REPORT 5

Team 7

Implementation of downloading and saving the state of the game, processing of input arguments. Description of recognized errors in the input file.

Functions handling input and output file are located in a header file_management.h. Those are:

int first_line_offset(char *pathname)

As it can be read from the comment, the function returns the number of characters on the first line of a file at a given pathname - which is pointer to inputfile. Useful to find the offset for fseek, used in read_from_file().

```
int first_line_offset(char *pathname)
/* The function returns the number of characters on the first line of a file at a given pathname.
{
    FILE *fp;
    char ch = '\0';
    int offset = 0;
    fp = fopen(pathname, "r");
    while (ch!='\n')
    {
        if(ch == EOF) break;
        ch = fgetc(fp);
        offset++;
    }
    fclose(fp);
    return offset;
}
```

void find_size(int *m, int *n, char *pathname)

As it can be read from the comment, the function populates variable int m with the number of rows and variable int n with the number of columns via their pointers(int *m, int *n) from input file that c. It is used in read_from_file(). Its presence is necessary to find the size of the board to use arrays and structs properly in placement and movement phase.

```
void find_size(int *m, int *n, char *pathname)
/* The function populates m with the number of rows and n with the number of columns.
{
    /* 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);

    /* at this point we know the size of the board */
    *m = rows;
    *n = columns;
    fclose(fp);
}
```

void read_from_file(char *pathname, tile_t **board)

This is the most important part of reading from file. This function is used to fill array of structures (board) ,of a size that was read earlier from input file using pointer to its pathname, with values of fishNum(number of fish on tile), playerID(Id of player on tile), active(1 if active, 0 if not).

```
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;
                board[i][j].active = 0;
            if (board[i][j].playerID != 0) board[i][j].active = 1;

        }
        ch = fgetc(fp);
    }
    fclose(fp);
}
```

void print_board(tile_t **board, int m, int n)

This function is responsible for printing the board in the interactive mode so as to visualize game state and allow the player to chose penguin to place/move. It takes variables m and n and pointer to board to access fishNum and playerID.

```
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");
    }
}
```

int save_to_file(tile_t **board, char *outputfile)

This function saves the data from board to the output file that is given by its pointer.

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

    //find_dimensions_from_board(board, &m, &n);


    fprintf(pointer_to_file ,"%d %d\n", m, n); //Saving m and n

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

Validity of reading from file is checked using:

int is_board_correct(int m, int n, int board[m][n][4])

This function checks if the board was generated correctly. It works during placement phase in both modes. It takes dimensions of the board m x n and values from board to check if number of fish on tiles is correct and if all the penguins are placed correctly.

```
int is_board_correct(int m, int n, int board[m][n][4])
{
    int i, j;
    for(i=0; i<m; i++) //Checking if the is no more than 3 fish on one tile
    {
        for(j=0; j<n; j++)
        {
            int display_value = 10 * board[i][j][0] + board[i][j][3]; //Display value is a value displayed and saved in file
            if(display_value >= 40 || display_value < 0) //Improper number of fish
            {
                int wrong_fish = (display_value - display_value%10)/10;
                printf("Skipper: Kowalski, analysis.\n");
                printf("Kowalski: There is too much or too few fish on tile [%d,%d]. Exactly: %d\n", i, j, wrong_fish);
                return 3;
            }
            else if(display_value>19 && display_value!=20 && display_value!=30) //Penguin on improper place
            {
                int team_mistake = board[i][j][3];
                int wrong_fish = (display_value - team_mistake)/10;
                printf("Skipper: Kowalski, analysis.\n");
                printf("Kowalski: There is a penguin on a tile [%d,%d] where there are %d fish. Its group number is %d. \n", i, j, wrong_fish, team_
                return 3;
            }
        }
    }
    return 0;
```