Data types of the program

```
            .data
SizeStr:    .asciiz "Please enter size of map: "
StepStr:    .asciiz "Please enter number of seconds: "
InputStr:   .asciiz "Please enter input of map:\n"
NewLine:    .asciiz "\n"
```

The program first starts by taking the map size and number of steps from the user.

```
Please enter size of map: 16
Please enter number of seconds: 3
Please enter input of map:
...............000000000000000.................0.0..........0....................................0....................................0....................................0.........................................0.
```

Then, if the number of steps is not an invalid number, it allocates space up to map size to hold the map (char) and size*4 to control the explosion times (int) of the bombs. It uses s2 registers to hold the char address and s3 registers to hold the int address.

It takes the map input from the user and adds it to the memory at address s2.

According to the user's input, the memory at address s3 sets the places where there are bombs to 1 and the places that do not have bombs to 0.

The main loop in which the program will flow begins.

```
MainLoop:       beq $s3, $s1, Exit # If counter of step equals to step input from user, finish the program
                addi $s3, $s3, 1 # Increases counter of step
```

If the number of steps is the same as the number of steps entered by the user, terminate the program. If not, increase the step counter by 1.

Go to SecondStep.

SecondStep examines the characters on the map one by one. If the current character is '.' it goes to SecondStepCond.

Prints 'O' to the current address in SecondStepCond. So '.' replaces with 'O' and sets the bomb's explosion time to 1.

Increases the loop index by 1.

Increments the string address by 1 to move to the next character.

Increments the int address by 4 (1 int = 4 bytes) to change the next character to int.

It checks the number of steps when the loop ends

If the number of steps is the same as the number of steps entered by the user, terminate the program. If not, increase the step counter by 1.

Go to ThirdStep.

```
SecondStepInnerLoop:        beq $t1, $s0, SecondStepOuterLoop # If inner loop finishes, go to beginning of the outer loop
                            lb $t4, 0($t2) # Get char from string array's current address
                            beq $t4, 46, SecondStepCond # If char is equal to '.'
                            add $t1, $t1, 1 # Increases inner loop's index
                            addi $t3, $t3, 4 # Go to next int of array
                            addi $t2, $t2, 1  # Go to next char of array
                            j SecondStepInnerLoop # Go back to beginning of the inner loop

    SecondStepCond:                 li $t5, 79 # Load decimal number of 'O' to t5
                                    li $t6, 0  # Load zero to t6
                                    sb $t5, 0($t2) # Load char 'O' to string array's current address
                                    sw $t6, 0($t3) # Load 0 to int array's current address
                                    add $t1, $t1, 1 # Increases inner loop's index
                                    addi $t3, $t3, 4 # Go to next int of array
                                    addi $t2, $t2, 1  # Go to next char of array
                                    j SecondStepInnerLoop # Go back to beginning of the inner loop

ExitSecondStepLoop:                 beq $s3, $s1, Exit # If counter of step equals to step input from user, finish the program
                                    addi $s3, $s3, 1 # Increases counter of step
```

In the ThirdStep, the explosion of the bombs in their turn is controlled.

String and int addresses are taken in the Third Step, as in the Second Step.

Nested loop begins.

Go to ControlPlace with jal in InnerLoop.

In Control Place, the character at our current address is controlled.

If the int value at the location is 1, the bomb must explode, so it jumps to ControlPlaceSit2.

If not, it is checked whether the position is 'O'. If it is not 'O', return to InnerLoop with jr $ra.

If it is 'O' (There is a bomb but it cannot explode this time), go to ControlPlaceSit1. The int value of the location is set to 1. This bomb must also explode during the next round control. Return to InnerLoop with Jr $ra.

In ControlPlaceSit2, the current position is replaced by '.' (meaning it exploded). Then, the up, down, left and right sides are checked by going to the conditions written for each of them respectively.

```
ContPlaceCond1:     addi $t0, $t0, -1 # Decreases index i by one to control up char of the current location
                    beq $t0, -1, ContPlaceCond2 # If t0 == -1, go to next condition
                    sub $t8, $t2, $s0 # Current address - size, to go to the up char of the current location
                    sb $t5, 0($t8) # Load char '.' to string array's current address

ContPlaceCond2:     addi $t0, $t0, 1 # Reset index i to previous state.

                    addi $t0, $t0, 1 # Increases index i by one to control down char of the current location
                    beq $t0, $s0, ContPlaceCond3 # If t0 == s0(Outside the boundaries of the map), go to next condition
                    add $t8, $t2, $s0 # Current address + size, to go to the down char of the current location
                    sb $t5, 0($t8) # Load char '.' to string array's current address

ContPlaceCond3:     addi $t0, $t0, -1 # Reset index i to previous state.

                    addi $t1, $t1, -1  # Decreases index j by one to control left char of the current location
                    beq $t1, -1, ContPlaceCond4 # If t1 == -1, go to next condition
                    add $t8, $t2, -1 # Current address - 1, to go to the left char of the current location
                    sb $t5, 0($t8) # Load char '.' to string array's current address

ContPlaceCond4:     addi $t1, $t1, 1 # Reset index j to previous state.

                    addi $t1, $t1, 1 # Increases index j by one to control right char of the current location
                    beq $t1, $s0, ExitContPlace # If t1 == s0(Outside the boundaries of the map), go to next condition
                    add $t8, $t2, 1  # Current address + 1, to go to the right char of current location
                    sb $t5, 0($t8) # Load char '.' to string array's current address

ExitContPlace:      addi $t1, $t1, -1 # Reset index j to previous state.
                    jr $ra # Jump back to ra
```

To give an example for ContPlaceCond1. First of all, it checks whether the index is out of bounds by decreasing the index (current row) by 1.

If it goes out of bounds, it passes to the next condition (ContPlaceCond2).

If not, it saves the address of the above character to t8. It puts the '.' character in t5 into the address in t8.

Each condition here checks its own indexes and leaves the location on the map as '0' or changes it to '.'.

Finally, it goes to ExitContPlace, restores index j and returns to $ra register.

The step counter goes to Exit when it reaches user input.

In Exit, go to the MapPrint subroutine with jal.

```
MapPrint:                   # Print Map Loop
                            move $t0, $s2 # Load address of string map array to t0 register
                            add $t1, $zero, $zero # Outside loop index -> t1, initialized as 0
        MapPrintLoop:       beq $t1, $s0, ExitMapPrintLoop # If outer loop finishes, exit the loop
                            la $a0, NewLine # Get address of NewLine to a0 register
                            li $v0, 4 # Load 4 to v0 register to determine program will print string
                            syscall # Call syscall to terminate v0's value and print string
                            add $t3, $zero, $zero # Inside loop index -> t3, initialized as 0
                            addi $t1, $t1, 1 # Increases outer loop's index

        MapPrintInsideLoop: beq $t3, $s0, MapPrintLoop # If inner loop finishes, go to beginning of the outer loop
                            lb $a0, 0($t0) # Load char from current address
                            li $v0, 11 # vo = 11 to print char
                            syscall # Call syscall

                            addi $t3, $t3, 1 # Increases inner loop's index
                            addi $t0, $t0, 1 # Go to next char of array
                            j MapPrintInsideLoop # Go to beginning of the outer loop

ExitMapPrintLoop:           jr $ra # Jump back to ra
```

Print the string address as the desired matrix with the nested loop.

Moves to a new line with NewLine.

For InnerLoop, a number with index 0 is created at t3.

OuterLoop index increases by 1.

Go to InnerLoop.

The character at the current address is retrieved and printed on the screen.

InnerLoop index increases by 1.

1 is added to the address to move to the next character.

Return to the beginning of the InnerLoop with J.

When the loop ends, go to the bottom line of the Exit label with j $ra.

The program ends with the ExitProgram label.