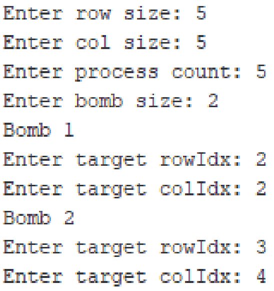
Description

Problem description is [The Bomberman Game | HackerRank]

How the program works

When run the code you must enter these inputs



```
Enter row size: 5
Enter col size: 5
Enter process count: 5
Enter bomb size: 2
Bomb 1
Enter target rowIdx: 2
Enter target colIdx: 2
Bomb 2
Enter target rowIdx: 3
Enter target colIdx: 4
```

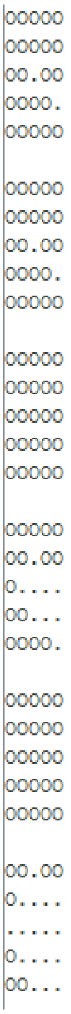
Process count indicates how many seconds after the initial state the program will terminate.

First target rowIdx, coldIdx are related with **Bomb1**,

Second target rowIdx, colIdx are related with **Bomb2**,

And so on,

Output of above inputs



```
OOOOO
OOOOO
OO.OO
OOOO.
OOOOO
```

→ Initial State (‘.’ Characters are initial bombs)

```
OOOOO
OOOOO
OO.OO
OOOO.
OOOOO
```

→ The state after 1 second

```
OOOOO
OOOOO
OOOOO
OOOOO
OOOOO
```

→ The state after 2 second

```
OOOOO
OO.OO
O....
OO...
OOOO.
```

→ The state after 3 second (Previous bombs have exploded and bombs are being placed in empty places)

```
OOOOO
OOOOO
OOOOO
OOOOO
OOOOO
```

→ The state after 4 second

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

→ The state after 5 second

Process count was 5 that’s why program terminate 5 seconds after the initial state.

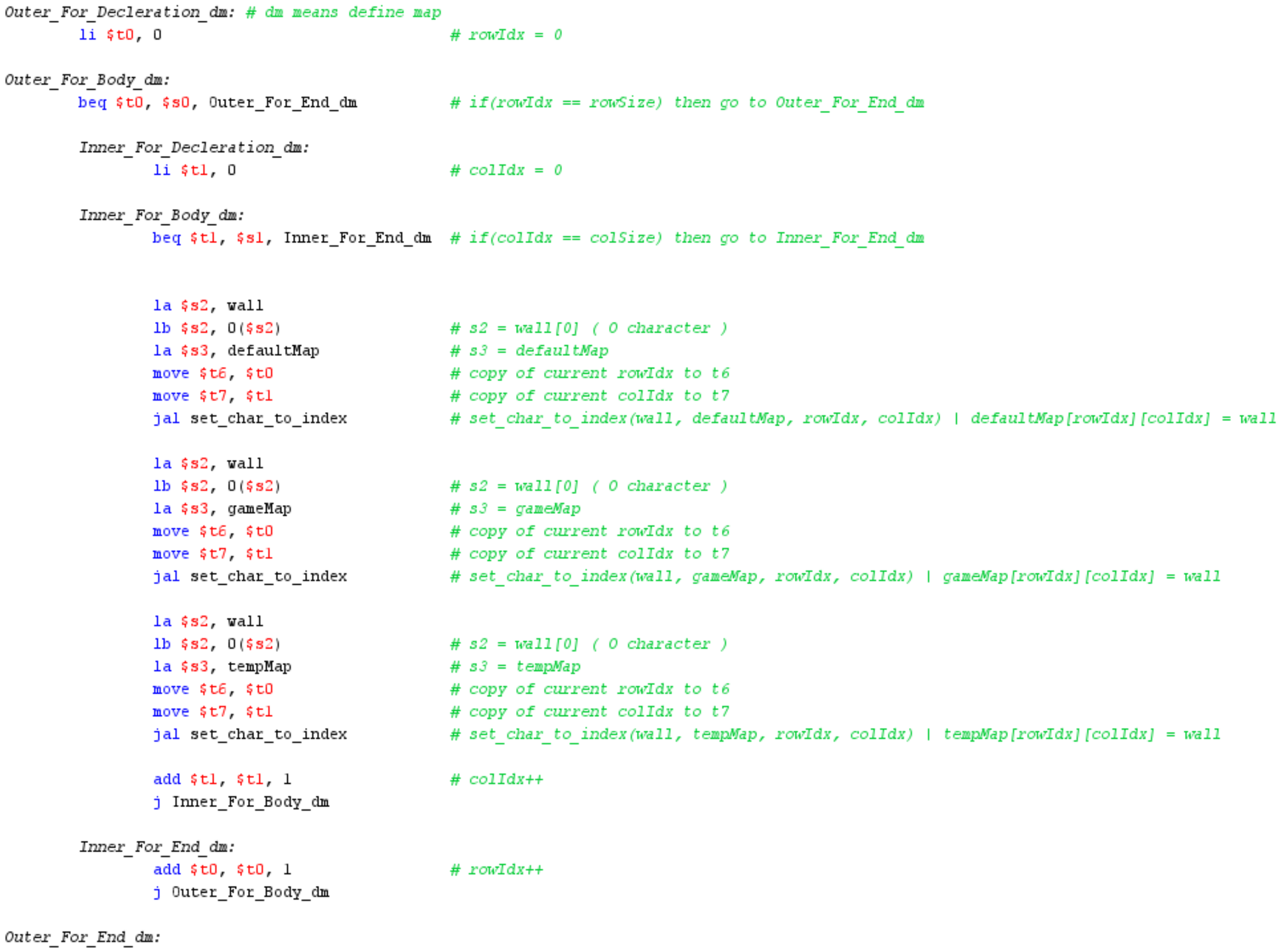
Constants

I have some constant registers and they does not change while program running.

Register	Usage
s0	It holds rowSize of matrices
s1	It holds colSize of matrices
t8	It holds process count, its value only decrement and it indicates the time remaining until the end of the program

Unused Labels

In the code I added a unused labels for to increase the understandability of the code for example:



```
Outer_For_Decleration_dm: # dm means define map
        li $t0, 0                               # rowIdx = 0

Outer_For_Body_dm:
        beq $t0, $s0, Outer_For_End_dm          # if(rowIdx == rowSize) then go to Outer_For_End_dm

        Inner_For_Decleration_dm:
                li $t1, 0                       # colIdx = 0

        Inner_For_Body_dm:
                beq $t1, $s1, Inner_For_End_dm  # if(colIdx == colSize) then go to Inner_For_End_dm


                la $s2, wall
                lb $s2, 0($s2)                  # s2 = wall[0] ( 0 character )
                la $s3, defaultMap              # s3 = defaultMap
                move $t6, $t0                   # copy of current rowIdx to t6
                move $t7, $t1                   # copy of current colIdx to t7
                jal set_char_to_index           # set_char_to_index(wall, defaultMap, rowIdx, colIdx) | defaultMap[rowIdx][colIdx] = wall

                la $s2, wall
                lb $s2, 0($s2)                  # s2 = wall[0] ( 0 character )
                la $s3, gameMap                 # s3 = gameMap
                move $t6, $t0                   # copy of current rowIdx to t6
                move $t7, $t1                   # copy of current colIdx to t7
                jal set_char_to_index           # set_char_to_index(wall, gameMap, rowIdx, colIdx) | gameMap[rowIdx][colIdx] = wall

                la $s2, wall
                lb $s2, 0($s2)                  # s2 = wall[0] ( 0 character )
                la $s3, tempMap                 # s3 = tempMap
                move $t6, $t0                   # copy of current rowIdx to t6
                move $t7, $t1                   # copy of current colIdx to t7
                jal set_char_to_index           # set_char_to_index(wall, tempMap, rowIdx, colIdx) | tempMap[rowIdx][colIdx] = wall

                add $t1, $t1, 1                 # colIdx++
                j Inner_For_Body_dm

        Inner_For_End_dm:
                add $t0, $t0, 1                 # rowIdx++
                j Outer_For_Body_dm

Outer_For_End_dm:
```

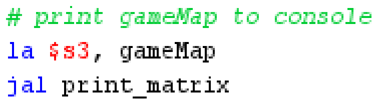
Some labels does not be jumped here. However, since I created a nested for structure, I created labels to indicate the parts of a normal for, and this increases readability. For example program never jump to ***Inner_For_Declaration_dm*** label but it indicates that part is declaration part of inner for loop.

Subroutines

I have 4 subroutine, names as “copy_to_matrix”, “set_char_to_index”, “get_char_from_index”, “print_matrix”. Each subroutine works like C function they have parameters and return values. I provide these with fill the registers before call them.

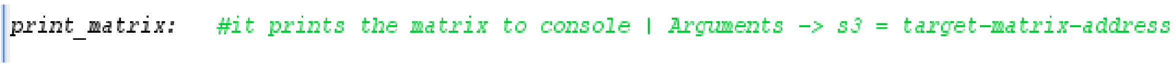
For example:

Print-matrix subroutine prints the matrix at the address held in the s3 register to the screen. And for use it we must fill s3 with adress of matrix which we want to print.



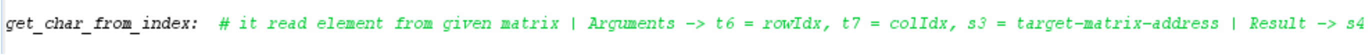
```
# print gameMap to console
la $s3, gameMap
jal print_matrix
```

I wrote as comments which arguments the subroutines need and what it returns. For example:



```
print_matrix:   #it prints the matrix to console | Arguments -> s3 = target-matrix-address
```

Or



```
get_char_from_index:  # it read element from given matrix | Arguments -> t6 = rowIdx, t7 = colIdx, s3 = target-matrix-address | Result -> s4
```

But for convenience, I will explain it here as well.

Subroutine Name	Arguments	Returns	Description

copy_to_matrix	s5 = sourceMatrixAddress
	s7 = targetMatrixAddress
		-	It copies source matrix to target matrix

set_char_to_index	t6 = targetRowIdx
	t7 = targetColIdx
	s2 = character to set
	s3 = targetMatrixAddress
		-	It writes given char (s5) to matrix

get_char_from_index	t6 = targetRowIdx
	t7 = targetColIdx
	s3 = targetMatrixAddress
		s4 = char of given index	it read element from given matrix

print_matrix	s3 = targetMatrixAddress	-	it prints the matrix to console