Project Demonstration 2: Milestones 4-5

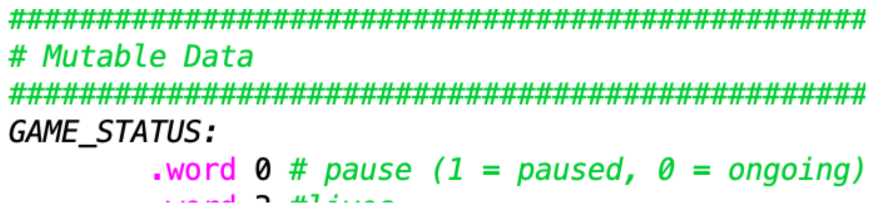
Kevin Trung Le 1007952805

Chin Chin Jim 1007935424

December 6, 2022

1 FEATURE 1: (EASY) Support "multiple lives" (at least 3)

Decide on how you will configure your bitmap display. To support multiple lives, we created a new data array called GAME_STATUS. With this, we defined a variable/register which we can name as remaining lives, preset as 3. Thus, we re-modify the logic where if it goes out of bounds, the game resets and the remaining lives variable gets mutated by subtracting it by 1. From this, until the variable is equal to 1, then the game will end.

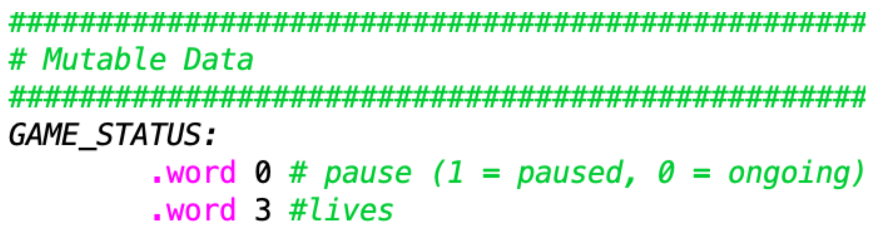


```
#################################################
# Mutable Data
#################################################
GAME_STATUS:
        .word 0 # pause (1 = paused, 0 = ongoing)
        .word 3 #lives
```

Figure 1: (Question 1) A screenshot of GAME_STATUS array representing the variable that stores remaining lives left in the game.

2 FEATURE 2: (EASY) Allow the user to pause the game p

Following on to FEATURE 1, under GAME_STATUS, we added a boolean variable such that the value is 0 if the game is ongoing and 1 if the game is paused. We initialized this variable as 0 to commence the game. From this there is a key listener function that responds when p is pressed such that if the current pause status is 0 then the variable will change to 1 and vice versa. If the game is paused (freezing the screen), then it will loop around the key listener until the user presses p again or presses q to quit the game.



```
##################################################
# Mutable Data
##################################################
GAME_STATUS:
        .word 0 # pause (1 = paused, 0 = ongoing)
        .word 3 #lives
```

Figure 2: (Feature 2) A screenshot of GAME_STATUS array representing the pause status of the game.

3 FEATURE 3: (EASY) Add 'unbreakable' bricks

The way we implemented unbreakable bricks was the redraw certain bricks in the middle of the screen as part of the draw function within the loop. This means that even if the ball collides with the brick, it will get reprinted thus unbreakable.

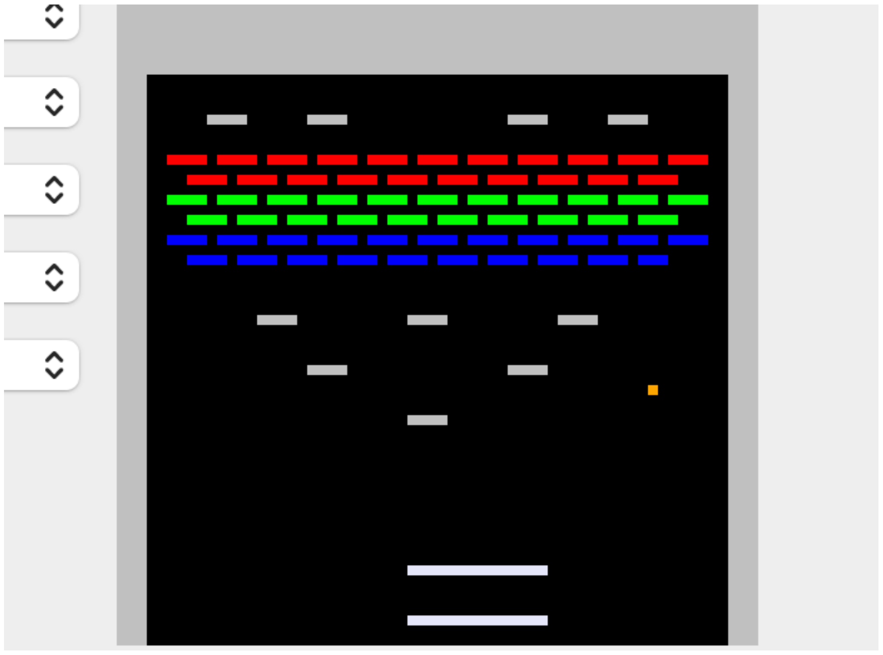


Figure 3: (Feature 3) A screenshot of the unbreakable "grey" bricks posed as obstacles in the game.

4 FEATURE 4 (EASY): Add a second paddle

For the second paddle we followed the controls j and l. We created another data mutable array for the second paddle and shifted its vertical location a 5 pixels below the other paddle. From this, the same logic follows as the first initial paddle but with different controls.

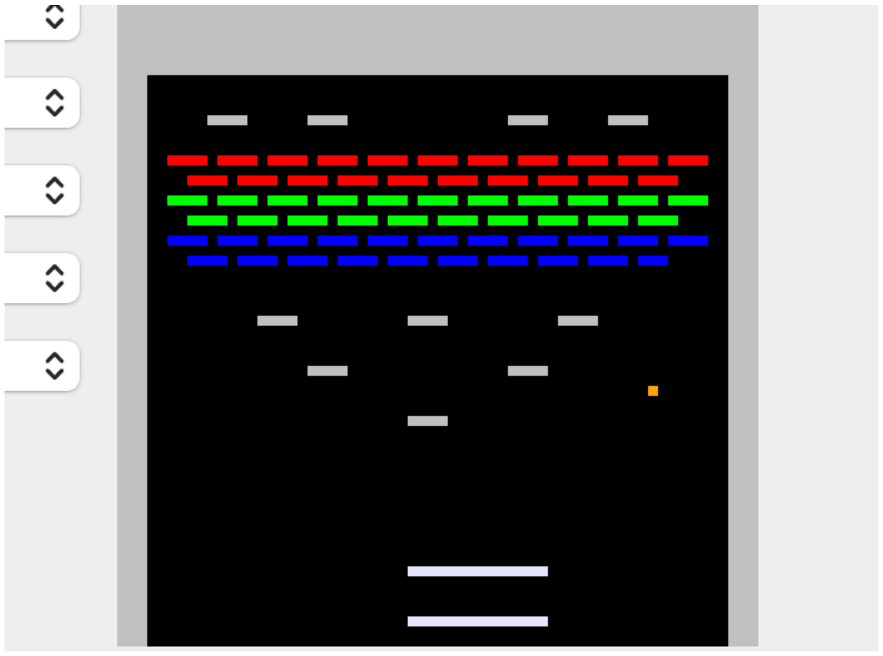


Figure 4: (Feature 4) A screenshot of FEATURE 4 where there are two paddles to control in the game.

5 FEATURE 5: (EASY) launch the ball at the beginning of each attempt

To implement this feature we added a new mutable variable within MY_BALL that represents a Boolean value is_launched that is initialize to 0. With this, the ball will follow the top paddle player one until the key b is pressed. Such that, in the function that responds to keys a and d, if is_launched is 0 (initially at the game) then the ball will move x coordinates just like the paddle. However, once b is pressed, is_launched is set to 1 and the ball begins to move by setting a direction and no longer responds to keys a and d. It is ensured that every single attempt, the ball gets reset back to the paddle where the launch can begin once again.

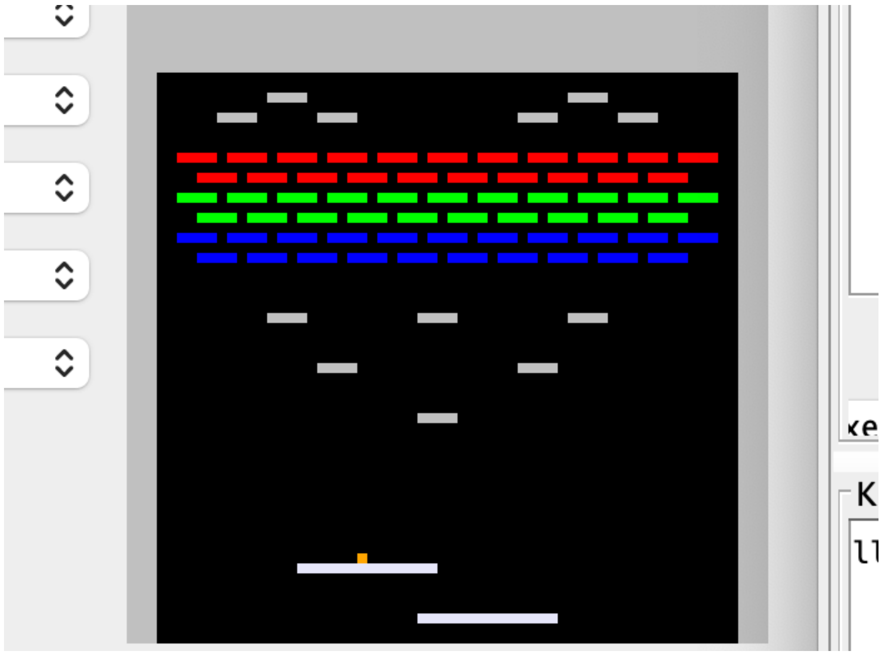


Figure 5: (Feature 5) A screenshot of FEATURE 5 where the ball is attached to the paddle until launch key.

6 FEATURE 6: (HARD) Require bricks be hit by the ball multiple times before breaking.

The general logic of implementing this feature is representing the status of breaking the brick with colours such that RED represents 3 more hits remaining to break, GREEN represents 2 more hits to break it and BLUE represents 1 more hit to break it. From this, at every single key frame, the ball checks if there is an obstacle top, left, right, or bottom. Following this, when it finds the obstacle (different colour), the general logic follows: keep on moving the pixel to the left unto the next left pixel is coloured black (getting starting address to draw a new coloured brick) and if the colour of the pixel obstacle is red then change to green, if it is green then change to blue, otherwise change to black.

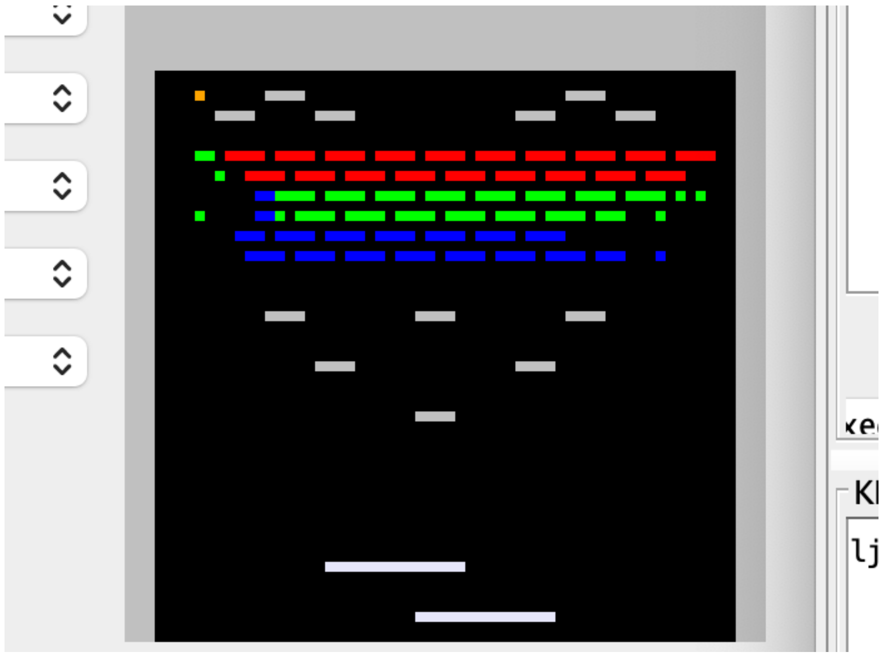


Figure 6: (FEATURE 6) A screen shot of FEATURE 6 where breaks are changing colours.