HOCHSCHULE
HAMM-LIPPSTADT

Battleship Game

Implementation Document

COURSE: COMPUTER SCIENCE II

SUMMER SEMESTER 2021

JULY 5, 2021

Jad Al Jourdi
Imad Chaar
Stephanie Chinenye Okosa
Maxim Emile Spezcyk

Table of Contents

1. Introduction
2. Rules
3. State Diagram
4. *GetLetter & GetNumber*
5. void *printGrid*
6. void *placeShip*
7. int *checkForFreeSpaceHorizontally*
8. int *checkForFreeSpaceVertically*
9. void *printShipPlacement*
10. void *takeTurn*
11. int *checkForSunk*
12. int *checkForWinner*
13. int *main*

1. Introduction:

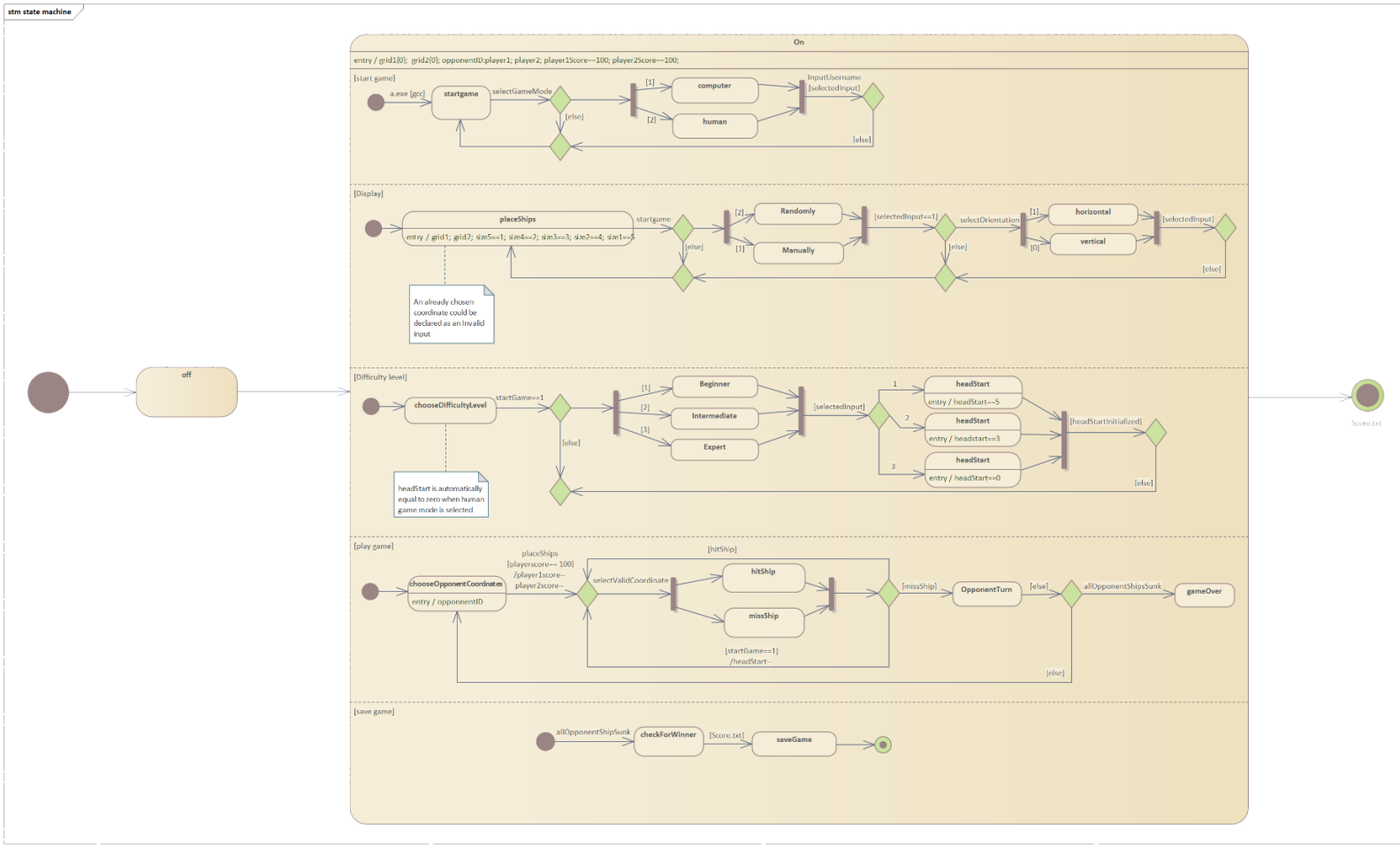
The following document clarifies the program that is simulated by the Battleship game. It is a two-player game whose objective is to hit and sink all the ships in the enemy's matrix, where the first player who sinks all 10 of the enemy's ships wins. Moreover, it elaborates how the battleship game functions were represented and illustrated based on the functions of the actual real-life game.

The game is made up of two grids (a grid for each player), each consists of a 2-dimensional array represented as 10 x 10 containing 10 ships of different sizes placed horizontally or vertically. Players have the option to choose between placing the ships manually or randomly on the grid. For placing the ships manually, the player must input a combination of a letter and a number for the rows and columns, respectively. After placing the ships, the player gets to choose between the difficulty levels provided. The game then begins, and the player starts guessing the coordinates of the ships on the grid and will be notified by a message stating: "hit", "miss" or "sunk". Finally, the result of the game is saved as a Score.txt file.

2. Rules:

Before the game starts, each opponent places their own ships on their grid either manually or randomly. Each ship must be placed horizontally or vertically across grid spaces and the ships must have at least one empty space between one another. Ship placements cannot be modified after the game starts. Players take turns choosing coordinates off their opponent's grid to attempt to hit the opponent's enemy ships. On your turn, input a letter and a number that identifies a row and column on your target grid. The coordinates are checked for a ship, a return message will be declared, either "miss" if there is no ship there, or "hit" if you have correctly guessed a space that is occupied by a ship. "Sunk" is stated when all coordinates of a single ship are hit. The first player to sink all 10 of their opponent's ships wins the game.

3. State Diagram showing the sequence of Battleship Game:



4. *GetLetter & GetNumber:*

Both functions were developed accounting for how the grids the x & y positions within the code are irregularly ordered and we would be required to flip their positions when inputted by the user.

The GetLetter function turns an int variable into its corresponding char variable on of the grid. Accepted inputs are 0-9 and the given outputs are A-J.

The GetNumber function turns a char variable into its corresponding int variable of the grid. Accepted inputs are A-J and the given outputs are 0-9.

```
char getLetter(int num) //fuction to get corresponding letter from a given index
{
    if (num == 0)
    {
        return 'A';
    }
    else if (num == 1)
    {
        return 'B';
    }
    else if (num == 2)
    {
        return 'C';
    }
    else if (num == 3)
    {
        return 'D';
    }
    else if (num == 4)
    {
        return 'E';
    }
    else if (num == 5)
    {
        return 'F';
    }
    else if (num == 6)
    {
        return 'G';
    }
    else if (num == 7)
    {
        return 'H';
    }
    else if (num == 8)
    {
        return 'I';
    }
    else if (num == 9)
    {
        return 'J';
    }
```

```
int getNumber(char letter) //function to get corresponding index from the letter
{
    if (letter == 'A')
    {
        return 0;
    }
    else if (letter == 'B')
    {
        return 1;
    }
    else if (letter == 'C')
    {
        return 2;
    }
    else if (letter == 'D')
    {
        return 3;
    }
    else if (letter == 'E')
    {
        return 4;
    }
    else if (letter == 'F')
    {
        return 5;
    }
    else if (letter == 'G')
    {
        return 6;
    }
    else if (letter == 'H')
    {
        return 7;
    }
    else if (letter == 'I')
    {
        return 8;
    }
    else if (letter == 'J')
    {
        return 9;
    }
    else
        return 0;
}
```

5. void *printGrid:*

This function prints the grid with the guesses of either player1 or player2 (the player being specified in the parameter of the function) indicated by either ‘A’ or ‘B’ respectively. The grid holds a value of -1 when a ship has been hit and holds a -2 value when an empty coordinate has been chosen by the user, else it is 0.

```
void printGrid(int grid[10][10], int player) //print guesses of the players
{
    printf("\t   1  2  3  4  5  6  7  8  9  10\n");
    printf("\t +---------------------------------+\n");

    for (int i = 0; i < 10; i++)
    {
        printf("%c\t|  ", getLetter(i));
        for (int j = 0; j < 10; j++)
        {
            if (grid[i][j] == -1 || grid[i][j] == -2)
            {
                if (player == 1)
                {
                    printf("A  ");
                }
                else
                {
                    printf("B  ");
                }
            }
            else
            {
                printf("   ");
            }
        }
        printf("  |\n");
    }
    printf("\t +---------------------------------+\n");
}
```

6. void *placeShip:*

The place ship function scans input from the user if manual placement was chosen and places each ship size until all 10 ships are placed on the grid. The *placeShip* function contains the *printShipPlacements*, *checkForFreeSpaceHorizontally* and *checkForFreeSpaceVertically* functions.

```
void placeShips(int grid[10][10],int player,int randomFlag)
{

    int randomX = 0;
    int randomY = 0;
    int horizontal = 0;

    int size2 = 0;
    int size3 = 0;
    int size4 = 0;
    int size5 = 0;

    if (randomFlag == 0)
    {
        printf("In order to place the ships manually use a combination of a letter for rows and a number for the columns.\nFor example, G5\n");
        printf("Press 0 for Vertical and press 1 for Horizontal placement\n\n\n");
    }
    while (size2 != 4  || size3 != 3 || size4 != 2 || size5 != 1) //continue until all ships are placed on the board
    {

        if (randomFlag == 1) //random placement
        {
            randomX = rand() % 10; //generate random x value
            randomY = rand() % 10; //generate random y value
            horizontal = rand() % 2; // generate random horizontal value
        }
        else //manual placement
        {
            printShipPlacements(grid); //show current placement
            char letter;
            if (size5 != 1)
            {
                printf("Enter location for ship with size 5:");
            }
            else if (size4 != 2)
            {
                printf("Enter location for ship with size 4:");
            }
            else if (size3 != 3)
            {
                printf("Enter location for ship with size 3:");
            }
            else if (size2 != 4)
            {
                printf("Enter location for ship with size 2:");
            }
            scanf("%c", &letter);
            scanf("%c", &letter);
            randomY = getNumber(letter);
            int num;
            scanf("%d", &num);
            randomX = num - 1;
            printf("Enter orientation:");
            scanf("%d", &horizontal);
            printf("\n");
        }
```

When placing the ships manually the grid must be checked to see if there is enough space to place a specific ship which lead to the development of the *checkForFreeSpaceHorizontally* and *checkForFreeSpaceVertically* functions.

7. int *checkForFreeSpaceHorizontally:*

When placing a ship manually horizontally we must check that the ship does not overlap with another ship or the border and there must be one or more space between every ship. When checking horizontally, the x variable of the grid is manipulated. First, we check if the ship begins or ends on the edges of the grid, allowing us to check either one block before the ship if the chosen x variable is greater than 0, and one block after when the chosen x coordinate with the ship size is less than or equal to 9. This step allows us to define the starting and ending indexes of the ship. After defining the starting and ending indexes we check to see if the supposed ship coordinates are free. Considering how ships are supposed to have at least one space apart from other ships in all directions, we also check one space above and below the ship. If the spaces on the grid are not free, a 0 is returned.

```
int checkForFreeSpaceHorizontally(int grid[10][10], int x, int y, int size) //check for horizontal space
{
    if (x + size -1 <= 9)
    {
        int endingIndex = size;
        int startingIndex = 0;
        if (x + size <= 9)
        {
            endingIndex = size + 1;
        }
        if (x > 0)
        {
            startingIndex = -1;
        }

        for (int i = startingIndex; i < endingIndex; i++)
        {
            if (grid[y][x + i] != 0)
            {
                return 0;
            }
            if (y > 0)
            {
                if (grid[y - 1][x + i] != 0)
                {
                    return 0;
                }
            }
            if (y < 9)
            {
                if (grid[y + 1][x + i] != 0)
                {
                    return 0;
                }
            }
        }

        return 1;
    }
    else
    return 0;
}
```

8. int *checkForFreeSpaceVertically:*

This function works in a similar fashion to the *ChechForFreeSpaceHorizontally* function, except in this case the y variable of the grid is manipulated.

```
int checkForFreeSpaceVertically(int grid[10][10], int x, int y, int size) //check for vertical space
{
    if (y + size - 1 <= 9)
    {
        int endingIndex = size;
        int startingIndex = 0;
        if (y + size <= 9)
        {
            endingIndex = size + 1;
        }
        if(y>0)
        {
            startingIndex = - 1;
        }
        for (int i = startingIndex; i < endingIndex; i++)
        {
            if (grid[y+i][x] != 0)
            {
                return 0;
            }
            if (x > 0)
            {
                if (grid[y+i][x - 1] != 0)
                {
                    return 0;
                }
            }
            if (x < 9)
            {
                if (grid[y + i][x + 1] != 0)
                {
                    return 0;
                }
            }
        }
        return 1;
    }
    else
        return 0;
}
```

9. void *printShipPlacement:*

Marks the grid with the players ship placements.

```
void printShipPlacements(int grid[10][10]) //print ships placed on the board
{
    printf("\t   1  2  3  4  5  6  7  8  9  10\n");
    printf("\t +---------------------------------+\n");

    for (int i = 0; i < 10; i++)
    {
        printf("%c\t|  ", getLetter(i));
        for (int j = 0; j < 10; j++)
        {
            if (grid[i][j] == 1 || grid[i][j] == 2)
            {
                printf("X  ");
            }
            else
            {
                printf("   ");
            }
        }
        printf("  |\n");
    }
    printf("\t +---------------------------------+\n");

}
```

10.void *takeTurn:*

The *takeTurn* function is responsible for fetching the attack coordinates from the player and returning a statement with a status of the hit or the validity of the chosen coordinates. The status could be miss or hit with sunk being specified when it occurs after a hit. The return statement will also declare when the coordinates have been chosen before in a previous turn.

```
void takeTurn(int opponentGrid[10][10], int opponentID, int randomFlag)
{
    int found = 1;

    while (found == 1) //continue if the user keeps guessing correctly
    {
        char letter;
        int x, y;
        if (randomFlag == 0) //user will play
        {
            printf("Enter location for guess:");

            scanf("%c", &letter); //random '\n' skipped
            scanf("%c", &letter); //get letter
            y = getNumber(letter); //convert letter into index
            int num;
            scanf("%d", &num); //get x location
            x = num - 1; //subtract by one to convert to corresponding index in the grid
            printf("\n\n");
        }
        else //computer will play
        {
            x = rand() % 10;
            y = rand() % 10;
            printf("The Computer's guess was:%c%d\n\n",getLetter(y),x+1);
        }

        if (opponentGrid[y][x] == opponentID)// ship/part of ship found
        {
            opponentGrid[y][x] = -1; //ship found
            printf("Ship found in this location! There will be another turn\n\n");
            found = 1;

            if (checkForSunk(opponentGrid, x, y, opponentID) == 1) //checking if the entire ship has sunk or not
            {
                printf("Opponent ship sunk!\n\n");
            }
        }
        else if (opponentGrid[y][x] == -2 || opponentGrid[y][x] == -1) //already explored area
        {
            printf("Area already explored! Guess another location.\n\n");
            found = 1;
        }
        else
        {
            opponentGrid[y][x] = -2; //explored area but ship not found
            printf("No ship found in this location!\n\n");
            found = 0;
        }
```

11. int *checkForSunk:*

The *checkForSunk* function is responsible for indicating when the hit coordinate is the final part of a ship. By scanning in all horizontal and vertical directions for a ship part we can declare if the final part has been found when we reach an empty coordinate.

```
int checkForSunk(int grid[10][10], int x, int y, int opponentID) //check whether opponent ship has sunk or not
{
    int resume = 1;

    if (x > 0)//check if left exists
        {
            if (grid[y][x - 1] == opponentID) //check left for unsunk part
            {
                return 0;
            }
            else if (grid[y][x - 1] == -1) //if some part already sunk
            {
                int i = 2;
                while (x-i >= 0)
                {
                    if (grid[y][x - i] == opponentID)   //keep going in that directionand keep checking if there is any unsunk part available or not
                    {
                        return 0;
                    }
                    if (grid[y][x - i] == 0) //end of ship found since each ship is sepearted by atleast one space which is represented by 0
                    {
                        break;
                    }
                    i++;
                }
            }
        }
```

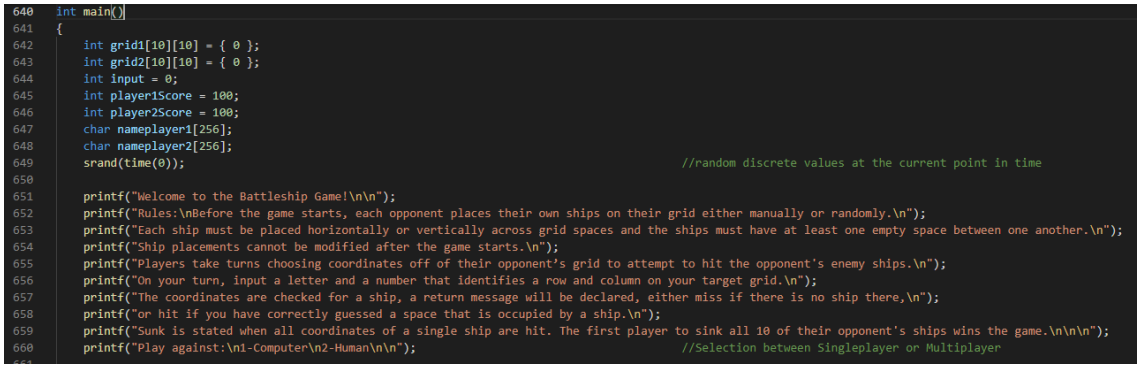
12. int *checkForWinner:*

The *checkForWinner* function scans the specified grid for any remaining ships. If no remaining ships are localised on the grid, a winner is declared.

```
int checkForWinner(int grid[10][10]) //checking if any player has won the game
{
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (grid[i][j] != 0 && grid[i][j]!= -1 && grid[i][j] != -2) //if 1 or 2 isn't present in the entire grid that means one of the player has won
            {
                return 0;
            }
        }
    }
    return 1;
}
```

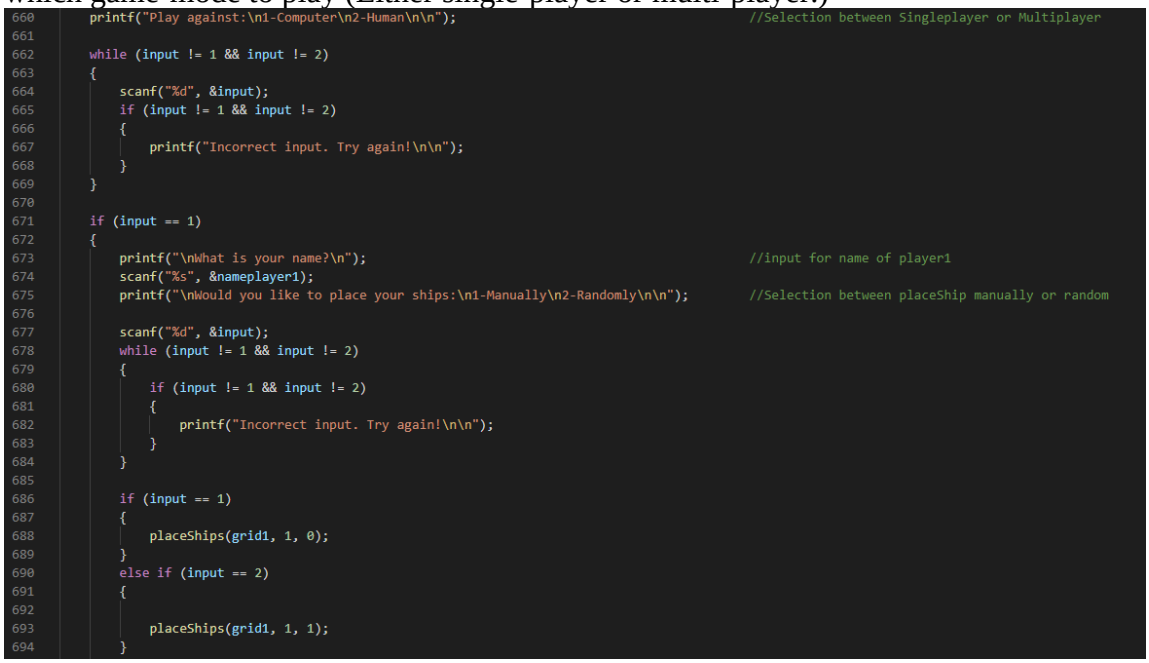
13. int *main:*

The *main* function's purpose is to run the actual game and establish connections between the functions.



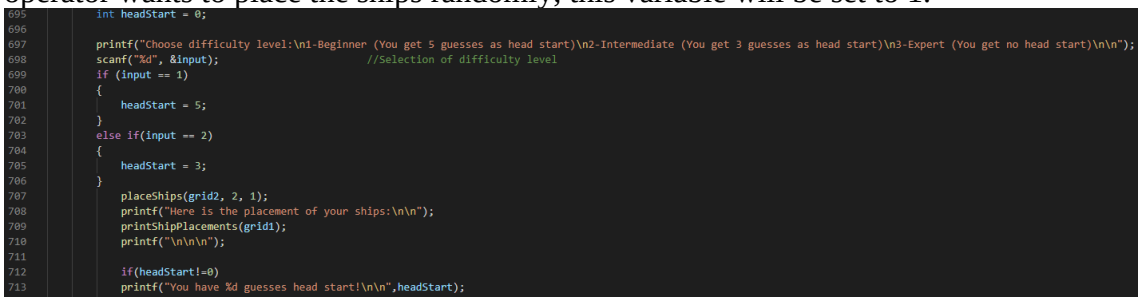
```
int main()
{
    int grid1[10][10] = { 0 };
    int grid2[10][10] = { 0 };
    int input = 0;
    int player1Score = 100;
    int player2Score = 100;
    char nameplayer1[256];
    char nameplayer2[256];
    srand(time(0));                                                    //random discrete values at the current point in time

    printf("Welcome to the Battleship Game!\n\n");
    printf("Rules:\nBefore the game starts, each opponent places their own ships on their grid either manually or randomly.\n");
    printf("Each ship must be placed horizontally or vertically across grid spaces and the ships must have at least one empty space between one another.\n");
    printf("Ship placements cannot be modified after the game starts.\n");
    printf("Players take turns choosing coordinates off of their opponent's grid to attempt to hit the opponent's enemy ships.\n");
    printf("On your turn, input a letter and a number that identifies a row and column on your target grid.\n");
    printf("The coordinates are checked for a ship, a return message will be declared, either miss if there is no ship there,\n");
    printf("or hit if you have correctly guessed a space that is occupied by a ship.\n");
    printf("Sunk is stated when all coordinates of a single ship are hit. The first player to sink all 10 of their opponent's ships wins the game.\n\n\n");
    printf("Play against:\n1-Computer\n2-Human\n\n");                  //Selection between Singleplayer or Multiplayer
```

The program starts by printing out the ruleset. After that it will prompt the user to choose which game-mode to play (Either single-player or multi-player.)



```
    printf("Play against:\n1-Computer\n2-Human\n\n");                  //Selection between Singleplayer or Multiplayer

    while (input != 1 && input != 2)
    {
        scanf("%d", &input);
        if (input != 1 && input != 2)
        {
            printf("Incorrect input. Try again!\n\n");
        }
    }

    if (input == 1)
    {
        printf("\nWhat is your name?\n");                               //input for name of player1
        scanf("%s", &nameplayer1);
        printf("\nWould you like to place your ships:\n1-Manually\n2-Randomly\n\n");       //Selection between placeShip manually or random

        scanf("%d", &input);
        while (input != 1 && input != 2)
        {
            if (input != 1 && input != 2)
            {
                printf("Incorrect input. Try again!\n\n");
            }
        }

        if (input == 1)
        {
            placeShips(grid1, 1, 0);
        }
        else if (input == 2)
        {

            placeShips(grid1, 1, 1);
        }
```

If single player is chosen, the program will ask for the player's name. Then the player needs to choose between manual or random ship placement. If the player chooses to place the ship manually the *randomFlag* variable inside the *placeShip* function will be set to 0 and if the operator wants to place the ships randomly, this variable will be set to 1.



```
        int headStart = 0;

        printf("Choose difficulty level:\n1-Beginner (You get 5 guesses as head start)\n2-Intermediate (You get 3 guesses as head start)\n3-Expert (You get no head start)\n\n");
        scanf("%d", &input);                        //Selection of difficulty level
        if (input == 1)
        {
            headStart = 5;
        }
        else if(input == 2)
        {
            headStart = 3;
        }
            placeShips(grid2, 2, 1);
            printf("Here is the placement of your ships:\n\n");
            printShipPlacements(grid1);
            printf("\n\n\n");

            if(headStart!=0)
            printf("You have %d guesses head start!\n\n",headStart);
```

After the ships are placed, it will be asked which game difficulty the player wants to play in the beginning of the game. When entering a 1 for beginner level the variable *headStart* will be set to 5, which means that the operator gets five head starts. For the intermediate level, the

operator will be given three head starts and for expert level zero head starts are given. Then it will place the ships for the bot with the *placeShip* function's RandomFlag set to 1. After that the player's grid will be printed with the already done ship placements.

```
715         int resume = 1;
716         while (resume)
717         {
718             printf("Player 1's current guesses:\n\n");
719             printGrid(grid2, 1);
720             printf("It is the turn of Player 1 now...\n\n");
721             takeTurn(grid2, 2, 0);
722             if (checkForWinner(grid2) == 1)
723             {
724                 printf("Player 1 has sunk all the ships of the Computer and has won the game!\n\n");
725                 FILE* fptr = fopen("Score.txt", "a");                                   //Save the result in a .txt file
726                 fprintf(fptr, "%s\t%d\n", nameplayer1, player1Score);
727                 fclose(fptr);
728                 break;
729             }
730             system("pause");
731             system("cls");
732             if (headStart == 0)
733             {
734                 printf("\nComputer's current guesses:\n\n");
735                 printGrid(grid1, 2);
736                 printf("It is Computer's turn now...\n\n\n");
737                 takeTurn(grid1, 1, 1);
738                 if (checkForWinner(grid1) == 1)
739                 {
740                     printf("Computer has sunk all the ships of Player 1 and has won the game!\n\n");
741                     break;
742                 }
743                 system("pause");
744                 system("cls");
745             }
746             else
747             {
748                 headStart--;
749             }
750             player1Score--;
751         }
```

If both grids are ready to be played on, the player and the computer take turns choosing coordinates, until all ships on a single matrix are sunk. For this operation, the functions *printGrid* and *takeTurn* are used in a while loop, that only stops when *checkForWinner* for the bot or the operator equals one which means that this player has won the game. (The value of the headstart variable is deducted by 1 for every turn the player misses a ship, after which the computer begins taking its turn)

```
725                 FILE* fptr = fopen("Score.txt", "a");
726                 fprintf(fptr, "%s\t%d\n", nameplayer1, player1Score);
727                 fclose(fptr);
```

If the player wins, the score, and the name of the player will be saved in a separate text file. All player scores are initialized at 100 and are reduced for every turn taken.

```
755     else if (input == 2)
756     {
757         srand(time(0));
758         printf("\nPlayer 2 kindly move away from the screen till Player 1 is setting up his ships!\n\n");
759         printf("What is your name?\n");
760         scanf("%s", &nameplayer1);
761         printf("\nPlayer 1 would you like to place your ships:\n1-Manually\n2-Randomly\n\n");
762 
763         scanf("%d", &input);
764         while (input != 1 && input != 2)
765         {
766             if (input != 1 && input != 2)
767             {
768                 printf("Incorrect input. Try again!\n\n");
769             }
770         }
771 
772         if (input == 1)
773         {
774             placeShips(grid1, 1, 0);
775         }
776         else if (input == 2)
777         {
778 
779             placeShips(grid1, 1, 1);
780         }
781 
782         printf("Here is the placement of your ships:\n\n");
783         printShipPlacements(grid1);
784 
785         system("pause");
786         system("cls");
```

If the user chooses to play multiplayer, first the username of player1 will be requested. Then, how he/she wants to place the ships. This works in the same way as it works for the single player mode.

```
788         printf("\nPlayer 1 kindly move away from the screen till Player 2  is setting up his ships!\n\n");
789         printf("What is your name?\n");
790         scanf("%s", &nameplayer2);
791         printf("\nPlayer 2 would you like to place your ships:\n1-Manually\n2-Randomly\n\n");
792 
793         scanf("%d", &input);
794         while (input != 1 && input != 2)
795         {
796             if (input != 1 && input != 2)
797             {
798                 printf("Incorrect input. Try again!\n\n");
799             }
800         }
801 
802         if (input == 1)
803         {
804             placeShips(grid2, 1, 0);
805         }
806         else if (input == 2)
807         {
808             placeShips(grid2, 1, 1);
809         }
810 
811         printf("Here is the placement of your ships:\n\n");
812         printShipPlacements(grid2);
813 
814         system("pause");
815         system("cls");
```

After player1 has placed his/her ships, player2 needs to do the same steps. This also works in the same way as it works for the single-player mode.

```
818         int resume = 1;
819         while (resume)
820         {
821             printf("Player 1's current guesses:\n\n");
822             printGrid(grid2, 1);
823             printf("It is the turn of Player 1 now...\n\n");
824             takeTurn(grid2, 2, 0);
825             if (checkForWinner(grid2) == 1)
826             {
827                 printf("Player 1 has sunk all the ships of Player 2 and has won the game!\a\n\n");
828                 FILE* fptr = fopen("Score.txt", "a");                                   //Save the result in a .txt file
829                 fprintf(fptr, "%s\t%d\n", nameplayer1, player1Score);
830                 fclose(fptr);
831                 break;
832             }
833             system("pause");
834             system("cls");
835 
836                 printf("\nPlayer 2's current guesses:\n\n");
837                 printGrid(grid1, 2);
838                 printf("It is Player 2's turn now...\n\n\n");
839                 takeTurn(grid1, 1, 0);
840                 if (checkForWinner(grid1) == 1)
841                 {
842                     printf("Player 2 has sunk all the ships of Player 1 and has won the game!\a\n\n");
843                     FILE* fptr = fopen("Score.txt", "a");                               //Save the result in a .txt file
844                     fprintf(fptr, "%s\t%d\n", nameplayer2, player2Score);
845                     fclose(fptr);
846                     break;
847                 }
848                 system("pause");
849                 system("cls");
850 
851             player1Score--;
852             player2Score--;
853         }
854 
855     }
856 
857 }
```

After both grids are ready to be played on, the players take turn choosing coordinates of off each other's grids until all ships on a single grid are sunk. The winner's name and score will be saved in a text file.