The 258 Project Report

Rudraksh Monga and Saleh Yasin

November 28, 2022

1 Milestone 1

In order to make our game of Breakout in Assembly in MIPS, we decided to use the "Arcade" configuration of Breakout. This means, we keep our display 256 pixels wide and 512 pixels high. Additionally, we keep one unit as an 8x8 block of pixels. By planning ahead, we calculated that we will need to keep two sections in the *.data* section of our program: immutable data types and mutable data types.

We decided to keep the addresses of the bitmap display and the keyboard as immutable data types. This was followed by a list of colours to be throughout the program. The order of the aforementioned list is: "Grey", "Black", "White, "Red", "Purple" and finally "Blue". This is shown in Figure 1.

We stored the relevant position data of the paddle and the ball as well as the vector for the ball's movement as mutable data types within the *.data* section of our program. We stored the (x, y) coordinates of the paddle, the current (x, y) coordinates of the ball in the current and previous frame, and a vector of the ball's movement as (x, y) coordinates.

```
.data
##############################################################################
# Immutable Data
##############################################################################
# The address of the bitmap display. Don't forget to connect it!
ADDR_DSPL:
    .word 0x10008000
# The address of the keyboard. Don't forget to connect it!
ADDR_KBRD:
    .word 0xffff0000
    #0x61 is "a"
    #0x64 is "d"
    #0x69 is "i"
    #0x71 is "q"

MY_COLOUR:
    .word 0xa3a3a3     # 0 grey color - wall
    .word 0x000000     # 4 black color - for redrawing/covering other drawn stuff, illusion of movement
    .word 0xffffff     # 8 white color - paddle
    .word 0xff0000     # 12 red color - brick layer 1
    .word 0xCC8899     # 16 purple color - brick layer 3
    .word 0x0000FF     # 20 blue color - brick layer 2
```

Figure 1: Immutable Data types

```
##############################################################################
# Mutable Data
##############################################################################

PADDLE_POSITION:
    .word 14           # 0 x_coordinate of paddle
    .word 14           # 4 previous x_coordiante of paddle

BALL_POSITION:
        .word 16       # 0 x_coordinate of ball
        .word 60       # 4 y_coordinate of ball
        .word 16       # 8 previous x_coordinate of ball
        .word 60       # 12 previous y_coordainte of ball

BALL_VECTOR:
        .word 0        # change in x_coordinate of ball
        .word 1        # change in y_coordinate of ball
```

Figure 2: Mutable Data types

After the memory layouts had been decided, we move forwards to drawing a static scene in the aforementioned Bitmap display. We choose to draw three grey walls: one for the top of the game screen, one for the left-hand side and one for the right-hand side. The paddle and ball were chosen to be white, with the paddle having a height of 1 unit and a width of 5 units and the ball having a high of 1 unit and a width of 1 unit. Each brick would be either Red, Purple or Blue, coupled with a height of 2 units and a width of 3 units. The resulting static scene is shown in Figure 3. There is some space left at the top of the screen for Milestone 4, where we plan to show the player their score.

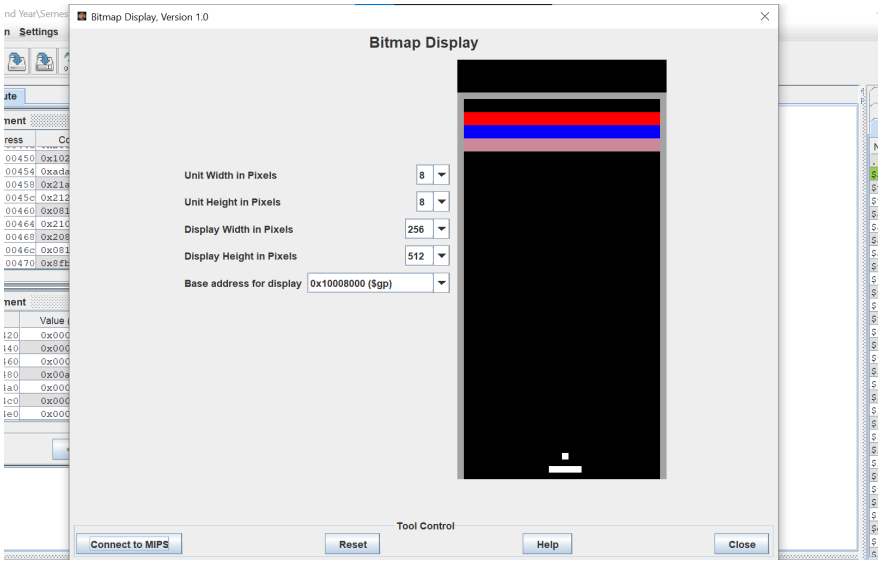

Figure 3: Static Scene

2 Milestone 2

After the static scene had been developed, we moved forward to Milestone 2: implementing player interaction. This meant, implementing paddle movement and having the ball move while the game loop refreshes the screen constantly. Ultimately, the screen refresh provided the illusion of movement. We also added an additional input functionality of allowing the player to press "q" on the keyboard to quit the game. The player would control the paddle by pressing "a" for left-hand movement and "d" for right-hand movement.

3 Milestone 3

After the player interaction was implemented, we started implementing the hardest part of the program's development: the program's reaction to the player's actions. This meant primarily implementing collision detection, followed by brick removal. For collision detection, we decided to make use of the *BALL_POSITION* data. This data set stored the ball's current position as well as the previous position. We split collision detection into three cases: Paddle Collision, Wall Collision and Brick Collision. All collisions made use of the ball's current position as well as its direction vector to determine where the ball would strike next. The collision only happened if the next position was NOT the black colour.

Paddle Collision occurred when the ball's next position would be striking a position with white colour. As such, this collision also made use of the paddle's current position. This allowed us to program different collision directions as to where the ball would hit the paddle. If it hit the paddle on the left-most edge, the ball moves diagonally towards the top-left. If it hit the paddle on the right-most edge, the ball moves diagonally towards the top-right. And if it hits the paddle anywhere in the middle, then the ball moves directly up.

Wall Collision occurred when the ball's next position would be striking a position with grey colour. This collision had five separate cases: Left Wall, Right Wall, Top Wall, Top-Right Corner, and Top-Left Corner. For Left Wall

and Right Wall, the ball vector's x-component would be reversed. For the Top Wall, the ball vector's y-component would be reversed. If the ball collided with the Top-Right Corner, then the ball would move towards the bottom-left. Finally, if the ball collided with the Top-Left Corner, then the ball would move towards the bottom right.

Colour Collision occurred when the ball's next position would be striking a position that was neither a white nor black colour. This collision took into account three different cases: Top Collision, Side Collision, and Bottom Collision. For Top Collision and Bottom Collision, the ball vector's y-axis component would change direction. These cases take place only when a ball strikes a brick above or below it. The third case, Side Collision, takes place when the ball strikes a brick on either side of it. During this, the ball vector's x-component would change direction.