Project Documentation

Introduction:

The project aims to implement a Tetris game with additional features like a scoring system and a scoreboard. The primary objective is to create an interactive and functional Tetris game with a focus on efficient memory usage and algorithms.

Methodology:

The implementation utilizes several classes to represent different elements of the game:

- Cell: Represents each cell on the game board, storing directions, cell value, and a lock status.

- Line: Represents a line in the board, consisting of linked list nodes made of cells. Provides methods for checking if it's full, popping, making it the head, and shrinking.

- Board: Represents the game board, made up of multiple lines. Provides methods for operations like shrinking lines, checking and popping full lines, deleting the top line, and getting the middle element of the top line.

- Tetrimo: Represents each piece in the game as a 4x4 array. Provides methods for changing array elements based on color and type, checking if a given position is valid, moving the piece, locking cells, deleting cells from the board, adding cells temporarily to the board, and rotating the piece.

- Next: Represents a queue of tetrimos that holds items. Provides methods for getting the first element, adding the next one, and holding the current one. The `cpasked` attribute has been introduced to toggle between standard and custom pieces.

- ScoreBoard: Manages the scoring system, using a binary search tree (BST) to organize player names and points. Provides methods for constructing the BST, traversing it, inserting new elements, and saving the structure to a file.

Implementation:

The project's implementation involves creating instances of these classes and integrating them to form the Tetris game. Customizations include handling rotations, theme changes, pausing, and going back to the main menu. The game progresses by dropping pieces, locking them, and generating new pieces until the game is over. Additionally, a custom piece difficulty changer has been added.

Results:

The game successfully implements the core mechanics of Tetris, including piece movement, rotation, and line clearing. The scoring system and scoreboard functionality provide a comprehensive gaming experience. The Next class has been extended to store custom pieces, and the `cpasked` attribute allows users to switch between standard and custom pieces.

Conclusion:

The project achieves its goal of implementing a Tetris game with additional features. The use of classes and data structures enhances code organization and readability. However, there are noted limitations, such as the absence of a portal feature and a segfault error during restart functionality. It is important to note that the restart feature won't work, and users can use the 'R' key, which might result in a segment fault (segmentation fault) error.

Future Improvements:

Suggestions for future improvements include addressing the segfault error associated with the restart functionality, implementing the portal feature, and enhancing security measures for file output. Additionally, further optimization of class usage for improved memory and algorithm efficiency could be explored.

Note:

For detailed information about functions and classes, refer to the provided NIKA_GOLESTANI.pdf and file and other supplementary documents.