COMP10050 Assignment 2: Othello Game

Name: Yap Kar Yen

Student ID: 20202149

GitHub username: kar-yen

Contents

Implement of Basic Game Structure

Implement of disc

By referring to the note of lecture 11, an enumerated typed is used to define the colour of a disc in each square of board. Member BLACK represents black disc, member WHITE represents white disc and member EMPTY represents no disc on the board. Each player is assigned to one colour and the disc will be printed on the board after each turn.

Implement of board

A 2D array of structure is designed for the board. A 2D array is used because Othello is played on a board with 64 squares in an 8 x 8 arrangement. A structure is used because the board has to hold the disc colour after each turn before the game ends. Therefore, the structure designation for the board is a 2D array of structure with member disc colour of enumerated typed. The board will be printed after each move and 'B' represents black disc and 'W' represents white disc.

Implement of players

The struct design for player refers to the example given in the lecture note 11. There are 3 members in player structure which are name of player, disc color, and score of players. Member name is a char array which has maximum length of 20, member color is an enumerated typed which has pre-defined in earlier while member score is an integer type which hold the number of discs on the board and it will be updated after each turn. Two variables p1 and p2 of type struct player are created and initialized at the beginning of the program.

Implement of Game Logic

Initialization of game

File involved: main.c, initialization.c, initialization.h

First, variables for player and board should be created. Both variables p1 and p2 of type struct player should be initialized once the program starts to run.

Variable board should be a 2-dimensional array of structure of type struct disc. From the example given in the question pdf, a welcome message is printed to remind the users that the program has run successfully. An extra feature has been added to the program to ask user if they wish to read the game rules before the game starts. In order to obtain players' name, the program has to print a message to prompt user to enter their name as input from the keyboard. Then, scan the input and stored them in the variables respectively. Black disc will be assigned to player 1 and white disc will be assigned to player 2. The score of both players is initialize to 2 at the beginning of game.

After that, the board is initialized to the starting position of a game which has four discs placed in a square in the middle of the grid, two with their black side facing up, two with their white side facing up, with the two discs with the same colour on a diagonal with each other. The convention is that the discs with the black side facing up are to the north-east and south-west as both players look at the board. To set the board as like starting position, a nested for loop is implemented in the function and the discs are placed in their position by using several if loops to check the coordinates.

The function printBoard is called to print the visualized board to the user. The graphical interface of board in the code is referred to the example board given in the question pdf, which 'B' represents black disc, 'W' represents black disc, row index from range 1 to 8 at the left side of the board and column index from range a to h at the bottom of the board. Straight lines and horizontal dashes are used to visualise the square of board. The score of each player is displayed at the top of the board so that players can keep track on their score. This printBoard function is called after each move to show the updated board and score.

Play a game

File involved: playGame.c, playGame.h, getPossibleMove.c, getPossibleMove.h, updateBoard.c, updateBoard.h

Initially, I only had a rough algorithm to manage the turns between the players. I know that I have to take the input from player and compare it with all possible moves of current player to check if it is a valid move. At first, I only knew that I had to consider the possible move at different direction, but never thought of breaking them down into different functions. But then what John told in the lecturer class later was that I could first design an algorithm in one direction, and then further modify the function in other directions. Therefore, I have started to write the code for the first direction, and the remaining functions have become easier to implement in the program due to their similarities. I used the same logic and applied the same algorithm when I was writing code to update the board and score after placing a valid move. I modified the functions to get all possible moves and the details are written below.

1. Take turn between players
   - Players have to take alternative turn, so a function to shift turns between two players must be implemented.

   - A while loop is implemented to handle the game before the board has filled up and both players passed the game consecutively due to neither player can legally place disc on the board.

   - There are three new variables which are:
     a) Turn of integer type:
        Variable to hold the number of turns. This variable is used to manage turn between 2 players.
     b) Pass of integer type:
        Variable to calculate number of consecutive pass. Reset to 0 if there is a valid move after first pass.
     c) spaceLeft of integer type:
        Variable to calculate the remaining square left on the board. The value is obtained by calculating the difference between 64 (the total square on board) and the total score of both players. The value is updated after each turn.

   - A message is printed to remind player to input their move when it is their turn.

   - Prompt player to enter their move in specific format and scan their input from keyboard.

   - The program should ask the players to re-enter another move if the previous user input is invalid.

   - Example of invalid move:

a) Enter a coordinate when there is no possible move.
b) Enter an invalid coordinate that do not has one or more contiguous discs of another colour between the new disc and another disc of own colour.
c) Pass the turn when there is a valid move.
d) Enter a coordinate that is outside the range of the board, e.g., v0.

- To filter the invalid move entered by player, the user input should compare with all the possible moves of current player.

2. Get possible move of player
   - To obtain all the possible moves of current player in each turn, every discs of current player on the board should be check if there is a valid move.

   - Therefore, a nested for loop to traverse through the whole board and an if loop with condition of the colour of disc on the square is same as the disc colour of current player is implemented.

   - A new disc must be placed in such a position that there exists at least one straight occupied line (horizontal, vertical or diagonal) between it and another disc of current player, with one or more contiguous opponent player's discs with different colour between them.

   - Thus, at least one opponent player's disc must be next to (horizontal, vertical or diagonal) the current player's disc to be a valid move.

   - To get a possible move, a nested for loop is implemented. The outer for loop has integer i as the index for row from range row-1 to row+1 where row is the variable that holds the row coordinate of current player's disc. The inner for loop has integer j as the index for column from range column-1 to column+1 where column is the variable that holds the column coordinate of current player's disc.

   - Since the move is valid if there exists at least one opponent player's disc around the current player's disc, so the range is limited to plus or minus 1 with row and column.

- There are 8 directions to check if a new disc can be placed on the board if there is an opponent player’s disc around an existing current player’s disc, that are:
  a) South-west (Upper left)
  b) South (Up)
  c) South-east (Upper right)
  d) West (Left)
  e) East (Right)
  f) North-west (Bottom left)
  g) North (Down)
  h) North-east (Bottom right)

- The directions are divided into three if loops which are if the opponent player’s disc is above the current player’s disc, the opponent player’s disc is on the same row as the current player’s disc, or the opponent player’s disc is below the current player’s disc.

- Check the position of new disc can be located in specific direction and stored the coordinates in a linked list.

- There are 8 functions to get the possible move in different directions. These functions are similar to each other but the only difference will be the starting index of the for loop and moving directions which controlled by incrementing and decrementing the row and column index when there is more one opponent player’s disc on the current direction.

- For example, if current disc colour is black, the program should check every black disc on the board to see if there is any white disc around the black disc. If there is a white disc, the program will search further towards the same direction to see if there are more white discs. The searching will be stopped if a black disc or empty square is reached. If a black disc is reached, it means that there is no possible move at this direction because a new black disc cannot be placed at there. In latter case, it means that a new black disc can be placed on the board, and all the white discs at the same direction will be flipped to black disc if player chose to place a new black disc on the relative position.

- Before a new possible move is inserted in a linked list, the new coordinate is checked by function checkList to see if it exists in the list to avoid duplication.

- The new coordinate is inserted in the linked list in ascending order of row then column to ease the comparison of the possible moves with user input.

- A linked list is used to store the possible moves because it is more flexible on adjusting the memory space to store an unknown number of values, and the memory space can be free after the values are no longer needed. Initially a dynamically allocated 2-dimensional integer array was used but it is more complicated to be created and accessed, so a linked list is used to replaced it after that.

3. Update board after each turn
   - If the move entered by player is valid, the program should take the move and update the scores and board after placing the move.

   - Firstly, the program should check if the valid move is either a pass (p) or a coordinate ('ld'). If player passes the game, print a message to inform player the turn is passed and increment the variable pass. In latter case, the program will check again if the current square has no disc and place the disc of current player on the relative position. The score of current player is increased by 1 and the opponent player's discs at the relative direction is flipped to another colour. Variable pass is reset to 0 and all the nodes in linked list are free.

   - The updated score and board are then printed out to the console so that players can keep track on their game progress.

   - All the functions in file updateBoard.c are to handle and update the board and scores after each move.

- The algorithm to update and flip the disc in relative direction is similar to the logic implemented for getting the possible move of a player.

- A nested for loop with same starting indexes and ranges are used to find the specific direction to take further action.

- 8 directions are divided into three mains if loops with condition of i equal to row-1, row and row+1. 8 different update functions are then called with different column.

- In those 8 functions, 2D array board, struct player and the coordinate of new relative disc are passed as argument. They are similar to each other but just with different starting index of the for loop and moving directions to check if there are more opponent player's discs.

- For example, if the new disc placed on board is a black disc, the coordinate of new black disc will be passed to the function and check if any white disc is around the relative black disc. If there is a white disc, the program will search further towards the same direction to see if there are more white discs. The searching will be stopped if a black disc or empty square is reached. If a black disc is reached, the direction is then turned to opposite direction (e.g., direction change from east to west) to flip the white disc that lies on the straight line between the new black disc and the latter black disc into a black disc. Then, the score of black disc player is increased and the score of white disc player is decreased. If an empty square is reached, it means that it is impossible to capture the white disc in this direction because there has no another black disc to flip the white disc into black disc.

- The score and board are then displayed to players after updated. The process is repeated until the game comes to an end.

End of game

File involved: endGame.c, endGame.h

The starting time and ending time of the game are stored in different variables and passed as an argument to function mainEnd. A game over message is printed to the console so that players are able to know the game is ended. Then, the game duration is calculated and shown at the console. There are 3 possible final results, which are player 1 won the game, player 2 won the game or the game is draw. If either player has won the game, the name and score of the winner will be announced at the console. The final board is printed as required in the question along with the name and final score of both players.

Then, a text file named “othello-results.txt” is created in current directory with information of:

1. Starting date and time of the game
2. Ending date and time of the game
3. Total duration of the game
4. The name and final score of both players
5. Final results (A player won the game or draw)
6. Final board

Functions Details

main.c

- int main()
  - Main function to play a game between two human players.

initialization.c

- void mainInitialize(disc board[][BOARD_SIZE], player *p1, player *p2)
  - Main function in this c file to initialize the game.
  - Parameters:
    - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - ➢ player *p1: Pointer to struct player p1 which representing player 1, pointer is used to modify name.
    - ➢ player *p2: Pointer to struct player p2 which representing player 2, pointer is used to modify name.

- void initializePlayer(player *p1, player *p2)
  - Function to initialize player by asking user to enter their name from keyboard and store in the variables.
  - Parameters:
    - ➢ player *p1: Pointer to struct player p1 which representing player 1, pointer is used to modify name.

    - ➢ player *p2: Pointer to struct player p2 which representing player 2, pointer is used to modify name.

- void convertNewLine(char str[])
  - Function to convert newline at the end of string to '\0' in a character array.
  - Parameter:
    - ➢ char str[]: A 1D character array.

- void initializeBoard(disc board[][BOARD_SIZE])
  - Function to initialize board to starting position shown in the question pdf.
  - Parameter:
    - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.

- void printBoard(disc board[][BOARD_SIZE], player *p1, player *p2)
  - Function to print the board and discs to visualize the board
  - Parameters:
    - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - ➢ player *p1: Pointer to struct player p1 which representing player 1, pointer is used to modify name.
    - ➢ player *p2: Pointer to struct player p2 which representing player 2, pointer is used to modify name.

playGame.c

- void takeTurn(disc board[][BOARD_SIZE], player *p1, player *p2)
  - Function to manage turn by shifting turn between two players
  - Parameters:
    - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - ➢ player *p1: Pointer to struct player p1 which representing player 1.
    - ➢ player *p2: Pointer to struct player p2 which representing player 2.

- void getMove(disc board[][BOARD_SIZE], player *current, player *opponent, int num, int *pass)

- Function to take in and check the move enter by user if they are valid.
- Parameters:
  - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
  - ➢ player *current: Pointer to struct of current player.
  - ➢ player *opponent: Pointer to struct of opponent player.
  - ➢ int num: Number 1 or 2 that represents players.
  - ➢ int *pass: Pointer to int which hold the number of consecutive passes of players.

- int isEmpty(possibleMovePtr sPtr)
  - Function to check if the linked list is empty. (i.e., first node is NULL)
  - Parameter:
    - ➢ possibleMovePtr sPtr: Pointer to the beginning of the linked list (e.g., first node), used to store all the possible moves of current player in each turn.

- void delete(possibleMovePtr *cPtr)
  - Function to remove/delete (dequeue) a node in the linked list
  - Parameter:
    - ➢ possibleMovePtr *cPtr: Double pointer that points to the current node in the linked list.

- int findInList(possibleMovePtr *sPtr, int r, int c)
  - Function to find if the move entered by user is found in the linked list.
  - Return 1 if the move is found, 0 otherwise.
  - Parameters:
    - ➢ possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
    - ➢ int r: The value that represents row in the move entered by user.
    - ➢ int c: The value that represents column in the move entered by user.

- void placeMove(disc board[][BOARD_SIZE], player *current, player *opponent, const char move[], int r, int c, int *pass, int num, possibleMovePtr *sPtr)
  - Function to handle the valid move of current player by placing and flipping the discs on board and free all the nodes in the linked list before next player's turn.
  - Parameters:

- disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
- player *current: Pointer to struct of current player.
- player *opponent: Pointer to struct of opponent player.
- const char move[]: A 1D character array with type const char that holds the move entered by user.
- int r: The value that represents row in the move entered by user.
- int c: The value that represents column in the move entered by user.
- int *pass: Pointer to int which hold the number of consecutive passes of players.
- int num: Number 1 or 2 that represents players.
- possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.

getPossibleMove.c

- void findMove(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
  - Function to find all the possible moves of current player by checking if any opponent player's disc around current player's disc.
  - The range of checking are from row - 1 to row + 1 and from column - 1 to column + 1 because a valid move is existing if there has a straight line between the new disc and another relative disc of current player, with one or more contiguous discs of opponent player between them.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
    - player opponent: Struct of opponent player.
    - int row: The value that represents row of current player's disc on board.
    - int column: The value that represents column of current player's disc on board.

- int checkList(possibleMovePtr cPtr, int row, int column)
  - Function to check if the coordinates of valid move already exist in the linked list by traverse through the list to avoid same coordinate appears in the list.

- Parameters:
    - ➢ possibleMovePtr cPtr: Pointer that points to the current node in the linked list.
    - ➢ int row: The value that represents row in the valid possible move.
    - ➢ int column: The value that represents column in the valid possible move.

- void insert(possibleMovePtr *sPtr, int row, int column)
    - Function to insert/add a new coordinate of valid possible move in the linked list in ascending order according to the value of row.
    - If two coordinates have same value of row, then they will be arrange in ascending order according to the value of column.
    - Parameters:
        - ➢ possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
        - ➢ int row: The value that represents row in the valid possible move.
        - ➢ int column: The value that represents column in the valid possible move.

- void findNorthWest(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
    - Function to find possible move at north-west direction by checking if there exists at least one or more contiguous disc of opponent player at north-west direction relative to current player's disc.
    - Parameters:
        - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
        - ➢ possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
        - ➢ player opponent: Struct of opponent player.
        - ➢ int row: The value that represents row of current player's disc on board.
        - ➢ int column: The value that represents column of current player's disc on board.

- void findNorth(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
    - Function to find possible move at north direction by checking if there exists at least one or more contiguous disc of opponent player at north direction relative to current player's disc.

- Parameters:
  - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
  - possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
  - player opponent: Struct of opponent player.
  - int row: The value that represents row of current player's disc on board.
  - int column: The value that represents column of current player's disc on board.

- void findNorthEast(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
  - Function to find possible move at north-east direction by checking if there exists at least one or more contiguous disc of opponent player at north-east direction relative to current player's disc.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
    - player opponent: Struct of opponent player.
    - int row: The value that represents row of current player's disc on board.
    - int column: The value that represents column of current player's disc on board.

- void findWest(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
  - Function to find possible move at west direction by checking if there exists at least one or more contiguous disc of opponent player at west direction relative to current player's disc.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
    - player opponent: Struct of opponent player.
    - int row: The value that represents row of current player's disc on board.

    - ➢ int column: The value that represents column of current player’s disc on board.

- void findEast(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
    - Function to find possible move at east direction by checking if there exists at least one or more contiguous disc of opponent player at east direction relative to current player’s disc.
    - Parameters:
        - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
        - ➢ possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
        - ➢ player opponent: Struct of opponent player.
        - ➢ int row: The value that represents row of current player’s disc on board.
        - ➢ int column: The value that represents column of current player’s disc on board.

- void findSouthWest(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
    - Function to find possible move at south-west direction by checking if there exists at least one or more contiguous disc of opponent player at south-west direction relative to current player’s disc.
    - Parameters:
        - ➢ disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
        - ➢ possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
        - ➢ player opponent: Struct of opponent player.
        - ➢ int row: The value that represents row of current player’s disc on board.
        - ➢ int column: The value that represents column of current player’s disc on board.

- void findSouth(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
    - Function to find possible move at south direction by checking if there exists at least one or more contiguous disc of opponent player at south direction relative to current player’s disc.

- Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
    - player opponent: Struct of opponent player.
    - int row: The value that represents row of current player's disc on board.
    - int column: The value that represents column of current player's disc on board.

- void findSouthEast(disc board[][BOARD_SIZE], possibleMovePtr *sPtr, player opponent, int row, int column)
    - Function to find possible move at south-east direction by checking if there exists at least one or more contiguous disc of opponent player at south-east direction relative to current player's disc.
    - Parameters:
        - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
        - possibleMovePtr *sPtr: Double pointer that points to the first node in the linked list.
        - player opponent: Struct of opponent player.
        - int row: The value that represents row of current player's disc on board.
        - int column: The value that represents column of current player's disc on board.

updateBoard.c

- The functions in this c file are mainly modified from the functions from getPossibleMove.c.

- void mainUpdate(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
    - Function to update the board by flipping discs of opponent player on board after current player placed a move and update the score of players after each turn.
    - Check the direction of opponent player's disc from current player's new disc in the range from row - 1 to row + 1 and from column - 1 to column + 1 to flip the opponent player's discs.
    - Parameters:

    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateNorthWest(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at north-west direction by checking opponent player’s disc at north-west direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateNorth(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at north direction by checking opponent player’s disc at north direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateNorthEast(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at north-east direction by checking opponent player’s disc at north-east direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateWest(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at west direction by checking opponent player’s disc at west direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateEast(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at east direction by checking opponent player’s disc at east direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.

    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateSouthWest(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at south-west direction by checking opponent player’s disc at south-west direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateSouth(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)
  - Function to update move at south direction by checking opponent player’s disc at south direction relative to current player’s new disc.
  - The score is updated as the function is calling.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player *current: Pointer to struct of current player.
    - player *opponent: Pointer to struct of opponent player.
    - int row: The value that represents row of current player’s new disc on board.
    - int column: The value that represents column of current player’s new disc on board.

- void updateSouthEast(disc board[][BOARD_SIZE], player *current, player *opponent, int row, int column)

- Function to update move at south-east direction by checking opponent player's disc at south-east direction relative to current player's new disc.
- The score is updated as the function is calling.
- Parameters:
  - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
  - player *current: Pointer to struct of current player.
  - player *opponent: Pointer to struct of opponent player.
  - int row: The value that represents row of current player's new disc on board.
  - int column: The value that represents column of current player's new disc on board.

endGame.c

- void mainEnd(disc board[][BOARD_SIZE], player p1, player p2, time_t t1, time_t t2)
  - Function to handle and manage the ending of game by printing the final results including name and score of winner, final board, scores of players and total duration of game in minutes and seconds.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player p1: Struct of player 1.
    - player p2: Struct of player 2.
    - time_t t1: Variable that store the starting date and time of the game after initialization.
    - time_t t2: Variable that store the ending date and time of the game once there is no more valid move to be made on board by both players.

- void createFile(disc board[][BOARD_SIZE], player p1, player p2, time_t t1, time_t t2, int min, int sec)
  - Function to create a new text file with content of the final results of game in current directory.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - player p1: Struct of player 1.
    - player p2: Struct of player 2.

  - time_t t1: Variable that store the starting date and time of the game after initialization.
  - time_t t2: Variable that store the ending date and time of the game once there is no more valid move to be made on board by both players.
  - int min: Variable that holds the value of minutes of the total duration of game.
  - int sec: Variable that holds the value of seconds of the total duration of game.

- void boardFile(disc board[][BOARD_SIZE], FILE *fp)
  - Function to print the final board in the file.
  - This function is modified from printBoard function.
  - Parameters:
    - disc board[][BOARD_SIZE]: 2D array of structure for board to have row and column.
    - FILE *fp: Pointer to text file.

END