Assembly Project: Breakout

Aviraj Newatia

April 5, 2023

1 Instruction and Summary

1. Which milestones were implemented?
   - Milestone 1
   - Milestone 2
   - Milestone 3
   - Milestone 4:
     - Launching the ball (spacebar) and moving the initial location of the paddle and ball ('a' and 'd')
     - Pausing the game ('p') and Resuming (spacebar or 'p')
     - Ball Speed Increases on Hit
     - Restart game at any point, even after losing ('r')
   - Milestone 5:
     - Animated Brick Breaking and Colour Changing
     - Multiple-Hit Bricks
     - Sound Effects
2. How to view the game:
   (a) Height 128
   (b) Width 64
   (c) Height Unit 2
   (d) Width Unit 2
3. Game Summary:
   - My game is a rendition of breakout, the classic arcade game. The goal of the game is to break all of the blocks on the screen using the ball which is controlled by bouncing it off of a movable paddle. In this game there is a single level with 40 bricks to break, some of which require multiple hits to destroy. After the first hit on these multi-hit bricks, they turn grey. After the second hit they are destroyed. Whenever a brick is hit, a sound is played to signify the collision, and the colour change or destruction of the brick is animated. At the beginning of the game, you can move the initial location of the paddle and the ball and then press space to launch/start the game. At any point, you can press 'p' to pause the game and then spacebar to resume it. You can also press 'r' at any point to restart the game or press 'q' to quit. If the ball falls below the paddle, the game ends, at which point you can press 'r' to restart or 'q' to quit. If you break all of the bricks, you win the game and can press 'r' to restart or 'q' to quit.

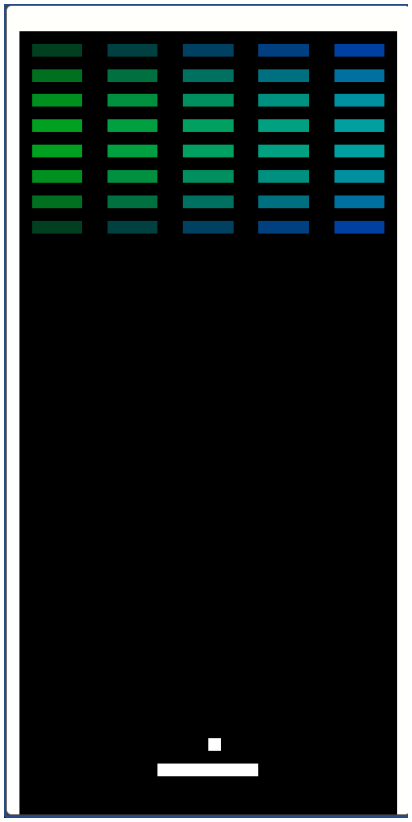

Figure 1: The initial screen for Breakout.

2 Attribution Table

Student 1 (Aviraj Newatia 1007837708)
Everything, my nonexistent partner didn't help at all

My game functions with 4 main game loops, 'game_loop' which is enacted when the game first starts allowing the player to move the initial location of the paddle and ball and launch the ball to start the game. 'paused_game_loop' is reached when the player pauses the game, and allows the player to resume or exit the game from that state. The third game loop is the 'moving_game_loop' which is the main actual "playing" game loop in which the ball is moved and all regular playing controls are bound to the handle_keyboard function. The last game loops is the 'exit_loop' which is a game loop which is reached when the ball falls below the paddle. This game loop exists to allow the player to restart or quit the game. Each of these game loops needs its own keyboard handling function so that only certain controls work in each situation. I could have designed a single keyboard handling function and checked where I came from, but this allowed me to minimize memory usage by increasing the manual logic performed using temporary registers and more lines of code.

The game plays a sound through syscalls, and the bricks are broken using loops that move left and right to remove coloured pixels and set them to black. The deletion functions also contain logic to check if the brick should be destroyed, or if its 'durability' should be reduced by colouring it grey first.

The game functions handle collisions through a filter system, which essentially does a binary search on the current memory and register values to determine how the ball collision should be handled. This allowed me to reuse a lot of my code in place of some more logic, and made the collision handling quite pretty and concise, as well as efficient.

The animation of the brick breaking or recolouring is done by using a sleep syscall whenever a brick colouring is done, which makes the destruction happen one pixel at a time, creating a ripple animation from the point of impact.

There's a reset variables function which is called when the game is restarted, which resets all of the memory locations and the required registers to their values before jumping the control flow back to the beginning of main, so that main functions properly to redraw the screen after first drawing the whole screen black again.