

Part I – Programming

You will program a scaled down version of the game Breakout. For those of you who are unfamiliar with Breakout, it was invented in 1976. It was a variation of the game Pong that came out earlier from Atari. What you need to do is program an interface to accept keyboard input that will update the location of a paddle, which the ball bounces off of. You are given several constants which are specified in the file “constants.inc.”

Ball starts on the screen at position “STARTX,” and “STARTY.” The ball will always start from this position if it falls off the screen. It will fall off of the screen if it ever goes beyond the last row on the screen. Every time it falls off the screen, you need to reset the ball position to “STARTX,” and “STARTY.” The ball has a x velocity “STARTVELX,” and a y velocity “STARTVELY.” As the game progresses, the ball moves. To calculate the new position of the ball, simply add the x and y velocity to the current ball position to obtain the new column and row position of the ball. If the ball hits either side of the screen (left or right) it will bounce back in the opposite direction and continue to move in the opposite direction (reverse x velocity). The same applies if the ball hits the top of the screen, it will reverse direction. If the ball is about to hit the paddle at the bottom of the screen, then it will also reverse the y direction of the ball. As stated, the only time you worry about the ball not bouncing back will be if it’s the bottom of the screen. If the ball is about to hit the paddle then its y direction needs to change. Furthermore, if the paddle is also directly under or directly above the paddle, then the direction of the ball reverses. When the ball moves, you need to check to see if the ball is currently located in a position where a brick is. The bricks are what you are aiming for. Here is really where our version of breakout differs. For the sake of simplicity we are only going to worry about changing the y direction of the ball if it hits a block. So, if the ball is currently on top of a block then change the current y direction of the ball. So when it hits a brick, it immediately changes direction. The positions of the bricks are specified with the constants in the “constants.inc,” file and specify a large block of bricks in a rectangular fashion. When a brick is hit, that brick disappears

The paddle bar will be controlled by the user using the arrow keys and will of a length specified by the constant “PADDLELEN,” which can be found in the file “constants.inc.” The start position of the paddle is specified by constants “PADDLEX,” and “PADDLEY.”

You need to display these bricks to the screen as well as the ball and the paddle all the time and constantly update their position. If the right arrow key is pressed, then the paddle moves right. If the left arrow key is pressed, then the paddle moves left. If the down arrow is pressed, then paddle moves down. If the up arrow is pressed, then the arrow moves up. The paddle may not move beyond the left or right side of the screen. The paddle can move up and down and must stay between rows 13 and 23 (inclusive – row count starting from row 0). If there is no input for the movement of the paddle, then it doesn’t move and everything else progresses as always. This means that the ball still needs to be updated and newest position displayed to the screen.

If the number of bricks displayed on the screen is 0, then the program ends. If the user ever hits the escape key, then the program ends.

You are NOT allowed to use any procedures from “util.lib,” or any pre-made macros. You may NOT use the interrupts for writing the screen, you must do it MANUALLY.

You may write to the video segment (text mode) at segment 0B800h. This constant is located in the “constants.inc,” file. Every screen position in this segment is composed of 2 bytes (1 word). The high part is the color attributes and the low part is the ASCII character to display.

Use the keyboard interrupts and OS interrupts to perform checking of the keyboard buffer and to perform processing on any input if present or to continue program execution if no input is currently available. Once everything is done, update the screen and start the process all over again.

You will find skeleton code for the main loop on the lab web page. There is one stipulation, you must keep track of the screen in an array called back buffer so that you might learn how to manipulate an array. In general, many games actually use a back buffer to prevent flickering of the screen, but it isn’t needed for this in this case. As stated, you are using a back buffer so that you can learn how memory is organized. This means that there is one required function “swapbuffer” which will copy the back buffer to the screen.

NOTE: You may need to add slowdown code to prevent the ball from moving excessively fast, i.e. move the ball every so many iterations of the main loop.

Grade distribution:

Part 1	
Ball movement	20
User input	20
Swapbuffer	20
Following Instructions	10%
Comments	
Program Description	5%
Code Comments	5%
Misc./Efficiency	10%