Project Report

Ebubechukwu A., Kaleb C., Divya P., & Ishwor T.

Department of Arts & Sciences, University of Lethbridge

CPSC3620: Data Structures & Algorithms

Prof. Shahadat Hossain

April 29th, 2022

Group Member Information:

Ebubechukwu Agada - Student ID: 001223003

Kaleb Calverley - Student ID: 001230246

Divya Pateliya - Student ID: 001230022

Ishwor Tandon - Student ID: 001231876

Board_Tiles Class:

Functions:

```
*  Default Class constructor, for Board_Tile class, it initializes
*  the parameter to be the initial configuration
* @param[in] a string that contains user input for slide puzzle game
* Big-oh: N/A this function is not affected by user input in terms of execution

    FUNCTION: Board_Tile(const std::string&)
* is a bool function that will return either true or false.
* true for if the user input is valid
* false for no valid input
* Big-oh: N/A this function is not affected by user input in terms of execution

    FUNCTION: bool inputChecker()
* returns a list of Board_Tile objects that are one move away
* from the current Board_Tile object
    FUNCTION void nextConfigs()

* returns number moves taken from initial board to reach the
* current configuration
* Big-oh: N/A this function is not affected by user input in terms of execution
    FUNCTION: int numMoves()

* returns moves From Start to represent moves from initial configuration to
* current configuration
* @param[in] a Board Tile object that represents initial configuration
* Big-oh: N/A this function is not affected by user input in terms of execution
    FUNCTION: string getMyMoves()
```

```
* @param[in] a Board Tile object that represents initial configuration
* Big-oh: N/A this function is not affected user by input in terms of execution
    FUNCTION: string getMyBoard()

* swaps numbers in the tile board from left to right
* @param[in] string c is a list of characters from initial configuration
* @param[in] int left is the left element of 0 in character array
* @param[in] int right is the right element of 0 in character array
* Big-oh: is O(1) this function runs on constant time, since input size does not vary
    FUNCTION: void swap(std::string c, int left, int right)

* displays current configuration for user clarity
* Big-oh: N/A this function is not affected by user input in terms of execution
    FUNCTION: void displayBoard()

* returns the Manhattan Distance of object when compared to
* goalconfig
* @param[in] a Board Tile object that represents goalconfig
* Big-oh: is O(1) this function runs on constant time, since input size does not vary
    FUNCTION: int Manhattan_Distance(const Board_Tile* goalconfig)

* returns the position of '0' in our boards config string
* Big-oh: is O(n) this function runs on linear time, since the placement of 0 can vary
    FUNCTION: int findZero()

* returns true if a move is valid for our board
* Big-oh: N/A these function is not affected by user input in terms of execution
    FUNCTION: bool checkUp()
    FUNCTION: bool checkDown()
    FUNCTION: bool checkLeft()
    FUNCTION: bool checkRight()

* moves 0 either move up, down, left, or right
* Big-oh: is O(n) this function runs on linear time, since the placement of 0 can vary
and will result in multiple options, either it can move to that spot or not
    FUNCTION: Board_Tile* moveUp()
    FUNCTION: Board_Tile* moveDown()
    FUNCTION: Board_Tile* moveLeft()
    FUNCTION: Board_Tile* moveRight()
```

```
* Returns D(C) value from A(C) + E(C)
* Big-oh: N/A these function is not affected by user input in terms of execution
    FUNCTION: int getBothD()
```

Data Members:

```
* represents a 3 x 3 tile board configuration
    string config

* Stores the value of Manhattan Distance.
    int manHatD

* represents the moves or steps taken that led from current configuration
  from initial configuration
    string movesFromStart
```

Sliding_Solver Class:

Functions:

```
* Default Class constructor, for Sliding_Solver class, it takes initial
* configuration and goal configuration as its parameters
* @param[in] a string that contains initial configuration
* @param[in] a string that contains goal configuration
* Big-oh: N/A this function is not affected by user input in terms of execution.
    Function: Sliding_Solver(std::string&, std::string&)

* a void function that solves the puzzle using the A* search algorithm
* Big-oh: The time complexity is dependant on the heuristic
    Function: void solve_Puzzle()

* Big-oh: N/A this function is not affected by user input in terms of execution.
    Function: friend bool operator>(Board_Tile&, Board_Tile&)
```

Data Members:

```
* represents a minHeap of Board_Tile objects
    priority_queue<Board_Tile
    vector<Board_Tile>
    greater<Board_Tile>> tileQueue

* represents initial configuration
    string inConfig

* represents goal configuration
    string goalConfig

* represents the last move that was made
    char prevMove = ' '
```

Implementation:

In our main file we create a Board_Tile function for Goal configuration and initial configuration using the default constructor. After this, our program checks if the user input for both configurations is valid by running the input checker on a if condition. If BOTH conditions pass, only then will the program run the other functions. After that, we create an object of sliding_Solver to run the solve_Puzzle function. In the back-end, we use Manhattan distance to compare possible configurations in our minHeap. The most cost-effective path will be the chosen path for solution.

Thoughts on Current Solution:

We used a brute force method for calculating Manhattan Distance, we do not like this because we wanted optimal performance for our program. The initial plan was to use a recursion method, but due to time constraints, and other factors, we decided to go for the safer option. It would have been faster if we had used the Manhattam_Distance function to calculate the board's total Manhattan distance using recursion; otherwise simply store the given distance which will create the first board without this argument. When we create a new board from an existing one, calculate the change in the total distance and pass the result into the new board.