Riya Danait
CSCE 1030 Computer Science I – Section 002
Professor Helsing
December 6th, 2017

Homework 6 Algorithm

The calculations involved using the rand() and modulo functions to get the range of the numbers that are then mapped to the pieces or rows and columns. Note that if you want to get a random number in a range where the minimum is non-zero, you can do that by using rand() % (max – min + 1) + min, where [min, max] is the desired interval of numbers.

The following is a sample output of a game. It shows that the RED player can move or strike. The RED player's pieces can also be captured based on the ranking.

```
RED 7 move from C2 to B2
    1 2 3 4 5
  +-----------+
A | X X X X X |
B | X 7 X X X |
C |           |
D |     B   S |
E | B   B F 2 |
  +-----------+
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit): B2
RED MOVE: Enter new coordinates of piece at B2:A2
RED 7 at B2 captured by BLUE 2 at A2.
    1 2 3 4 5
  +-----------+
A | X X X X X |
B | X   X X X |
C |           |
D |     B   S |
E | B   B F 2 |
  +-----------+
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit): D5
RED MOVE: Enter new coordinates of piece at D5:C5
RED S move from D5 to C5
    1 2 3 4 5
```

Examples of moves that are not possible:

```
Initializing game board ...
Resume previously saved game ? (R to resume, N to start new game): R
Could not open the file game.dat to load the game
Could not load previously saved game. Startng new game...
Assigning BLUE pieces to the board ...
Assigning RED pieces to the board ...
    1 2 3 4 5
  +-----------+
A | X X X X X |
B | X X X X X |
C |           |
D | 6 1 B B S |
E | 4 F 8 B 4 |
  +-----------+
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit): D3
Error: Invalid piece or not moveable at D3. Try again...
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit): D4
Error: Invalid piece or not moveable at D4. Try again...
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit): E2
Error: Invalid piece or not moveable at E2. Try again...
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit): E4
Error: Invalid piece or not moveable at E4. Try again...
RED MOVE: Enter current coordinates of piece (e.g., D2, or QQ to quit):
```

Design Algorithm:

1. Make 3 separate files: a header file, a main.cpp file, and a func.cpp file.
2. In the header file, include the iostream, stdio.h, cstdlib, and string libraries. Define all global variables, enums, structs, and all functions declarations.
3. Create a void function to print out the rules of the game Stratego.
4. Create a constant global variable called BOARD_SIZE and set it equal to 5.
5. Define two enumerated type variables called PieceType and Color. In Pieces, define EMPTY = ' ', FLAG = 'F', BOMB = 'B', MARSHAL = '1', GENERAL = '2', COLONEL = '3', MAJOR = '4', CAPTAIN = '5', LIEUTENANT = '6', SERGEANT = '7', MINER = '8', SPY = 'S' , and HIDDEN = 'X' using the rules of the game. In Color, define RED = 'R', BLUE = 'B', and NONE = 'N'. Make a struct with enumerated type variables and include a boolean for if a piece is moveable.
6. Declare the following void functions: one to initialize the board, one to assign the pieces, one to print out the board, two to validate current and new coordinates, 1 to update the board and one to clean up or return memory to the free store. Pass through the board size and the 2-dimensional dynamic array for the struct data (pass the array as a pointer to a pointer). I have also included (as bonus) to functions that save and load the game.
7. In the main function, include the directive for the header file. Set a boolean value for if the game is over, if the board should be revealed and if the input is valid and initialize them to false. Then create a pointer to a pointer and a string for the current position and new position. Then allocate memory for the board pieces. Call all the necessary functions. Be sure to create a while loop based on whether the game is over and if the input is valid to continue the game. If the user enters QQ, quit and as part of the bonus, ask them if they want to save the game. Otherwise, exit. Then set the boolean for valid input to be false to reset for new coordinates. Then create another while loop using the boolean valid input to get the new position. Call the validate new coordinates function. After that, check to see if the game is over, and if it is, set the reveal to be true and print out the board. Then at the very end call the cleanup function.
8. In the func.cpp, include the directive for the header file. The following functions also need to be included:
    a. For the function that initializes the board, define two variables for the row and columns of the game board. Then create a for loop that is controlled by the int size that was passed through, and the int row that was created. Inside that for loop, make a nested for loop that is controlled by the ints size and column. Inside that loop, set the array boardPieces to EMPTY, and boardColor to NONE. The board is now initialized.
    b. In the function assign pieces, declare and initialize the int col and row to be 0. Seed the random function.
        i. For the FLAG: We need to assign it to the back row (row 0) for the BLUE player. So, create a do-while loop that picks a random column between 0 and the size – 1 (meaning we mod the random number by size), and

while the 2-dimensional dynamic array boardPieces is not EMPTY, it sets the boardPieces to be FLAG, and boardColor to be BLUE. Apply the same logic for RED, but instead of passing in 0 as the row, we pass in size – 1 (which would be 4 in a 5x5 array).

ii. For the rest of the pieces, there is no restriction on the location of the pieces (e.g. can be in either of the two back two rows for BLUE and RED).

iii. For the BOMBs: We need to randomly assign 3 BOMBs anywhere on the two rows for each side. We can create a for loop that runs three times, and inside that for loop we can create a do-while loop that picks a random row between 0 and 1 (meaning we mod the random number by 2), and a random column between 0 and size – 1 (meaning we mod the random number by size), and while the array boardPieces is not EMPTY, we set the boardPieces to be BOMB, and the boardColor to be BLUE. For the RED side, we apply the same logic, but for the row, we have to add size – 2 to the mod because we need to choose a random row between size – 2 and size – 1 (3 and 4 if size is 5).

iv. For the MARSHAL/GENERAL: We need to choose one of either of these pieces randomly, and assign them randomly to any of the two back rows. We can do the same steps as we did in (c) to assign the BLUE/RED pieces to any of the two back rows, but include an if-else statement to set a randomPiece to either MARSHAL or GENERAL.

v. Assigning both the MINER and the SPY involve the exact same process used in (c).

vi. For the COLONEL/MAJOR/CAPTAIN/LIEUTENANT/SERGEANT: We can choose any 3 of these to assign to any place in the two back rows for both BLUE and RED. We can run a for loop 3 times, and each time, we can generate a random int called val. We can mod val by 5 and add 3 to get a random number in the range of 3 to 7 inclusive. Then we can pick a random row between 0 and 1 for the BLUE and mod it by 2. We can do the same for column, but mod it by size. Then while the boardPieces aren't EMPTY, we can add a switch statement that determines the piece that aligns with the random number val. We will also need to include a default statement that sets the randomPiece to EMPTY. Accordingly, we can set boardPieces to randomPiece, and boardColor to BLUE. We use the same approach for the RED pieces. When we mod the random number for row, we need to add size – 2, which will pick a random row between size – 2 and size – 1. Then we use the same switch statement for the RED as we did the BLUE.

c. For the bool function that validates the current coordinates, we simply need to convert the first coordinate to the row number and the second coordinate to the column number. We check to make sure the row and column inputs are in range with the board size, and then we check to see if the piece is moveable-this means we cannot move a flag, a bomb, an empty space, or any of the BLUE pieces.

d. For the bool function that validates the new coordinates, we use the same process, checking to make sure it is in the range, the piece is moved one unit to the left, right, up, or down.
e. For the bool function that updates the board, we need to handle the case when the new position is empty. We move the piece to the new position and clear the current position and then return false. Otherwise we check if the piece occupying the position is a FLAG. If it is, RED wins the game, and it returns true. If the piece is a BOMB, then all pieces other than the MINER get blown up, and the BOMB remains intact. If the piece is a MINER, then it clears the BOMB and takes its position. If the new piece is a MARSHAL or GENERAL (1 or 2), then a SPY can capture it. Move the piece to the new position and clear the current position. Next we need to check if the new position is occupied by a BLUE piece that is inferior in rank. If it is, then the RED piece can take its place. If the destination position of a BLUE piece is of the same rank as the RED piece, both get removed from the board.
f. For the cleanup function, we simply return the memory to the free store, first by deleting the inside with a for loop and then deleting boardPieces in its entirety.
g. For the function that prints out the board, we need to pass in the two arrays and the integer size. Then we need to declare the row and column variables once more. We can start by printing the column header, which is just the numbers 1 through 5. We can do this with a simple for loop that is controlled by size and int i. We use printf to i each time the for loop increments i. Then we can print a new line. To print the top border, we can use the same for loop, but instead print out the dashes and plus signs. To print each row of the board, we print out the row name (A, B, C, D, E) which can be controlled by adding the ASCII value 65 ('A') to the row number to get the row name, and then print it as a character using static casting. We use the same code in printColor.cpp to actually print out the BLUE and RED pieces, else we print empty spots (e.g. entire row 2 in this case). Finally, we print out the bottom border in the same way we did the top border.
h. BONUS: Saving and loading the game are two separate functions that simply involve creating an output file that saves the current state of the board when the user enters QQ. Then to reopen the game, it reads from the input file and opens that state.