Introduction to Programming
2nd Project

Programming Group
guido.salvaneschi@unisg.ch

University of St. Gallen – December 4th, 2025

Declaration of Authorship

"I hereby declare

- that I have solved this project without any help from others and without the use of documents and aids other than explicitly stated;
- that I have mentioned all the sources used and that I have cited them correctly according to established academic citation rules;
- that I have acquired any immaterial rights to materials I may have used such as images or graphs, or that I have produced such materials myself;
- that I will not pass on copies of this work to third parties or publish them without the University's written consent;
- that I am aware that my work can be electronically checked for plagiarism and that I hereby grant the University of St. Gallen copyright in accordance with the Examination Regulations in so far as this is required for administrative action;
- that I am aware that the University will prosecute any infringement of this declaration of authorship and, in particular, the employment of a ghostwriter, and that any such infringement may result in disciplinary and criminal consequences which may result in my expulsion from the University or my being stripped of my degree"

By submitting a solution to this project, you confirm through conclusive action that you are submitting the "Declaration of Authorship," that you have read and understood it, and that it is true.

Warning: You must work individually and without the help of any other person on this project. All submissions will be checked for plagiarism. If we suspect plagiarism, we may ask you for a clarification meeting and a personal code presentation. Found plagiarism leads to failing the entire exam.

Please follow HSG regulations on written work, including DDoS Section 5 `https://erlasse.unisg.ch/lexoverview-home/lex-II_B_1_20`.

Submission deadline: December 18, 2025, 10:00.

Task 1:

Implement the program specified below in a module file `gomoku.py` and submit it before the deadline to the GitHub classroom repository (link to the submission platform: `https://classroom.github.com/a/49oaOWlk`). Start from the provided Implementation Scaffold and implement the specified functionality in it. Add further functions to structure your code, but do not remove or rename the ones given in the scaffold. Make sure your program, i.e., its `main()` function, is only executed when the module is directly executed, not when it is imported in another Python program. Only submit your `gomoku.py` file. The provided `ui.py`, `test_gomoku.py`, `conftest.py` and `pytest.ini` files are available as provided on the submission platform and cannot be overwritten. Make sure your code is of high quality, i.e., it uses Python adequately, is well-structured and formatted, uses self-explaining identifier names, and contains reasonable comments that concisely document all non-trivial design decision. Your implementation will be assessed for functionality (50%), compliance (20%), and style (30%).

Info: We provide some basic tests in `test_gomoku.py`. You can use them to test your implementation locally. To do so, download the files `test_gomoku.py`, `conftest.py`, and `pytest.ini` to the directory with your `gomoku.py` and run `pytest` in that directory. The exactly same tests will be available and executed on the submission platform every time you submit. Make sure, these tests work also on your submission on GitHub to ensure compatibility. You can update your submission until the deadline. Only the final submission will be considered for grading.

Note that such tests only cover *basic* functionalities for your early feedback and are not indicative of the fact that the application fully implements the functionalities required for the project.

Gomoku

Gomoku is a two-player strategy board game traditionally played using black and white *Go* stones on the intersections of a square grid, typically fifteen rows by fifteen columns. [1]

In this assignment, however, we adopt the standard digital representation of the game: the board is modeled as a grid of square cells, and each stone occupies exactly one cell. Players place stones by selecting the row and column of an empty cell.

The game in this assignment begins with an empty board. The two players alternate turns, each placing one stone of their color on any unoccupied cell of the grid.

A player wins if they succeed in forming a consecutive line of **five** of their stones, horizontally, vertically, or diagonally. If the entire grid becomes filled without either player achieving such a five-stone line, the game ends in a draw.

Figure 1: Gomoku game illustration.[3]

Step-by-step Program Description

1. The game starts in the menu (see “Menu”).
   (i) All menu items are displayed to the users.
   (ii) The users are prompted to select a menu item by entering its number. On invalid input, the prompt is repeated until a valid item number is provided.
   (iii) The users have the following choices:
   - “Play standard game”: The program continues with step 3 and the next game is played with fifteen rows and fifteen columns, without ’remove’ option (i.e., the last-played stone(s) cannot be removed).
   - “Play game with adjustable size”: The program continues with step 2 and the next game is played without ’remove’.
   - “Play game with adjustable size and remove”: The program continues with step 2 and the next game is played with remove (i.e., the last-played stone can be removed).
   - “Scoreboard”: The program continues with step 6.
   - “Exit”: The program ends.
2. The users are asked to enter the grid size for the next game (see “Entering Grid Size”).
   (i) The users are prompted to enter the number of columns for the next game as a number from 10 to 20. On invalid input, the prompt is repeated until a valid number is provided.
   (ii) The users are prompted to enter the number of rows for the next game as a number from 10 to 20. On invalid input, the prompt is repeated until a valid number is provided.
   (iii) The program continues with step 3.
3. The users are asked to enter their names for the next game (see “Entering Names”).
   (i) The users are prompted to enter the name for player A for the next game. The name must have at least one character. On invalid input, the prompt is repeated until a valid name is provided.

[1] It’s worth mentioning that *Gomoku* is a very simplified version of the full game of Go, and the game of *Go* itself is historically significant in computer science: it was long considered one of the most challenging board games for artificial intelligence due to its enormous branching factor and strategic depth. This culminated in the development of *AlphaGo* by Google DeepMind.

[3] Image generated with Gemini Nano Banana Pro.

(ii) The users are prompted to enter the name for player B for the next game. The name must have at least one character and be different from the name of player A. On invalid input, the prompt is repeated until a valid name is provided.

(iii) The program continues with step 4 with an empty grid and player A.

4. A turn is played (see "Game Round").

(i) The turn header (text above the grid in every turn, see "Game Round") is displayed, announcing the name of the player whose turn it is and which color they have.

(ii) The current grid is displayed.

(iii) The current player is prompted to enter the *row* and the *column* in which they want to place a stone. Both numbers must refer to positions that exist in the current game's grid, and the selected square must be empty. If the game is played with remove, then entering -1 instead of row and column is also valid. On invalid input, the prompt is repeated until a valid position (or -1 in the remove mode) is provided.

(iv) If -1 was entered and the grid is not empty and remove is enabled, the stone that was last added is removed from the grid and the program continues with step 4 and with the other player. If -1 was entered and the grid is empty, the program just continues with step 4 and with the same player.

(v) If a valid row and column were entered, the stone is added to the grid at the selected position.

(vi) If a user won, i.e., there are five stones of the same color in one line next to each other, the program continues with step 5. If the grid is full, the program continues with step 5. Otherwise, the program continues with step 4 and with the other player.

5. The game result is shown (see "Game Result").

(i) If the board contains five stones with the same color next to each other vertically, horizontally or diagonally, a message is displayed that the respective user won. Otherwise, a message is displayed that the game was a draw.

(ii) If the game was not a draw, the winner's score in the scoreboard is increased by 1.

(iii) The users are prompted to press enter to return to the menu. On any input, the program continues with step 1.

6. The scoreboard is shown (see "Scoreboard").

(i) The scoreboard, a list of all player names with their score, is displayed to the users.

(ii) The users are prompted to press enter to return to the menu. On any input, the program continues with step 1.

Implementation Scaffold

Your implementation of the `gomoku.py` module must implement the following functions. Other modules have to be able to import them from your module file and use them as described below. You may add additional functions to your module to structure your code and all functions below definitely may call other functions to implement their functionality. As starting point, you can use the provided `gomoku.py` file, which defines all these functions as required without an implementation.

1. `main()` is the entry point into the game and runs its full implementation. Returns `None`.
2. `menu()` displays the menu to the user and repeatedly asks for the user's choice until a valid answer was provided. Returns the number of the selected menu item as integer.
3. `save_scoreboard(scoreboard)` saves the scoreboard provided in the parameter `scoreboard` to the file `scoreboard.dat`. If the file already exists, it is overwritten. Returns `None`.
4. `load_scoreboard()` loads the scoreboard from the file `scoreboard.dat`. Returns the scoreboard without entries where the score is 0. If the file does not exist, is not readable, or does not contain data in the expected format, an empty scoreboard (an empty dictionary) is returned.
5. `is_grid_full(grid)` receives the current grid state in parameter `grid`. The function then evaluates whether the grid is full. In particular, when the game is a draw, the grid is full. If the grid is full, the function returns `True`, otherwise `False`.
6. `is_game_won(grid)` receives the current grid state in parameter `grid`. This is the core function of the game logic. The function evaluates whether a player won. The function checks if there are five stones of the same color in one line next to each other, in all possible directions, as in the game rules. If a player won, the function returns `True`, otherwise `False`.
7. `play_turn(grid, player_name, is_player_a, can_remove)` receives the current grid state in parameter `grid`, the name of the player whose turn it is as string in `player_name`, whether the player is player A as boolean in `is_player_a`, and whether the game is played with remove as boolean value in the parameter `can_remove` (`True` means remove is enabled). The function first displays a turn start message with the player's name whose turn it is and their stone color. Then it displays the current grid and prompts the user to enter the *row* and the *column* of the cell in which they want to place a stone. If `can_remove` is `True`, the user may instead enter -1 to request removal of the last stone. The prompt is repeated until a valid value is provided. Finally, the function returns a pair of integers `(column_index, row_index)` giving the internal grid indices of the selected cell (see warning below). If `can_remove` is `True` and the user input was -1, the function returns `None`.
8. `play_standard_size_game()` first asks the users for their names. It then runs a game with a grid of fifteen rows and fifteen columns without remove until a player won or the game is a draw. It then shows the result of the game and prompts the user to press enter to return to the menu. On any input, the function returns the name of the player that won as string or `None` if the game is a draw.
9. `play_game()` first asks the users for the number of rows and columns the grid shall have in the next game. Then the function behaves exactly like `play_standard_size_game()`, but it executes a single game with the grid size defined by the users.
10. `play_game_with_remove()` behaves exactly like `play_game(rows, columns)`, but the single game it executes is played with the possibility to remove the last stone (i.e. with `can_remove = True`).

In all functions above, the *scoreboard* is a dictionary, where the keys are player names as strings and the values are their respective score as integer.

In all functions above, the *grid* is represented as a two-dimensional list. The outer list has one entry per column and therefore its length equals the number of columns of the game board. Each entry of the outer list is itself a list representing the rows of that column, ordered from bottom to top. Thus, the value `grid[c][r]` refers to the field in column `c` (counted from the left, starting at 0) and row `r` (counted from the bottom, starting at 0).

Each field of the grid contains one of the following values:

- `None`: the field is empty,
- `True`: the field contains a stone placed by player A,
- `False`: the field contains a stone placed by player B.

For example, on a standard-sized Gomoku board of 15×15, the valid indices are `grid[0..14][0..14]`. If `grid[0][0]` is `None`, the bottom-most cell of the left-most column is empty. If `grid[3][2]` is `True`, then player A has placed a stone in the fourth column from the left and the third row from the bottom.

Warning: Please be aware that the internal representation of the grid (row 0 = bottom) is different from the representation shown in the UI (row 1 = top). When converting between UI input and internal indices, you must take this inversion into account.

Example. On a `15 x 15` board, if the user enters:

```
Please enter row and column (e.g., '8 10'):1 3
```

they mean: "top row, third column". Internally, this corresponds to:

$$\texttt{row index} = 15 - 1 = 14, \qquad \texttt{column index} = 3 - 1 = 2.$$

Therefore, the value must be written into:

`grid[2][14]`.

Conversely, the bottom-most row in the UI (row 15) corresponds to internal row `0`.

UI Module

The provided `ui.py` module implements the user interaction of the game. It implements the following functions, which must be used in your Gomoku implementation for all user interaction. Besides the documentation below, looking at the provided code may clarify what each function does. The representation of scoreboard and grid is the same as described above in "Implementation Scaffold."

Warning: Use `ui.py` as provided and do not change it. On the submission platform, only the provided version will be available. You cannot replace it with your own, altered implementation.

- `display_grid(grid)` displays the grid as provided in the parameter `grid` Returns `None`.
- `display_headline(headline)` displays a headline in uppercase that is provided as string in `headline`. Returns `None`.
- `display_menu(items)` displays the menu screen. It receives the menu items as list of strings in `items` and displays the items in the order of occurrence in the list as enumeration, starting with the number 1. Returns `None`.
- `display_message(message)` displays a message that is provided as string in `message`. Returns `None`.
- `display_scoreboard(scoreboard)` displays the scoreboard received in the `scoreboard`. Returns `None`.
- `display_turn_start(player_name, is_player_a)` receives the name of the player as string in `player_name` and whether the player is player A as boolean in `is_player_a`. The function displays that a new turn starts with the name and stone color of the player who's turn it is. Returns `None`.

- `prompt(message)` displays a prompt with the string provided in `message` and waits for a line of user input on STDIN. Returns the string read from STDIN without trailing `\n` linefeed character.

To clear the screen, `display_headline` and `display_turn_start` print a number of empty lines at the beginning.

Implementation Constraints

The program has to adhere to the following constraints:

1. The program does only use the provided UI module for input and output, i.e., it does not print to STDOUT or read from STDIN (e.g., using `print` and `input`).
2. In one execution, the program reads at most one file at most once.
3. The program persists scoreboard updates as soon as possible such that no data is lost if the program is interrupted directly after the respective game function in the scaffold completed.
4. No other import than `import ui` and `import pickle` is used.
5. The game logic (asking for the grid size, asking for the names, deciding whether the game continues, removing turns, deciding who's turn it is, performing a turn, evaluating whether a player won, showing the result of the game, and the flow between these actions) is implemented at most once.

User Interactions

In the following, you find examples of interactions with the users in the "Step-by-step Program Description." They can and shall be implemented by solely using the provided "UI Module." Make sure that your implementation exactly resembles the interactions below.

Menu

Command Line

```
GOMOKU

1. Play standard game
2. Play game with adjustable size
3. Play game with adjustable size and remove
4. Scoreboard
5. Exit

Please enter its number to select an item:
```

An interaction with two invalid inputs followed by the selection of the scoreboard looks like:

Command Line

```
GOMOKU

1. Play standard game
2. Play game with adjustable size
3. Play game with adjustable size and remove
4. Scoreboard
5. Exit

Please enter its number to select an item: Invalid Value
Please enter its number to select an item: 0
Please enter its number to select an item: 4
```

Scoreboard

Command Line

```
GOMOKU SCOREBOARD

1. Justine (83)
2. Donald (53)
3. Lisa (48)
4. Marge (35)
5. Homer (20)
6. Mike (10)

Please press ENTER to return to the menu:
```

An empty scoreboard is represented by an empty dictionary results in:

Command Line

```
GOMOKU SCOREBOARD

no scores available

Please press ENTER to return to the menu:
```

Entering Grid Size

An interaction configuring twelve rows and fifteen columns looks like:

Command Line

```
CONFIGURE GRID

Please enter the number of columns (10..20): 15
Please enter the number of rows    (10..20): 12
```

Prompts are repeated until a valid input is provided. An interaction with invalid values eventually configuring twelve rows and fifteen columns looks like:

Command Line

```
CONFIGURE GRID

Please enter the number of columns (10..20): Hello World
Please enter the number of columns (10..20): 21
Please enter the number of columns (10..20): True
Please enter the number of columns (10..20): 15
Please enter the number of rows    (10..20):
Please enter the number of rows    (10..20): -12
Please enter the number of rows    (10..20): 12
```

Entering Names

An interaction configuring the player names Marge and Homer looks like:

Command Line

```
CONFIGURE NAMES

Please enter the name of player A: Marge
Please enter the name of player B: Homer
```

Prompts are repeated until a valid input is provided. An interaction with invalid values eventually configuring Marge and Homer looks like:

Command Line

```
CONFIGURE NAMES

Please enter the name of player A:
Please enter the name of player A: Marge
Please enter the name of player B: Marge
Please enter the name of player B:
Please enter the name of player B: Homer
```

Game Round

The ninth round of a game, using the standard grid, where Marge has to play and selects empty cell at row 12 and column 9, looks like:

Command Line

```
GOMOKU

Marge, it is your turn! Your stone color is X.

    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   | X | O | O |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   | O | X |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   | O |   |   |   |   |   |   |
10|   |   |   |   |   |   |   |   | X |   |   |   |   |   |   |
11|   |   |   |   |   |   |   |   | X |   |   |   |   |   |   |
12|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
13|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
15|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Please enter row and column (e.g., '8 10'): 12 9
```

On invalid input, the prompt is repeated. E.g., if it is Homer's turn and he provides three invalid inputs before selecting the empty cell at row 13 and column 9, the turn's interaction could look like:

Command Line

```
GOMOKU

Homer, it is your turn! Your stone color is O.

    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   | X | O | O |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   | O | X |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   | O |   |   |   |   |   |   |
10|   |   |   |   |   |   |   |   | X |   |   |   |   |   |   |
11|   |   |   |   |   |   |   |   | X |   |   |   |   |   |   |
12|   |   |   |   |   |   |   |   | X |   |   |   |   |   |   |
13|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
15|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Please enter row and column (e.g., '8 10'): 7 9
Please enter row and column (e.g., '8 10'): 0 0
Please enter row and column (e.g., '8 10'): 1
Please enter row and column (e.g., '8 10'): 13 nine
Please enter row and column (e.g., '8 10'): 13 9
```

Game Round with Remove

When the game is played with remove enabled, the first three rounds of a game looks like:

Command Line

```
GOMOKU

Homer, it is your turn! Your stone color is X.

    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
10|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
11|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
12|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
13|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
15|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Please enter row and column (e.g., '8 10'): -1
Please enter row and column (e.g., '8 10'): 4 2




GOMOKU

Marge, it is your turn! Your stone color is O.

    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4 |   | X |   |   |   |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
10|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
11|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
12|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
13|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
15|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Please enter row and column (e.g., '8 10'): 1 1
```

Command Line

```
GOMOKU

Homer, it is your turn! Your stone color is X.

    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1 | O |   |   |   |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4 |   | X |   |   |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
10|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
11|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
12|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
13|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
15|   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Please enter row and column (e.g., '8 10'): -1



GOMOKU

Marge, it is your turn! Your stone color is O.

    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4 |   | X |   |   |   |   |   |   |   |   |   |   |   |   |
5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
10|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
11|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
12|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
13|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
15|   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Please enter row and column (e.g., '8 10'):
```

Game Result

A result screen when a player with the name Marge won looks like:

Command Line

```
CONGRATULATIONS

Marge won the game!

Please press ENTER to return to the menu:
```

A result screen when the game was a draw looks like:

Command Line

```
OH NO - A DRAW

Unfortunately, nobody won the game :(

Please press ENTER to return to the menu:
```