```
GameState Game; // Creating a Game variable that will hold informations needed to run the program
Game.command_line = generateCommandLine(Game, argc, argv);  // Generates 2D dynamic array of strings
handlingParameters(argc, argv, Game);  // Stores all commands inside Game.command_line array
Game.file_type = 0;  // We use this variable to check if the file we open is inputboardfile or
outputboardfile
Game.phase = 0;  // We use this variable to check if current game phase is placement (1) or movement(2)
Game.penguins_from_command = 0;  // This variable checks if the user enter number of penguins in placement
phase
Game.check_for_id = 0;  // This variable checks if we got instruction to check id from the command line
Game.our_player_number = 0;  // This variable will be used to determine our player number based on his ID

for(int i = 0; i < argc - 1; i++)
{
  if(checkParameterType(&Game, i) == 0)
  {
    printf("One of the parameters you entered is wrong!\n");
    return 2;
  }  // Inside this loop we run through each command and check what it does and assign proper values from
those commands
}
```

Error handling, Program checks:

- Any spelling errors inside parameters
- If number of files is equal to 2
- Checks if row number and column number is a number and if it is different than 0
- Checks if the fish number on the field is a number between 0 and 3
- Checks if the player number on the field is a number between 0 and 9
- Checks if the player number on the field match number of players from below the board
- Checks if there are any extra characters between fields
- Checks if the column number and row number from the file match the actual board
- Checks if player ID contains a space or a quotation mark
- Checks if player ID is the same as of some other player ID mentioned earlier
- Checks if player numbers are in order and start from 1
- Checks if player score is a number
- Checks if there is player ID for our player
- If in placement phase checks if we got any penguins from the command
- Checks if we put all our penguins in placement and checks if we there is no more moves in movement


if(Game.check_for_id == 0)
{
...
}else
{
printf("marcel"); // Temporary ID
return 0;
}

```
if(Game.file_type == 2)  // Checks if we got boards files from the commands
{
   ...

}else if(Game.file_type == 1)
{
    printf("You entered input board file name but did not enter output board file name!");
    return 2;
}else if(Game.phase != 0)
{
    printf("You did not enter input board file and output board file");
    return 2;
}
```

```c
if(!handleInputFile(&Game))  // handleInputFile returns 1 if something went wrong
{

}else
{
  printf("There is a mistake in board file! "); // type of mistake is printed out in handleInputFile
function
  fclose(Game.input_board);
  return 2;
}
```

```c
if(checkForStrayPenguins(Game) == 1)  // Checks if there are any penguins that don't belong to any player
{
    printf("There is a stray penguin!\n");
    return 2;
}
```

```
if(Game.penguins_from_command == 0)  // Checks if we got any penguins from command
{
    printf("We didn't get any penguins from command!\n");
    return 2;
}
if(Game.penguins == ourPenguins(Game))  // Checks if we put all our penguins
{
    printf("All penguins have been placed");
    return 1;
}else
{
    if(automaticPlacePenguin(&Game) == 0)   // Tries to put penguin somewhere and if unable returns 0
    {
        printf("You can't put penguin anywhere!\n ");
        handleOutPutFile(Game);
        return 1;
    }else
    {
        printf("Program placed a penguin!");
        handleOutPutFile(Game);
        return 0;
    }
}
}else if(Game.phase == 2 && Game.penguins_from_command == 0)
```

```
}else if(Game.phase == 2 && Game.penguins_from_command == 0)  // Checks if it is a movement phase
{
    if(checkMove(Game) == 0) // This function checks if there are any moves left for any player
    {
        printf("All penguins are blocked! ");
        handleOutPutFile(Game);
        return 1;
    }else
    {
        automaticMovePenguin(Game);  // Moves our penguin automatically
        handleOutPutFile(Game);
        return 0;
    }
}else
{
    printf("You entered number of penguins even though it is a movement phase!\n");
    return 2;
}
```

```c
int handleInputFile(GameState *Game)
{
    char rows[30], columns[30];  // Stores rows number and columns number from the board file
    char c;  // Gets all the characters from the file
    int rows_number, columns_number;  // Stores int values of rows and columns number from the board file
    int i = 0, j;
    while((c = getc(Game -> input_board)) != ' ')  // Getting rows number
    {
        if(c >= '0' && c <= '9' && c != ' ') // Checks if rows number is a number
        {
            rows[i] = c;
            i++;
            if(i >= 30)
            {
                printf("Row number is too big! \n");
                return 1;
            }
        }else
        {
            printf("row number is not a number! \n");
            return 1;
        }
    }
```

```
i = 0;
while((c = getc(Game -> input_board)) != ' ' && c != '\n')  // Gets columns number
{
    if(c >= '0' && c <= '9' && c != ' ')  // Checks if columns number is a number
    {
    columns[i] = c;
    i++;
    if(i >= 30)
    {
        printf("Columns number is too big! \n");
        return 1;
    }
    }else
    {
        printf("row number is not a number! \n");
        return 1;
    }
}

rows_number = atoi(rows);  // Changes rows number from a string to an integer
printf("Rows number is: %d\n", rows_number);
columns_number = atoi(columns);
printf("Columns number is: %d\n", columns_number);

Game -> board_height = rows_number;
Game -> board_width = columns_number;
```

```c
rows_number = atoi(rows);  // Changes rows number from a string to an integer
printf("Rows number is: %d\n", rows_number);
columns_number = atoi(columns);
printf("Columns number is: %d\n", columns_number);

Game -> board_height = rows_number;
Game -> board_width = columns_number;

if(rows_number == 0 || columns_number == 0)  // Checks if rows or columns number is equal to 0
{
    if(rows_number == 0)
    {
        printf("Number of rows is equal to 0!\n");
        return 1;
    }else
    {
        printf("Number of columns is equal to 0!\n");
        return 1;
    }
}
Game -> board = allocateMemory(*Game);  // Allocates memory for our board
int check = 0;  // flag that is either 0 if we get fish number of 1 if we get player number
int value = 0;  // temporary value for fish number and player number
int three = 1; // Used to check if there are any extra characters in file between board files
int current_columns = 0;  // Used to check is there are any extra columns
```

```c
for(int i = 0; i < Game -> board_height; i++)  // This for loop gets fields from the board file
{
    int j = 0;
    while((c = getc(Game -> input_board)) != '\n')
    {
        if(three % 3 != 0 || (i == 0 && j == 0))
        {
            if(c != ' ' && check == 0)
            {
                if(c >= '0' && c <= '3' && c != ' ')
                {
                    value = c - '0';  // According to Stack Overflow it works and is valid with all standards
                    Game -> board[i][j].fish_no = value;
                    check = 1;
                    three++;
                }else if(c > '3' && c <= '9')
                {
                    printf("Number of fishes on field: [%d][%d] is too big!\n", i ,j);
                    return 1;
                }else
                {
                    printf("Number of fishes on field: [%d][%d] is not a number!\n", i, j);
                    return 1;
                }
            }else if(c != ' ' && check == 1)
            {
                if(c >= '0' && c <= '9')
                {
                    value = c - '0';
                    Game -> board[i][j].player_no = value;
                    check = 0;
                    j++;
                    three++;
                    current_columns++;
                }else
                {
                    printf("Number of players on field: [%d][%d] is not a number!\n", i, j);
                    return 1;
                }
            }
        }else if(c == ' ')
        {
            three++;
        }else
        {
            printf("There are more than two elements on field: [%d][%d]", i, j);
            return 1;
        }
    }
    if(current_columns != columns_number)
    {
        printf("There is different number of columns than there should be!\n");
        return 1;
    }else current_columns = 0;
}
```

```c
while((c = getc(Game -> input_board)) != EOF)  // Checks last m+2 and consecutive rows
{

    j = 0;
    do   // checks player id
    {
        if(c != ' ' && c != '"' && c != '\n' && c != EOF)
        {
            if(j < 500)
            {
            Game -> player_data[expected_player].playerID[j] = c;
            j++;
            }else
            {
                printf("Id of this player is too long!\n");
                return 1;
            }
        }else if(c == ' ')
        {
            printf("Player ID contains a space! ");
            return 1;
        }else if(c == '"')
        {
            printf("Player ID contains a quotation mark! ");
            return 1;
        }else if(c == '\n') // It is used to check if there is any new line instead of player ID and if there is the program assumes that there are no more players and data for us
        {
            while((c = getc(Game -> input_board)) != EOF)
            {
                if(c != ' ' && c != '\n' && c != EOF)  // Here it checks if there are mno more players or data for us
                {
                    printf(":<");
                    return 1;
                }
            }
            Game -> total_players = expected_player - 1;
            return 0;
        }else
        {
            printf("Player ID contains end of file character!");
            return 1;
        }

    }while((c = getc(Game -> input_board)) != ' ');  // PART 1: CHECKING PLAYER ID
```

```
for(i = strlen(Game -> player_data[expected_player].playerID); i >= j; i--)  // We clear playerID string characters after the characters we get from the file
{
    Game -> player_data[expected_player].playerID[i] = 0;
}

if(samePlayerID(*Game, expected_player) && expected_player != 1)  // Checks if the players ID's are the same
{
    printf("Two player ID's are the same!\n");
    return 1;
}
```

```
c = getc(Game -> input_board);
if(c >= '1' && c <= '9')
{
  Game -> player_data[expected_player].player_number = c - '0';  // Gets player number from the file
  if(expected_player != Game -> player_data[expected_player].player_number)
  {
      printf("Player numbers from the file is not in order!");
      return 1;
  }

  if(!strcmp(Game -> player_data[expected_player].playerID, "marcel"))  // Checks if the player ID is our's
player ID
  {
      Game -> our_player_number = Game -> player_data[expected_player].player_number;
  }

  if( getc(Game -> input_board) == ' ')
  {
  char score[100];
  i = 0;
  while((c = getc(Game -> input_board)) != '\n' && c != EOF)  // Gets player score from the file
  {
      if(c >= '0' && c <= '9' && i < 100)
      {
          score[i] = c;
          i++;
      }else
      {
          printf("Player score is not a number!\n");
          return 1;
      }

  }

  Game -> player_data[expected_player].score = atoi(score);
  printf("Player ID is: %s\n", Game -> player_data[expected_player].playerID);
  printf("Player number is: %d\n", Game -> player_data[expected_player].player_number);
  printf("Player score is: %d\n", Game -> player_data[expected_player].score);
  Game -> lines_number++;
  expected_player++;
  }else
  {
      printf("Player number is longer than 1 character! ");
      return 1;
  }
}else
{
  printf("Player number is not a number or is equal to 0 \n");
  return 1;
}

}
if (Game -> our_player_number == 0)
{
printf("There is no id for our player!\n");
return 1;
}
```

```c
int handleOutPutFile(GameState Game)
{
    int i, j;
    fprintf(Game.output_board, "%d %d\n", Game.board_height, Game.board_width);  // Puts board height and board width to output file
    for(i = 0; i < Game.board_height; i++)  // Puts board to the output file
    {
        for(j = 0; j < Game.board_width; j++)
        {
            fprintf(Game.output_board, "%d%d ", Game.board[i][j].fish_no, Game.board[i][j].player_no);
        }
        fprintf(Game.output_board, "\n");
    }
    for(i = 1; i < Game.lines_number + 1; i++)  // Puts players ID's, numbers and scores to output file
    {
        fprintf(Game.output_board, "%s %d %d\n", Game.player_data[i].playerID, Game.player_data[i].player_number, Game.player_data[i].score);
    }
    fclose(Game.output_board);
    return 0;
}
```

```c
int samePlayerID(GameState Game, int k)
{

    for(int i = 1; i < k; i++)  // Loop goes through all the ID's and checks if any of them are equal to the player k ID
    {
        if(!strcmp(Game.player_data[i].playerID, Game.player_data[k].playerID))
        {
            return 1;
        }
    }
    return 0;
}
```

```c
int numberOfPenguins(GameState Game, int command_number)  // Returns number of penguins from the command line
{
    char number[10];
    int len = strlen(Game.command_line[command_number]);
    int j = 0;
    for(int i = 9; i < len; i++)
    {
        number[j] = Game.command_line[command_number][i];
        j++;
    }
    Game.penguins = atoi(number);
    return Game.penguins;
}
```

```c
int isItBoardParameter(GameState Game, int command_number)  // Checks if the parameters is a board file parameter
{
    int len = strlen(Game.command_line[command_number]);
    int j = 0, i;
    char extension[4];
    int answer = 0;
    memset(extension, 0, strlen(extension));  // Extension holds last 4 characters from the command line parameter
    for(i = len - 4; i < len; i++)
    {
        extension[j] = Game.command_line[command_number][i];
        j++;
    }
    if(!strcmp(extension, ".txt"))  // If the parameter ends with .txt we assume it is a board file
    {
        answer = 1;
    }else answer = 0;
    return answer;
}
```

```c
char** handlingParameters(int argc, char *argv[], GameState Game)  // Puts all command line parameters into an array of strings inside of our program
{
    int i, j;
    for(i = 1; i < argc; i++)
    {
        strcpy(Game.command_line[i - 1], argv[i]);
    }
    for(i = 0; i < argc - 1; i++)
    {
        printf("%s\n", Game.command_line[i]);
    }
    return Game.command_line;
}
```

```c
int checkParameterType(GameState *Game, int command_number)
{
    if(!strcmp(Game -> command_line[command_number], "phase=placement"))  // Checks if the command line parameters says phase=placement
    {
        Game -> phase = 1;
        printf("Game phase is placement!\n");
        return 1;
    }else if(!strcmp(Game -> command_line[command_number], "phase=movement")) // Checks if the command line parameters says phase=movement
    {
        Game -> phase = 2;
        printf("Game phase is movement!\n");
        return 1;
    }else if(!strncmp(Game -> command_line[command_number], "penguins=", 9)) // Checks if the command line parameters says penguins
    {
        Game -> penguins = numberOfPenguins(*Game, command_number);
        Game -> penguins_from_command = 1;
        printf("There are %d penguins\n", Game -> penguins);
        return 2;
    }else if(isItBoardParameter(*Game, command_number))  // // Checks if the command line parameters gives us board file name
```

```c
}else if(isItBoardParameter(*Game, command_number))  // // Checks if the command line parameters gives us
board file name
{
    printf("It is a board file!\n");
    if(Game -> file_type == 0)  // First file is input board file
    {
        printf("It is an input board file!\n");
        char line[255];
        Game -> file_type = Game -> file_type + 1;
        printf("%s", Game -> command_line[command_number]);
        Game -> input_board_command_number = command_number;
        Game -> input_board = fopen(Game -> command_line[command_number], "r+");
        return 3;
    }else if(Game -> file_type == 1) // Second file is output board file
    {
        printf("It is an output board file!\n");
        char line[255];
        if(!strcmp(Game -> command_line[Game -> input_board_command_number], Game ->
command_line[command_number]))  // If the input and board file are the same
            // then we open it in r+ mode
        {
            Game -> output_board = fopen(Game -> command_line[command_number], "r+");
        }  // Else we open it in w mode
        else
        {
            Game -> output_board = fopen(Game -> command_line[command_number], "w");
        }
        Game -> file_type = Game -> file_type + 1;
        return 4;
    }else if(Game -> file_type == 2)
    {
        printf("There are too many files!\n");
        return 0;
    }
}else if(!strcmp(Game -> command_line[command_number], "id"))
{
    printf("It asks for id!\n");
    Game -> check_for_id = 1;
    return 5;
}else  // This else will work if none of the paramaters types match the parameter put from the user
{
    printf("This: '%s' command line parameter is wrong!\n", Game -> command_line[command_number]);
    return 0;
}
```

```c
int checkMove(GameState Game)
{
    Movement_coordinates Coordinates;
    int i, j, anymove;
    for(i = 0; i < Game.board_height; i++)  // This goes through the whole board and checks if there are any moves possible for every penguin we find
    {
        for(j = 0; j < Game.board_width; j++)
        {
            if(Game.board[i][j].player_no != 0)
            {
                Coordinates.coordinate_1 = i;
                Coordinates.coordinate_2 = j;
                printf("We check if there is any move possible at board[%d][%d]\n", Coordinates.coordinate_1, Coordinates.coordinate_2);
                anymove = anyMovePossible(Coordinates, Game);
                if(anymove != 0)
                {
                    printf("There is a possible move!\n");
                    return 1;
                }
            }
        }
    }
    printf("There are no possible moves!\n");
    return 0;
}
```

```
int ourPenguins(GameState Game)
{
    int i, j;  // This function checks how many penguins we have and returns this number
    int number_of_our_penguins;
    number_of_our_penguins = 0;
    for(i = 0; i < Game.board_height; i++)
    {
        for(j = 0; j < Game.board_width; j++)
        {
            if(Game.board[i][j].player_no == Game.our_player_number)
            {
                number_of_our_penguins++;
            }
        }
    }
    return number_of_our_penguins;
}
```

How checking if there are any moves possible works:
We check if penguin can move 1 tile UP, DOWN, LEFT or RIGHT
If it is in the corner we don’t check 2 movement possibilities
If it is on the edge we don’t check 1 movement possibility

```
int anyMovePossible(Movement_coordinates Coordinates, GameState Game)
{
    if(Coordinates.coordinate_1 > 0 && Coordinates.coordinate_1 + 1 < Game.board_height)
    {
        if(Coordinates.coordinate_2 > 0 && Coordinates.coordinate_2 < Game.board_width)
        {
            if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].player_no == 0) // LEFT
            {
                return 1;
            }else if(Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].player_no == 0) // UP
            {
                return 2;
            }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // DOWN
            {
                return 3;
            }else if(Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].player_no == 0) //RIGHT
            {
                return 4;
            }else return 0;
        }else if(Coordinates.coordinate_2 == 0)
        {
            if(Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].player_no == 0) // UP
            {
                return 2;
            }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // DOWN
            {
                return 3;
            }else if(Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].player_no == 0) //RIGHT
            {
                return 4;
            }else return 0;
        }else if(Coordinates.coordinate_2 == Game.board_width - 1)
        {
            if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].player_no == 0) // LEFT
            {
                return 1;
            }else if(Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].player_no == 0) // UP
            {
                return 2;
            }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // DOWN
            {
                return 3;
            }else return 0;
        }else
        {
            printf("You missed a case!\n");
            return 0;
        }
    }else if(Coordinates.coordinate_1 == 0)
```

```
}else if(Coordinates.coordinate_1 == 0)
{
    if(Coordinates.coordinate_2 > 0 && Coordinates.coordinate_2 < Game.board_width)
    {
        if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].player_no == 0) // LEFT
        {
            return 1;
        }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // DOWN
        {
            return 3;
        }else if(Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].player_no == 0) //RIGHT
        {
            return 4;
        }else return 0;
    }else if(Coordinates.coordinate_2 == 0)
    {
        if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // DOWN
        {
            return 2;
        }else if(Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 + 1][Coordinates.coordinate_2].player_no == 0) //RIGHT
        {
            return 4;
        }else return 0;
    }else if(Coordinates.coordinate_2 == Game.board_width)
    {
        if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].player_no == 0) // LEFT
        {
            return 1;
        }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // DOWN
        {
            return 3;
        }else return 0;
    }else
    {
        printf("You missed a case!\n");
        return 0;
    }
}else if(Coordinates.coordinate_1 + 1 == Game.board_height)
```

```
}else if(Coordinates.coordinate_1 + 1 == Game.board_height)
{
    if(Coordinates.coordinate_2 > 0 && Coordinates.coordinate_2 < Game.board_width)
    {
        if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].player_no == 0) // LEFT
        {
            return 1;
        }else if(Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].player_no == 0) // UP
        {
            return 2;
        }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // RIGHT
        {
            return 4;
        }else return 0;
    }else if(Coordinates.coordinate_2 == 0)
    {
        if(Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].player_no == 0) // UP
        {
            return 2;
        }else if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 + 1].player_no == 0) // RIGHT
        {
            return 4;
        }else return 0;
    }else if(Coordinates.coordinate_2 == Game.board_width)
    {
        if(Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].fish_no != 0 &&
Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2 - 1].player_no == 0) // LEFT
        {
            return 1;
        }else if(Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].fish_no != 0 &&
Game.board[Coordinates.coordinate_1 - 1][Coordinates.coordinate_2].player_no == 0) // UP
        {
            return 2;
        }else return 0;
    }else
    {
        printf("You missed a case!\n");
        return 0;
    }
}else
{
    printf("You missed a case!\n");
    return 0;
}
}
```

```c
void movePenguin(GameState Game)  // Early version of moving the Penguin for interaction Phase
{
    int player_number;
    Movement_coordinates Coordinates;
    printf("Please, type where the penguin you want to use is: ");
    scanf("%d \n %d \n", Coordinates.coordinate_1, Coordinates.coordinate_2);
    printf("Please, type where you want to put your penguin: ");
    scanf("%d \n %d \n", Coordinates.coordinate_3, Coordinates.coordinate_4);
    player_number = Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2].player_no;
    Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2].player_no = 0;
    Game.player_data[player_number].score += Game.board[Coordinates.coordinate_3]
[Coordinates.coordinate_4].fish_no;
    Game.board[Coordinates.coordinate_3][Coordinates.coordinate_4].fish_no = 0;
    Game.board[Coordinates.coordinate_3][Coordinates.coordinate_4].player_no = Game.current_player;
    return;
}
```

```c
void placePenguin(GameState Game)  // Early version for placing the penguin in placement phase for interactive mode
{
    Movement_coordinates Coordinates;
    printf("Please, type where you want to put your penguin: ");
    scanf("%d \n %d \n", Coordinates.coordinate_1 ,Coordinates.coordinate_2);
    Game.player_data[Game.current_player].score = Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2].fish_no;
    Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2].fish_no = 0;
    Game.board[Coordinates.coordinate_1][Coordinates.coordinate_2].player_no = Game.current_player;
    return;
}
```

```c
#include "game_struct.h"

void print_board(GameState Game)
{
    printf(" ");

    for (int i=0;i<Game.board_width;i++)
    {
        printf(" %d",i); // Here we print the number of columns
    }
    printf("\n");
    for(int i = 0; i < Game.board_height; i++)
    {
        printf("%d ",i);  // Here we print the number of rows
        for(int j = 0; j < Game.board_width; j++)
        {
            if ((Game.board[i][j].fish_no==0) && (Game.board[i][j].player_no==0))
            {
                printf("~ ");  // If there are no penguins and fish on the field we print the 'wave'
            }
            else if(Game.board[i][j].fish_no==0)
            {
                printf("%d ",Game.board[i][j].player_no);
            }
            else
            {
                printf("%d ",Game.board[i][j].fish_no);
            }
        }
        printf("\n");
    }
}
```