ÇAĞRI YILDIZ 1901042630

Project 1 Report

Notes:

My assembly code is working version 2 but also working version 3 but not totally correct. That is the reason why i take the second input from user. If we change the jal detonate lines to jal detonate_loop program works on version 3 but its output doesnt print correctly after the 3. Second. Array inputs is taking one by one row*column times. Also in version 2 my detonate function doesnt checks the boundries so if my output different from yours this is the reason.

Assembly (.asm) Code:

Main:

- Reads user input for seconds, row, and column.
- Allocates memory for dynamic and zero arrays (temp array).
- Fills dynamic array with user input.
- Prints dynamic array.
- Allocates memory for a new array (zero_array).
- Fills zero_array with '0'.
- Prints zero_array.
- Initiates a detonation
- Deallocates memory.
- Program is terminated

FUNCTIONS:

fill_array :

- Takes user input character and fills the dynamic array.

print_array :

- Prints the dynamic array.

fill_zeros :

- Fills the zero_array with '0'.

arrayz_print :

- Prints the zero_array.

detonate :

- Detonates bombs in the array, updating adjacent cells.

Flow:

1. User inputs seconds, row, and column.
2. Dynamic array is filled and printed.
3. Zero array is filled and printed.
4. Detonation simulation occurs.
5. Final state of the grid is printed.

Input Example 1:

- Seconds: 3
- Row: 3
- Column: 3

PROGRAM OUTPUT

```
Enter the seconds please : 3
Enter the row please : 3
Enter the column please : 3
Enter a character: .Enter a character: .E
...
.0.
...
```

PLEASE ENTER INPUT ROW*COLUMN TIMES

AFTER 1 SECONDS

```
000
000
000
```

AFTER 2 SECONDS

```
0.0
...
0.0
```

AFTER 3 SECONDS

EXPECTED OUTPUT

```
...
.O.
...
```

it looks the same after the first second. After the second second, Bomberman has placed all his charges:

```
OOO
OOO
OOO
```

At the third second, the bomb in the middle blows up, emptying all surrounding cells:

```
O.O
...
O.O
```

Input Example 2:

- Seconds: 3
- Row: 6
- Column: 7

PROGRAM OUTPUT

```
Enter the seconds please : 3
Enter the row please : 6
Enter the column please : 7
Enter a character: .Enter a character: .Enter a character: .Enter a character: .En
```

```
.......
...0...
....0..
.......
00.....
00.....

0000000
0000000
0000000
0000000
0000000
0000000

000.000
00...00
000...0
..00.0.
...000.
...0000
```

EXPECTED OUTPUT

```
.......
...O...
....O..
.......
OO.....
OO.....
```

Bomberman plants bombs in all the empty cells during his second second, so this is the state after 2 seconds:

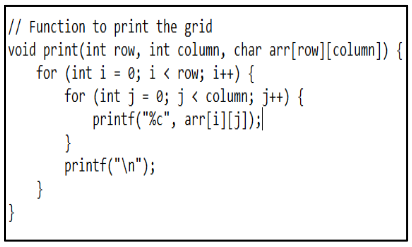
```
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
```

In his third second, Bomberman sits back and watches all the bombs he planted 3 seconds ago detonate. This is the final state after **3** seconds:

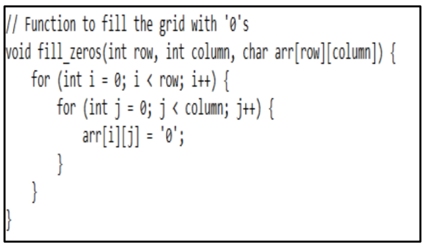
```
OOO.OOO
OO...OO
OOO...O
..OO.OO
...OOOO
...OOOO
```

As you see, my output is a little bit different because my detonate function doesnt checks the boundries

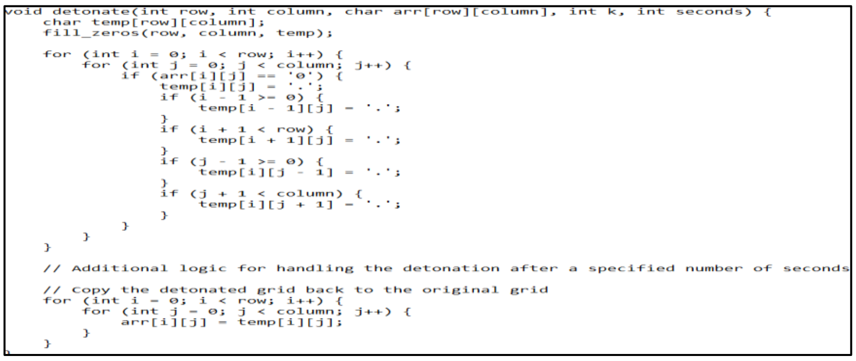
C (.c) Code:



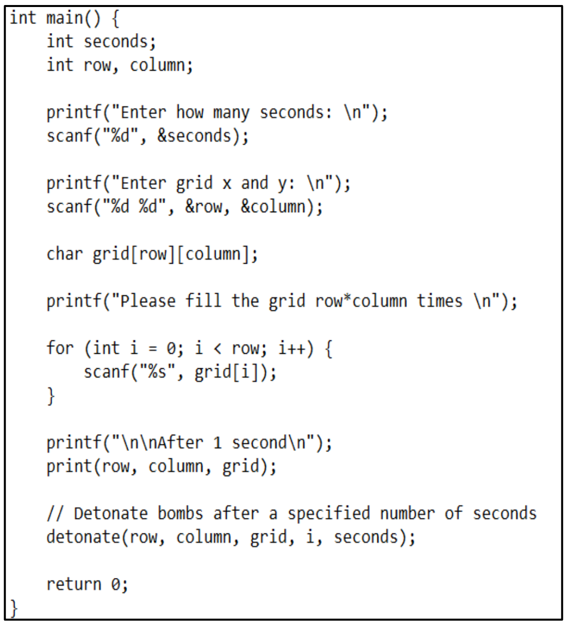
```c
// Function to print the grid
void print(int row, int column, char arr[row][column]) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            printf("%c", arr[i][j]);
        }
        printf("\n");
    }
}
```



```c
// Function to fill the grid with '0's
void fill_zeros(int row, int column, char arr[row][column]) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            arr[i][j] = '0';
        }
    }
}
```



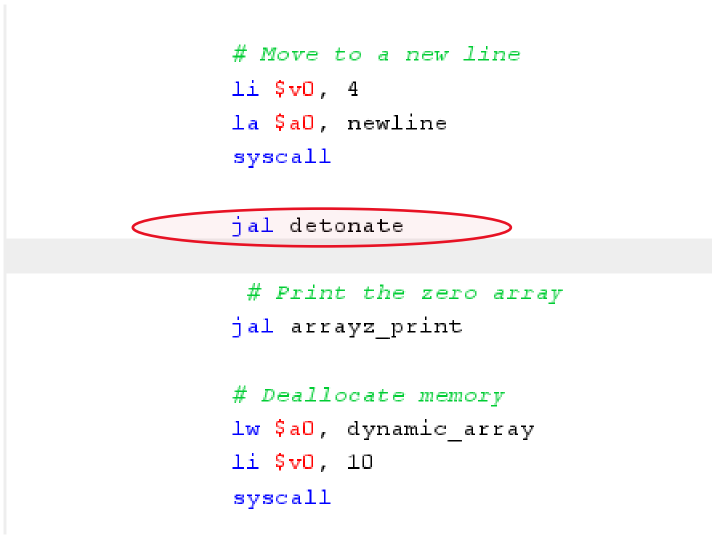
```c
void detonate(int row, int column, char arr[row][column], int k, int seconds) {
    char temp[row][column];
    fill_zeros(row, column, temp);

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            if (arr[i][j] == '0') {
                temp[i][j] = '.';
                if (i - 1 >= 0) {
                    temp[i - 1][j] = '.';
                }
                if (i + 1 < row) {
                    temp[i + 1][j] = '.';
                }
                if (j - 1 >= 0) {
                    temp[i][j - 1] = '.';
                }
                if (j + 1 < column) {
                    temp[i][j + 1] = '.';
                }
            }
        }
    }

    // Additional logic for handling the detonation after a specified number of seconds

    // Copy the detonated grid back to the original grid
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            arr[i][j] = temp[i][j];
        }
    }
```



```c
int main() {
    int seconds;
    int row, column;

    printf("Enter how many seconds: \n");
    scanf("%d", &seconds);

    printf("Enter grid x and y: \n");
    scanf("%d %d", &row, &column);

    char grid[row][column];

    printf("Please fill the grid row*column times \n");

    for (int i = 0; i < row; i++) {
        scanf("%s", grid[i]);
    }

    printf("\n\nAfter 1 second\n");
    print(row, column, grid);

    // Detonate bombs after a specified number of seconds
    detonate(row, column, grid, i, seconds);

    return 0;
}
```

Conclusion:

My assembly code is working well in version 2 but i wanted to make version 3 but i couldnt do the detonate loop part. If we change the **jal detonate** to **detonate_loop** code is working on version 3 but as i said it gives correct output for 3 seconds but after that not giving correct outputs.



```
# Move to a new line
li $v0, 4
la $a0, newline
syscall

jal detonate

 # Print the zero array
jal arrayz_print

# Deallocate memory
lw $a0, dynamic_array
li $v0, 10
syscall
```