Project Demo 2

Chloe Lam and Noelle Ransom

December 5, 2022

Configuration

As stated in the header of breakout.asm, our configuration is as follows:

- Unit width in pixels: 8
- Unit height in pixels: 8
- Display width in pixels: 256
- Display height in pixels: 256
- Base Address for Display: 0x10008000 ($gp)

Memory Plan

Our registers are used as follows:

- $t0: colours
- $t1: the display address
- $t2: the ball's location in the x direction
- $t3: the ball's location in the y direction
- $t4: the ball's address
- $t5: the ball's speed in the x direction
- $t6: the ball's speed in the y direction (may also be used as an iterator for drawing objects during initialization)
- $t7: used to compare two different values (comparator)
- $t8: an iterator used when drawing objects and the controller for the ball speed
- $t9: an iterator used to control the display's visible frame rate
- $s0 - $s5: temporary memory used in functions
- $s6: paddle 1's left-most pixel location
- $s7: paddle 2's left-most pixel location
- $a1: the number of lives the player has

Memory in Code

```
# Next, draw the paddle in the center bottom of the screen
li $a0, paddle_x
li $a1, paddle_y
jal get_location_address

addi $a0, $v0, 0            # Put return value in $a0

addi $s6, $a0, 0                # store the location of the left most pixel of the paddle

# Now, draw a second paddle in the center bottom of the screen, below the previous paddle
li $a0, paddle_x
li $a1, paddle_y
addi $a1, $a1, 3
jal get_location_address

addi $a0, $v0, 0            # Put return value in $a0

addi $s7, $a0, 0                # store the location of the left most pixel of the paddle
```

Figure 1: Usage of $s6 and $s7 registers.

```
# Initialize ball
la $t1, ADDR_DSPL  # t1 stores base address for display
lw $t1, 0($t1)
la $t0, MY_COLOURS  # t0 stores colour
lw $t0, 0($t0)

lw $t2, ball_x  # t2 stores the x position of the ball
lw $t3, ball_y  # t3 stores the y position of the ball
lw $t4, DISPLAY_WIDTH  # t4 stores the amount of space between rows in display
mult $t3, $t4  # multiply t3 and t4 to get the updated y position
mflo $t3  # t3 now contains the updated y position

add $t4, $t2, $t3  # t4 now contains t2 + t3
add $t4, $t4, $t1  # add the base address to t4

lw $t5, ball_speed_x  # t5 stores the speed of the ball on the x axis
lw $t6, ball_speed_y  # t6 stores the speed of the ball on the y axis

sw $t0, 0($t4) # paint the ball

li $t9, 0  # t9 is a counter that controls ball speed
```

Figure 2: Usage of $t0 to $t6 registers and $t9 register.

```
# The following series of functions draws all three walls used in the game.
prepare_top_wall:
    # know where to write to the display
    la $t1, ADDR_DSPL           # temp = &ADDR_DSPL
    lw $t1, 0($t1)              # display = *temp
    # getting the colour we want to draw with
    la $t0, MY_COLOURS          # temp = &MY_COLOURS
    lw $t0, 0($t0)              # colour = temp[0]

    # set line length to 256 pixels wide. 32 units wide, where each unit is 8 pixels wide.
    li $t9, 32                  # UNIT_COUNT = 32

    # Now let's iterate 32 times, drawing each unit in the line
    li $t8, 0                   # i = 0
draw_top_wall_loop:
    slt $t7, $t8, $t9           # i < UNIT_COUNT ?
    beq $t7, $0, prepare_vertical_walls # if not, then done
        # BODY
        sw $t0, 0($t1)          # Paint unit with colour
        addi $t1, $t1, 4        # Go to next unit

    addi $t8, $t8, 1            # i = i + 1
    b draw_top_wall_loop        # loop back to draw_top_wall_loop
```

Figure 3: Usage of $t7 to $t9 registers as iterative registers during initialization.

Static Scene

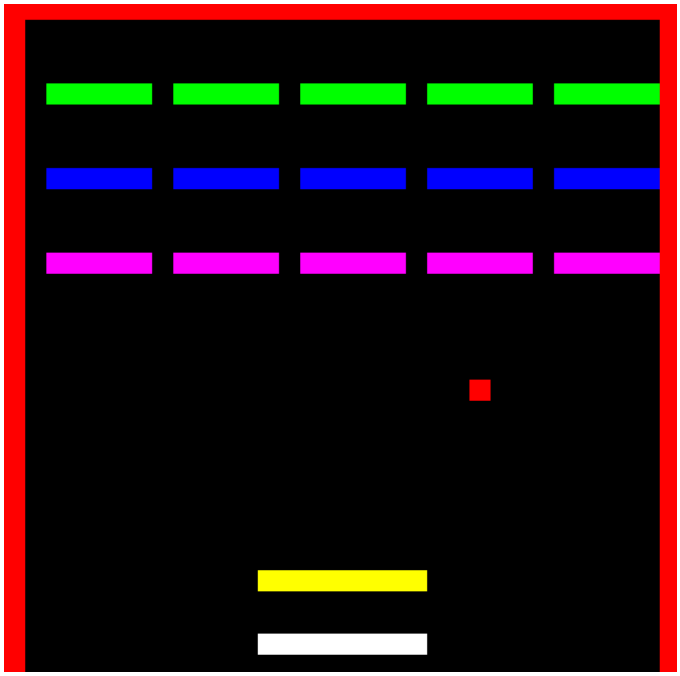

Figure 4: A screenshot of the static scene.

Handling Ball Collision

How the program will handle ball collision depends on the object the ball is colliding with:

- If the ball collides with a side wall, it will continue to move in the same y-direction, but it will move in the opposite direction along the x-axis. In other words, if the ball hits the wall going left, the ball will bounce off the wall going right. If the ball hits the wall going right, the ball will bounce off the wall going left.
- If the ball collides with the top wall, it will continue to move in the same x-direction, but it will move in the opposite direction along the y-axis. In other words, the ball will move downwards after colliding with the wall going upwards.
- If the ball collides with a brick, it will move as it does when colliding with the top wall.
- If the ball collides with the paddle, it will continue to move in the same x-direction, but it will move in the opposite direction along the y-axis. In other words, the ball will move upwards after colliding with the paddle going downwards.
- If the ball reaches the bottom of the screen, the game will immediately end.

How to Play

Players can control the top (yellow) paddle using the 'a' and 'd' keys:

- 'a' moves the paddle left
- 'd' moves the paddle right

Players can control the bottom (white) paddle using the 'j' and 'l' keys:

- 'j' moves the paddle left
- 'l' moves the paddle right

Once the player reaches a game-over state, players can reset the game by pressing the 'r' key.

A player can pause the game at any time by pressing the 'p' key.

A player can exit the game at any time by pressing the 'esc' key.

Game Features

- The player starts the game with 3 lives. The player loses a life whenever the ball reaches the bottom of the screen. The number of lives resets when the player resets the game after a game ends.
- The program enters a game-over state when the player runs out of lives. The player can reset the game by pressing the 'r' key on their keyboard, where no progress from the previous attempt is saved in the new game.
- The ball's speed increases every time the ball collides with a purple brick.
- The user can pause the game when the player presses the 'p' key on their keyboard.
- There are two paddles that are controlled by different keys. The top paddle is controlled by the 'a' and 'd' keys, and the bottom paddle is controlled by the 'j' and 'l' keys.
- Purple bricks require 3 hits to break. Blue bricks require 2 hits to break. Green bricks require 1 hit to break. Everytime the ball hits the brick, the colour changes depending on how many hits it still needs.