REPORT - 4

(By - Team 007)

Data structures for board and printing board on the screen

This report contains the description of data structures and functions representing the board and other elements of the game.

Find Size

The function populates **m** with the number of rows and **n** with the number of columns. Necessary to find out the size of the board by reading the first line from the input file.

```
void find_size(int *m, int *n, char *pathname) {
      /* Populating M and N */
      // figuring out the size size of the board. First line of the file.
      FILE *fp;
      fp = fopen(pathname, "r");
      int rows = 0, columns = 0;

      fscanf(fp, "%d %d\n", &rows, &columns);
      printf("# of rows: %d\n# of cols: %d\n", rows, columns);
      /* at this point we know the size of the board */
      *m = rows;
      *n = columns;
      fclose(fp);
}
```

Tile_t structure

The structure tile_t is defined in the tiles.h file which has
This structure is essential for finding player and penguin data on the board used for movement and placement phases.

```
typedef struct {
      int fishNum;
      int active;
      int isPenguin;
      int playerID;
} tile_t;
```

Deallocate Memory

Later the tile memory is deallocated to prevent memory leaks and data loss or corruption.

```
void deallocate_mem(tile_t** arr, int n) {
      for (size_t i = 0; i < n; i++)
            free(arr[i]);
      free(arr);
}
```

File IO (Reading input)

Standard C file IO is used to read the input map and setting all the board variables from the buffer.

```
void read_from_file(char *pathname, tile_t **board) {
      FILE *fp;
      fp = fopen(pathname, "r");
      char ch;
      // figuring out the size size of the board. First line of the file.
      int rows, columns;
      find_size(&rows, &columns, pathname);
      /* at this point we know the size of the board */
      fseek(fp, first_line_offset(pathname), SEEK_SET);
      char buff[100];
      //fgets(buff, 100, (FILE*)fp);

      for(int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                  fscanf(fp, "%d", buff);
                  board[i][j].fishNum = buff[0]/10;
                  board[i][j].playerID = buff[0]%10;
            }
            ch = fgetc(fp);
      }
      fclose(fp);
}
```

Print Board

Two **for loops** are used to iterate over the board and print it on the std output important for debugging process before being written into files.

```
void print_board(tile_t **board, int m, int n) {
  for (int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%d ", board[i][j].fishNum, board[i][j].playerID);
    }
    printf("\n");
  }
}
```

Save to file

Similar to read file, save to file uses write commands to write the saved data structures to desired format on the output file.

```
int save_to_file(tile_t **board, char *outputfile) {
      int m,  n;
      find_size(&m, &n, outputfile);
      FILE *pointer_to_file;
      //Open file for writing this time
      pointer_to_file = fopen(outputfile, "w");
      //find_dimensions_from_board(board, &m, &n);

      fprintf(pointer_to_file ,"%d %d\n", m, n); //Saving m and n
      //Printing array
      for(int i=0; i < m; i++) {
        for(int j=0; j < n; j++) {
//int display_value = 10 * board[i][j].fishNum + board[i][j].playerID;
          fprintf(pointer_to_file, "%d%d ", board[i][j].fishNum,
board[i][j].playerID);
         }
          fprintf(pointer_to_file, "\n");
      }
      return 0;
}
```

Testing

This function was created for A/B testing during development phase.
It uses most of the above mentioned functions like allocating memory, reading, printing and writing to the output file.

```c
void test(tile_t **board, char *pathname) {
      int rows = 0, columns = 0;
      find_size(&rows, &columns, pathname);
      board = (tile_t**)realloc(board, rows * sizeof(tile_t *));
      if (board == NULL) {
            printf("Memory failed to allocate\n");
            exit(1);
      }
      for (size_t i = 0; i < rows; i++) {
            board[i] = (tile_t*)calloc(columns, sizeof(tile_t));
            if (board[i]==NULL) {
                  printf("Memory failed to allocate\n");
                  exit(0);
            }
      }
      read_from_file(pathname, board);
      print_board(board, rows, columns);
      // board[0][0].fishNum = 0;
      printf("\n");
      print_board(board, rows, columns);
      save_to_file(board, pathname);
      deallocate_mem(board, columns);
     // board[0][0]->fishNum = 3;
     //printf("%d", board[0][0]->fishNum);
}
```