Project 1

Sliding Puzzle Game

CSC 5 – 40718

Andrew Campbell

Intro:

This is a game based on the sliding puzzle games. It is a 4x4 board equaling 16 spaces, with one spot being empty. The empty spot is what is used to move numbers around until the numbers or ending configuration is made. In this game's case it is listing the numbers in order with the very bottom right corner spot being left empty.
http://www.cs.brandeis.edu/~storer/JimPuzzles/SLIDE/CornellCrossword/KeithArticle2011.pdf

I chose this game because it seemed like it would be a good test of all the things learned in class so far, especially arrays because of the way I store the game board's current values, while at the same time being able to be built using simple ascii artwork and still keep its simplicity for the end user.

How to play:

The objective of the game is that you are trying to put the numbers in order from their current mixed state, in as few moves as possible if you want a challenge. There will be one blank spot on the board that will be used for moving numbers around, only a number directly above/below or to the left/right of the empty spot can be moved into it. The blank spot will then become the moved number's old spot, which can then accept another number to be moved into it and so on. A number that is not left/right or above/below the empty spot cannot be moved and if the user chooses such a number an error will come back letting them know it is an invalid move.

Example of playing:

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

Choosing the 7 will have the following effect:

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

From here only the 4 or 7 can be chosen as a valid move due to the empty spot being in the corner.

Attempted Addition: I worked on adding an auto play where the computer would solve the puzzle on it's own. This proved more than I could manage however. My first attempt was simply to have it randomly choose 1 of the 4 possible moves and just have it continue that a few million times thinking it might get it. After seeing that being less than fruitful I had it check to see if the number it was moving was in the correct spot and if it was it would not move it, this managed to work a bit better but plenty of scenarios where it might get trapped on one side of the board or between numbers, so still unable to win. I tried adding so that only the outer edge numbers would be 'locked' in place, but no luck with that either. Discussing the game with a friend I was reminded about 'Manhattan Distance' and how it might be used in a game like this.

The way I tried to implement this was taking each number's distance to it's ending location, each square is counted as 1 distance, so if a number would need to go up 2 spaces and over 3, it's distance would be 5 – this is ignoring any use of the blank space, just purely it's distance if it was the only number on the board. I hoped to do this for every number on the board, then add it all together for total distance on that turn. Then take this score after each of the possible moves and whichever has the lowest would be the move used for that turn. It was in this area where I ran into a brick wall, because it seemed to keep using the same 3 or 4 moves continuously just in reverse order, shuffling the same numbers back and forth. I think to solve this the program would need to keep track of further back than just the previous move and possibly 3 or 4 moves and not allow if it was repeating. This quickly spiraled from making a simple use of random to spam millions of moves in hopes of brute forcing the win, to a project far out of my skill range.

Pseudo code:

Create array by randoming 0-15 in the array and assigning numbers as for loop iterates.

Print board with current values

User inputs number they would like to move;

Function checks if that is valid move by checking where it is in the array to the 0 (the blank spot) if -/+ 1 it is accept and function swaps the values.

Reprint board using the new values

Checks if current game board matches winning game board, checks the current array against the pre done winning array for this.

If win, print message and save score (moves it took) to file, else repeat and ask for next move.

Attempted code:

The computer would first check for the game board's edges and not attempt those moves. Then from the available moves it would run what the board distance if that move were to take place, the move that came back with the lowest distance would be the one chosen.

Methods used:

Cin – line 65

Cout – line 52

Variables – lines 41-49

If statement – line 95

Else if – line 123

Else – line 88

Nested if – line 93

Switch statement – line 66

For loop – line 132-139

While loop – line 86-94

Do while loop – line 54

File operations – line 70-75

2d Arrays – line 23/25 setup, then in almost every function are used as input

Void function – line 126

Return value function – line 194

Pass by Reference – line 145