUNT
    lw $t1, 0($t0)
    addi $t1, $t1, 1

sleep_one_frame:
    la $t0, FRAME_COUNT
    li $v0, 32
    li $a0, 16 # 1frame/16 ms = 60 fps
    syscall

    lw $t0, ADDR_KBRD               # $t0 = base address for keyboard
    lw $t8, 0($t0)                  # Load first word from keyboard → input detector
    beq $t8, 1, keyboard_input      # If first word 1, key is pressed

    # BRANCH: PAUSED ≠ 0 meaning true, do nothing.
    lw $t0, PAUSED
    bne $t0, $zero, game_loop
    # BRANCH: PAUSED == 0 meaning continue game
    la $t0, FRAME_COUNT
    sw $t1, 0($t0)
    lw $t2, SPEED
    bne $t1, $t2, game_loop
    sw $zero, 0($t0) # initialize to 0
    jal move_down
    j game_loop
```

Figure 12: Code that makes FRAMES by making a loop that sleeps for 1/60 second and moves one block down by 60 frames.

- 2nd EASY feature: Easy(2), make gravity faster as score reaches certain level.



```
clear_done:
    # update scores
    la $t0, SCORE
    sw $s1, 0($t0)
    jal draw_score
    ble $s1, 20, clear_clean_up
    la $t1, SPEED
    li $t2, 40
    # MILESTONE: Easy-2 (speed up gravity at user achieving certain level)
    sw $t2, 0($t1) # update speed to 40 after score is 20
```

Figure 13: Code that executes after removing (scoring columns), and checks if score went over 20 to set SPEED 40 (the frame needed to automatically drag 1 tick down).

- 3rd EASY feature: Easy(4), display gameover state and add retry feature.

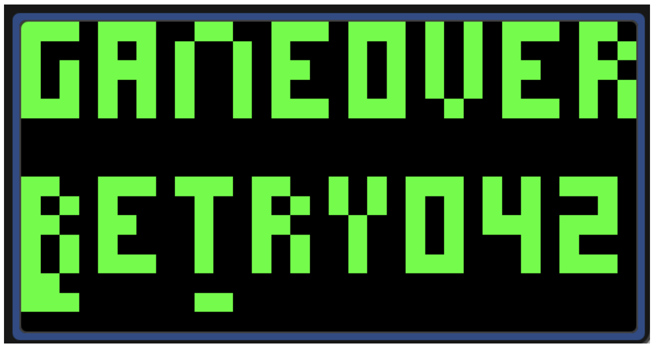
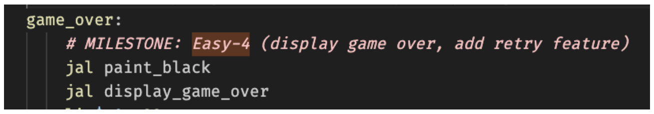


Figure 14: Game over screen that is displayed at game over. RETRY text flickers every 400ms. At pressing any button, game resets. (Pixels at bottom left are intended.)



```
game_over:
    # MILESTONE: Easy-4 (display game over, add retry feature)
    jal paint_black
    jal display_game_over
```

Figure 15: How it looks on the display.

- 4th EASY feature: Easy(6), add pause feature when pressing p.

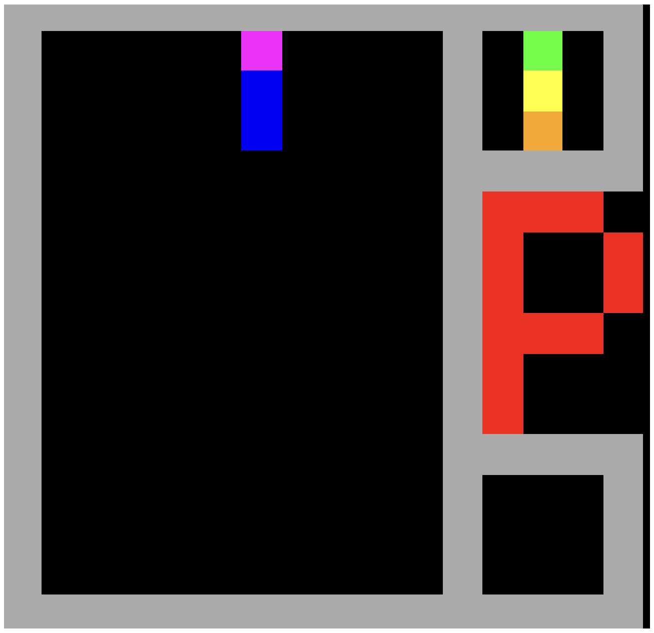
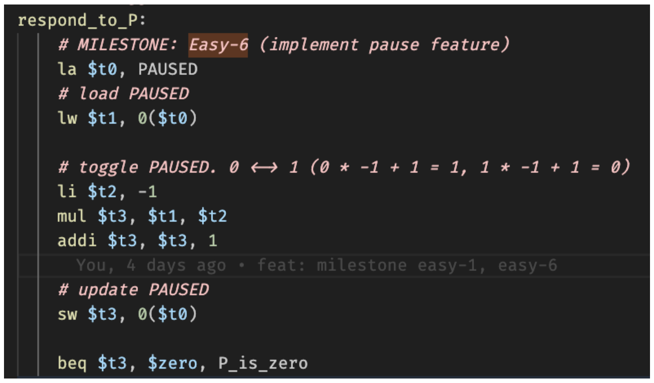


Figure 16: Paused screen. The block does not fall automatically, and any game interaction is ignored, excluding the resume key p.



```
respond_to_P:
    # MILESTONE: Easy-6 (implement pause feature)
    la $t0, PAUSED
    # load PAUSED
    lw $t1, 0($t0)

    # toggle PAUSED. 0 <-> 1 (0 * -1 + 1 = 1, 1 * -1 + 1 = 0)
    li $t2, -1
    mul $t3, $t1, $t2
    addi $t3, $t3, 1
      You, 4 days ago • feat: milestone easy-1, easy-6
    # update PAUSED
    sw $t3, 0($t0)

    beq $t3, $zero, P_is_zero
```

Figure 17: The code that toggles state word PAUSED. Toggle boolean logic is state $\times -1 + 1$.

- 5th EASY feature: Easy(9), display highest score.

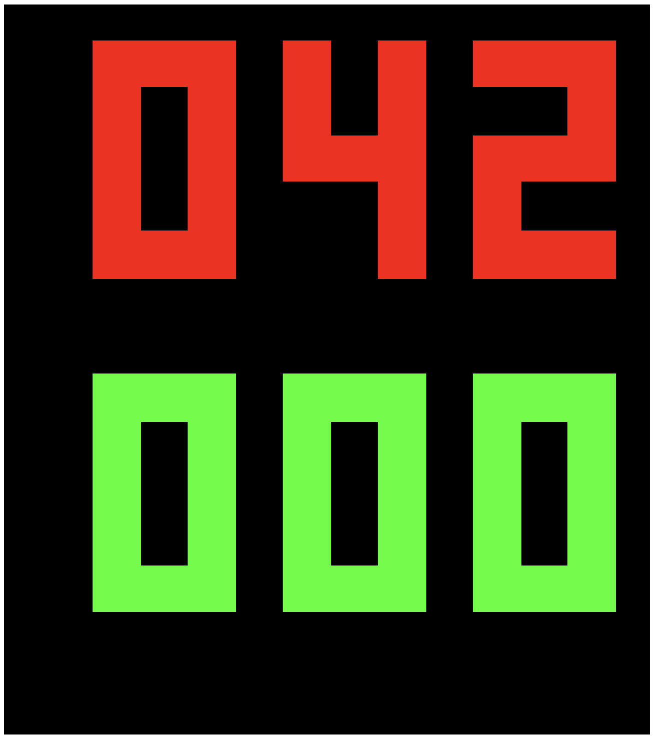
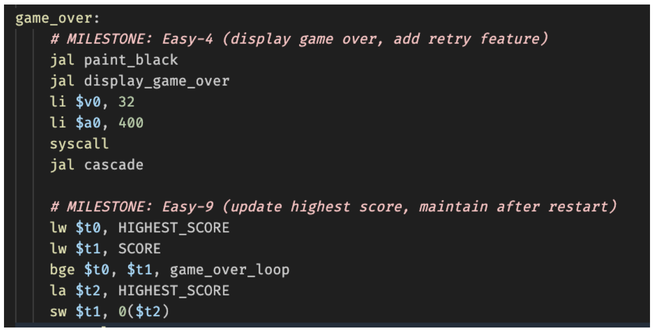


Figure 18: The updated highest scoreboard (in RED). After every retry, the score is updated.



```
game_over:
    # MILESTONE: Easy-4 (display game over, add retry feature)
    jal paint_black
    jal display_game_over
    li $v0, 32
    li $a0, 400
    syscall
    jal cascade

    # MILESTONE: Easy-9 (update highest score, maintain after restart)
    lw $t0, HIGHEST_SCORE
    lw $t1, SCORE
    bge $t0, $t1, game_over_loop
    la $t2, HIGHEST_SCORE
    sw $t1, 0($t2)
```

Figure 19: Code that updates scoreboard. Executed on gameover.

- 6th EASY feature: Easy(10), show next block

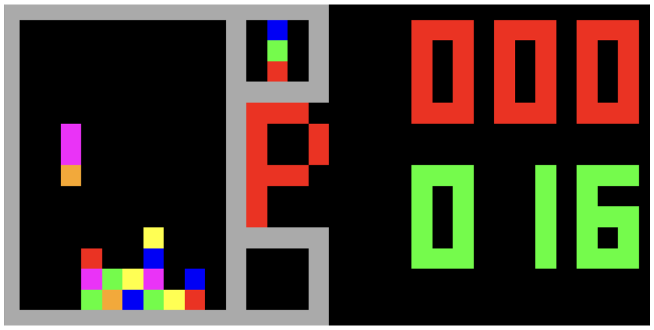
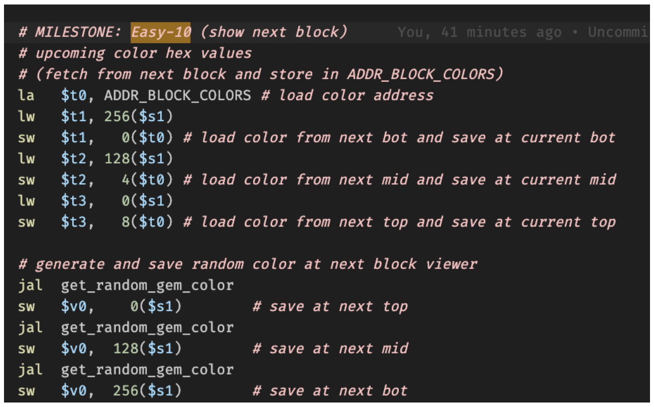


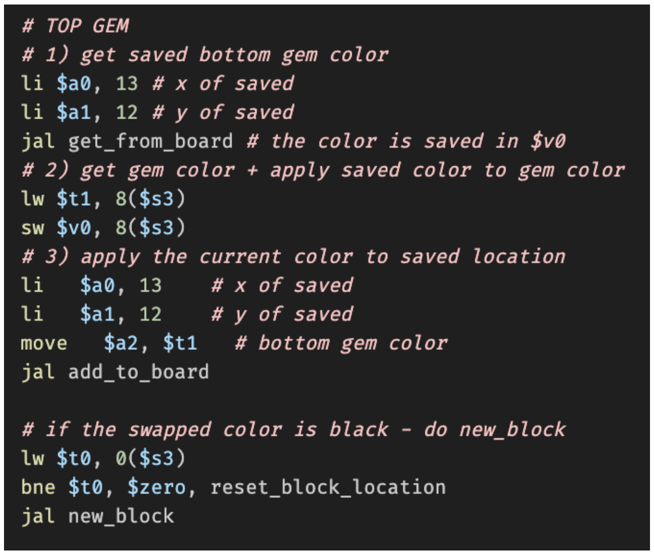
Figure 20: Upper middle gray box is where the next column block will appear.



```
# MILESTONE: Easy-10 (show next block)      You, 41 minutes ago • Uncommi
# upcoming color hex values
# (fetch from next block and store in ADDR_BLOCK_COLORS)
la   $t0, ADDR_BLOCK_COLORS # load color address
lw   $t1, 256($s1)
sw   $t1,   0($t0) # load color from next bot and save at current bot
lw   $t2, 128($s1)
sw   $t2,   4($t0) # load color from next mid and save at current mid
lw   $t3,   0($s1)
sw   $t3,   8($t0) # load color from next top and save at current top

# generate and save random color at next block viewer
jal  get_random_gem_color
sw   $v0,    0($s1)        # save at next top
jal  get_random_gem_color
sw   $v0,  128($s1)        # save at next mid
jal  get_random_gem_color
sw   $v0,  256($s1)        # save at next bot
```

Figure 21: The code that fetches color from the next block cell.

- 7th EASY feature: Easy(12), implement save



```
# TOP GEM
# 1) get saved bottom gem color
li $a0, 13 # x of saved
li $a1, 12 # y of saved
jal get_from_board # the color is saved in $v0
# 2) get gem color + apply saved color to gem color
lw $t1, 8($s3)
sw $v0, 8($s3)
# 3) apply the current color to saved location
li   $a0, 13    # x of saved
li   $a1, 12    # y of saved
move   $a2, $t1   # bottom gem color
jal add_to_board

# if the swapped color is black - do new_block
lw $t0, 0($s3)
bne $t0, $zero, reset_block_location
jal new_block
```

Figure 22: Code swaps the columns in the SAVE (middle bottom gray box) with the column that is active.

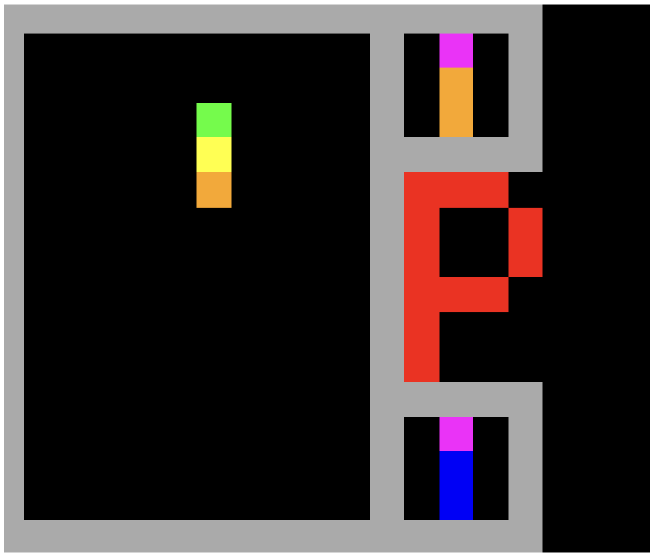


Figure 23: How it looks on the display.