Bomberman - Version 3

Variables:

- Strings : to print info lines
- int Row : holds row size
- int Col : holds column size
- int Step : holds how many steps to go through
- char Grid[row][col] : main grid
- char MemGrid[row][col] : memory grid to hold previously planted bombs

Constants:

- DATA_SIZE : holds size of a char to make dynamic allocation

Functions:

- main : main function
- takeInput : takes initial info from the user which needed to start the game
- createGrid : creates the main grid dynamically and fills it with user input
- printGrid : prints the current main grid
- firstStep : first step of the game, creates a memory grid which holds a copy of the main grid
- secondStep : second step of the game, bomberman fills all the cells with bombs
- thirdStep : third step of the game, bombs planted 3 sec ago explode, and destroy the bombs in neighboring cells, if any.

Flow of the code:

1. **takeInput** function is called, and needed data is taken from the user. The given input is saved in Row, Col, Step variables in the memory.
2. **createGrid** function is called, and the main grid dynamically allocated in the heap, then each line of the grid is taken from the user. The grid is filled with the given lines.
3. **firstStep** function is called, and the memory grid dynamically allocated in the heap. Main grid is copied cell by cell to the memory grid. This way previously planted bombs will be easily detected.
4. **printGrid** is called to print the initial state of the main grid.
5. '*Step_Loop*' is created to loop through the given step count.

    5.1. **secondStep** function is called to plant bombs everywhere in the main grid. By the time, 2 seconds has passed.
    5.2. **thirdStep** function is called to explode all the bombs that has planted 3 seconds ago. The exploding bombs will destroy all the neighboring located bombs.
    5.3. **firstStep** function is called to update the memory grid since the main grid has changed into a new state.
    5.4. Since the beginning, 5 seconds has passed. The loop will iterate 'till the end of the time the user wanted.

6. **printGrid** function is called after the last second to print the final state of the main grid.
7. After printing, the program will be terminated.

Function implementations:

- **takeInput**:

  ➔ Info line printed
  ➔ Value of '*Row*' variable is taken
  ➔ Info line printed
  ➔ Value of '*Col*' variable is taken
  ➔ Info line printed
  ➔ Value of '*Step*' variable is taken
  ➔ A new line printed for readability
  ➔ Function returned to **main**

- **createGrid**:

  ➔ Value of '*Row*' variable is loaded from the memory.
  ➔ Value of '*Col*' variable is loaded from the memory.
  ➔ Size of the main grid is calculated for heap allocation:

    - *total num of elements = Row * Col*
    - *total size = (total num of elements) * DATA_SIZE*

  ➔ Address of allocated memory is loaded to the '*Grid'* variable for further usage.
  ➔ A string buffer is created to get the grid line by line from the user:

    - *total size for a line = Col + 1 (EOF)*

  ➔ Address of the buffer is loaded to a register for further usage.
  ➔ A row counter is created.
  ➔ '*Row_Loop*' is opened to iterate through the rows of the grid.

    - Info line printed.
    - Each line for the grid is taken from the user.
    - A new line is printed for readability.
    - A column counter is created.
    - '*Col_Loop*' is opened to iterate through the columns of the grid.

      - Current index of the grid is calculated:

        - *index = (rowCounter * Col) + colCounter*
        - *addr in mem of curr ind = index + Grid*

      - Current char of the given line is loaded into the current index of the grid
      - Column counter is incremented.
      - Index of buffer is incremented.

    - Row counter is incremented.

  ➔ After all the cells is filled with user input, the function will return to **main**.

- **printGrid**:

➔ Value of '*Grid*' variable is loaded from the memory.
➔ Value of '*Row*' variable is loaded from the memory.
➔ Value of '*Col*' variable is loaded from the memory.
➔ A new line is printed for readability.
➔ A row counter is created.
➔ '*Row_Loop*' is opened to iterate through the rows of the grid.

- A column counter is created.
- '*Col_Loop*' is opened to iterate through the columns of the grid.

  - Current index of the grid is calculated:

    - *index = (rowCounter * Col) + colCounter*
    - *addr in mem of curr ind = index + Grid*

  - Char at the current index of the grid is printed.
  - A space is printed for readability.
  - Column counter is incremented.

- A new line is printed for readability.
- Row counter is incremented.

➔ After printing whole grid, the function will return to **main**.

- **firstStep**:

➔ Value of '*Grid*' variable is loaded from the memory.
➔ Value of '*Row*' variable is loaded from the memory.
➔ Value of '*Col*' variable is loaded from the memory.
➔ Size of the memory grid is calculated for heap allocation:

- *total num of elements = Row * Col*
- *total size = (total num of elements) * DATA_SIZE*

➔ Address of allocated memory is loaded to the '*MemGrid'* variable for further usage.
➔ A row counter is created.
➔ '*Row_Loop*' is opened to iterate through the rows of the grid.

- A column counter is created.
- '*Col_Loop*' is opened to iterate through the columns of the grid.

  - Current index of the grid is calculated:

    - *index = (rowCounter * Col) + colCounter*
    - *addr in mem of curr ind of main grid = index + Grid*
    - *addr in mem of curr ind of mem grid = index + MemGrid*

  - Char at the current index of the main grid is loaded to the current index of the memory grid.
  - Column counter is incremented.

- Row counter is incremented.

➔ After copying the main grid into the memory grid, the function will return to **main**.

◆ **secondStep**:

➔ Value of '*Grid*' variable is loaded from the memory.
➔ Value of '*Row*' variable is loaded from the memory.
➔ Value of '*Col*' variable is loaded from the memory.
➔ A row counter is created.
➔ '*Row_Loop*' is opened to iterate through the rows of the grid.

- A column counter is created.
- '*Col_Loop*' is opened to iterate through the columns of the grid.
    - Current index of the grid is calculated:
        - *index = (rowCounter * Col) + colCounter*
        - *addr in mem of curr ind = index + Grid*
    - Column counter is incremented.
    - Current cell is checked to detect if it is empty or has a bomb.
    - If it already has a bomb, next char of the grid will be checked.
    - If it is empty, a bomb will be planted.
- Row counter is incremented.

➔ After planting bomb every cell of the main grid, the function will return to **main**.

◆ **thirdStep**:

➔ Value of '*Grid*' variable is loaded from the memory.
➔ Value of '*MemGrid*' variable is loaded from the memory.
➔ Value of '*Row*' variable is loaded from the memory.
➔ Value of '*Col*' variable is loaded from the memory.
➔ A row counter is created.
➔ '*Row_Loop*' is opened to iterate through the rows of the grid.

- A column counter is created.
- '*Col_Loop*' is opened to iterate through the columns of the grid.
    - Current index of the grid is calculated:
        - *index = (rowCounter * Col) + colCounter*
        - *addr in mem of curr ind of main grid = index + Grid*
        - *addr in mem of curr ind of mem grid = index + MemGrid*
    - Current cell of the memory grid will be checked to detect if it has a bomb or empty.
    - If it is empty, it will jump to the '*Else*' label.

- If it has a bomb, an empty cell will be loaded to the current cell, and '*If*' labels will check neighboring cells:

    - Column counter will be checked if it is in the range of the grid.
    - If not, it will pass to the next '*If*' label.
    - If it is in the column range, neighboring cells will be cleaned:

        - *x_index_neg = curr ind of main grid – 1*
        - *x_index_pos = curr ind of main grid + 1*

    - Row counter will be checked if it is in the range of the grid.
    - If not, it will pass to the next '*If*' label.
    - If it is in the row range, neighboring cells will be cleaned:

        - *y_index_neg = (rowCounter – 1) * Col + colCounter*
        - *y_index_pos = (rowCounter + 1) * Col + colCounter*

    - In '*Else*' label, column counter is incremented.
- Row counter is incremented.

➔ After checking and cleaning each cell of the grid, the function will return to **main**.