# MID/MIS Documentation

Owen Huyn
Hun Ko
Shane Noormohamed
Matthew Schweitzer
Howin Tsui

# MIS:

## Game Logic Functions

**isSunk(x: IN int; y:IN int; OUT int)**
Checks whether the ship at coordinates Ax,By is sunk.
Output -1 if there is no ship, or if it is not sunk.
Output the integer representing the ship if it is sunk.

**reader(fileName: IN string, stats: OUT vector<string>)**
reads from a users stats file and returns it line by line as a vector of strings.

**statistics(playerName: IN string, name: IN string, playerHitPercent: IN double, totalShots: IN int, score: IN int)**
appends a users current statistics to their text file.

**HitPercent(load: IN string, HitPercent: OUT string)**
Reads from a users save file and returns the users hit percent.

**totalShots(load: IN string, totalShots: OUT string)**
Reads from a users save file and returns how many shots the user have taken.

**addShip(ship: IN int; orientation: IN int; x: IN int; y:IN int; OUT bool)**
Check if the inputs are valid, and that the proposed ship does not exceed the dimensions of the grid. If the checks are passed, then place a ship at the specified location on the grid, and return true. Otherwise, return false.
The *orientation* variable represents horizontal when 0, and vertical when 1.
The *ship* variable represents the index of the ship to be placed, not the length of the ship.

**shoot(x: IN int; y:IN int; OUT int)**
Attempt to shoot at the coordinates Ax,By of the current grid.
return 3 if the shot misses
return 2 if the shot hits
return 0 if the location has already been shot at.

**getBlock(x: IN int; y:IN int; OUT int)**
return the value of the grid at x,y.

**clear()**
reset all of the grid's values to 0.

**Grid::AIsetup()**
This method will setup the grid for the computer, which will allow the user to shoot at ships. It will generate 7 ships with blocks ranging from 1 to 5. It will randomize a position and direction for each ship. Once it is complete, the game may proceed.

**sunkenShips(load: IN string, sunkenShips: OUT string)**
Reads from a users save file and returns how many ships have been sunk for both the player and computer.

**addShipBlock(x: IN int; y:IN int)**
Check to make sure x and y are within range, and set the value of the grid at Ax,By
to 1 in order to represent a ship.

***save*(save: IN string; score: IN integer)**
Saves a users game state so it can be loaded at a later time.

***load*(load: IN string, Score: OUT string)**
Loads a users saved game state allowing them to start their game back at the point where they last saved.

## User Interface Functions

**IsVerticalCB**
*-checkbox1_checkedChanged()*
This function allows the users to change the orientation of the ships that they are placing

**a_PlacePos_textChanged() & b_PlacePos_textChanged()**
If the user enters an invalid input into the textbox, inputs in each textbox turn red and the "Place Ship" button becomes not clickable.
Input: User Input
Output: "Place Ship" Button becomes enabled or a_PlacePos and b_PlacePos's forecolor becomes red if the inputs are invalid.

**a_AttackPos_textChanged & b_AttackPos _textChanged**
If the user enters an invalid input into the textbox, inputs in each textbox turn red and the "Hit!" button becomes not clickable.
Input: User Input
Output: "Hit!" Button becomes enabled or textBox1's forecolor and textBox2's forecolor becomes red

**ShootShipButton_click()**
This function takes user inputs from textbox1 and textbox2 and fires at the specified position. After that, it will call upon the AI function and update the grid. This function also calculates the score and will notify the user when the game is over.

**PlaceShipButton_click()**
This function takes user input from a_PlacePos, b_placePos, and  isVerticalCB to place ships in the specified location.

**Help button ( HelpButton_click() )**
This function will open up a help screen (form2.h) when the button is clicked.

**Restart button ( ResetButton_click() )**
This function will close and reopen the program when the button is clicked.

**Save button ( SaveButton_click() )**
This function will save the present game when the button is clicked.

**Load( LoadButton_click() )**
This function will load a specified game when the button is clicked.

**Stats button (Stats_click())**
Graphical user interface button loads the statistics page.

**Stats Button(StatsButton_click()**
Graphical user interface button that loads a players previous stats.

# MID:

## User Interface Functions

**isSunk(x: IN int; y:IN int; OUT int)**

First, check that the block being requested is hit (value of 2). If it is not, then return -1 to indicate that there is no sunk ship at the requested location.

```
If the requested block is indeed hit, then,
Using a for loop to run through each of the ships,
     check if the orientation of the ship is horizontal, if it is, then
          check if the current ship, as specified by the for loop, is the same one the user is asking for, if it is
              run through each block of the ship, and check if it is a hit.
                  if any of them are not a "hit", return -1;
                  end if
              end for
         the code getting to this point means that the ship is sunk, so return the index of the outer for loop, . representing the ship that is sunk
    if the orientation is vertical, then
        do all the same as above, except checking vertically instead of horizontally
    end if
end for
if the code gets to this point, it means there is no sunk ship at the requested location, therefore,
return -1
```

**addShip(ship: IN int; orientation: IN int; x: IN int; y:IN int; OUT bool)**

Steps: Check orientation, check if coordinates fit in the grid, check to make sure there are no ships already placed in the way of the proposed ship. If all these checks are passed, place the proposed ship. Output success(true) or failure(false).

```
If the orientation is horizontal
     If the coordinates proposed, added to the size of the ship, fit on the grid
        Create a for loop that runs for the size of the ship(counter i)
            if the block at (x+I,y) is not empty,
               return false
            end if
        end for
        the code getting to this point means that all checks have been passed.
        Using a for loop that runs for the size of the ship(counter i)
           place a ship block, using addShipBlock(x+I,y)
        end for
     the code getting to this point means that a ship has been placed successfully
     return true
  end if
otherwise, if the orientation is vertical,
  do the above block, calculating for vertical placement instead of horizontal
  (by replacing x+I with x, and y with y+i)
```

shoot(x: IN int; y:IN int; OUT int)

Check the value of the grid at AxBy and alter it accordingly. If it is a ship(1), set it to hit(2) and return 2. If it is empty(0), set it to miss(3) and return 3. If the location has already been shot at (hit or miss), then return 0, signifying that the shot failed.

getBlock(x: IN int; y:IN int; OUT int)

Look at grid location AxBy and return the value

clear()

Using nested for loops, run through the grid and reset all values to 0.

## **reader():**

Reads a users stats file and returns line by line all of their stats(score, name, hit percent, shots fired)

Inputs:

String fileName(name of the file to load)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| lines | - | integer | state |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| stats | - | Vector<string> | state |

| | |
|---|---|
| | if (myfile.is_open()) |
| | getline(myfile,line) |
| stats | stats.push_back(line) |

## **statistics():**

saves a users name, hit percent, total shots fired and score to their statistics text file so they can view their previous scores at a later time.

Inputs:

String playerName(name of the file to load)
String name(name of the player)
double playerHitPercent(user's hit percent)
integer totalShots(how many shots the user has taken)
integer score(the users current score)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| - | - | - | - |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| - | - | - | - |

| | | | | |
|---|---|---|---|---|
| if (myfile.is_open()) | | | | |
| myFile | Store playerName | Store HitPercent | Store Total shots fired | Store score |

### HitPercent():

Reads through the users save file and returns how often the user hit an AI ship in percent.

Inputs:

String load(name of the file to load)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| lines | - | integer | state |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| HitPercent | - | double | state |

<table>
<tr><td rowspan="4"></td><td colspan="2">if (myfile.is_open())</td></tr>
<tr><td colspan="2">for(int lines = 0; getline(myfile,line) &amp;&amp; lines &lt; 12; lines++)</td></tr>
<tr><td>if(lines == 10)</td><td>If(lines&lt;10|| lines &gt;10)</td></tr>
<tr><td>Double HitPercent = atoi(line.c_str())</td><td rowspan="2">Does nothing</td></tr>
<tr><td>HitPercent</td><td>Return HitPercent</td></tr>
</table>

### sunkenShips()

Reads through the users save file for both player and AI and returns the number of ships sank for each player.

Inputs:

String load(name of the file to load)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| lines | - | integer | state |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| sunkenShips | - | integer | state |

<table>
<tr><td rowspan="4"></td><td colspan="2">if (myfile.is_open())</td></tr>
<tr><td colspan="2">for(int lines = 0; getline(myfile,line) &amp;&amp; lines &lt; 10; lines++)</td></tr>
<tr><td>if(lines == 9)</td><td>If(lines&lt;9 || lines &gt;9)</td></tr>
<tr><td>int sunkenShips = atoi(line.c_str())</td><td rowspan="2">Does nothing</td></tr>
<tr><td>sunkenShips</td><td>Return sunkenShips</td></tr>
</table>

### ***Grid::Save()***

Is used on a player or AI grid and ship array. For each ship in the array it copies down the specifications of each ship(size and where they are placed) and saves them to a text file along with a copy of the grid including all the states of each grid block(hit, miss, ship, no ship). Also stores the players current score, amount of ships sunk, hit percent, and total shots fired.

Inputs:

String save(name of the save file)
int score(the players current score in the game)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| score | - | integer | State |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| - | - | - | - |
| - | - | - | - |

Range Checks:

if(ships[i][1] == 0){ships[i][1] = 58}
if(ships[i][2] == 0){ships[i][2] = 58
if(score>topScore){myfile << score} else{myfile << topScore}

<table>
<tr><td></td><td colspan="4">myfile.open(save+".txt")</td></tr>
<tr><td rowspan="3"></td><td colspan="2">for (int i = 0; i < 7);</td><td>for(int i = 0; i < 10)</td><td rowspan="2"></td></tr>
<tr><td>if(ships[i][1] == 10)</td><td>if(ships[i][2] == 10)</td><td>for(int j = 0; j < 10)</td></tr>
<tr><td>ships[i][1] = 0</td><td>ships[i][2] = 0</td><td rowspan="2">myfile << grid[i][j]</td><td rowspan="2">myfile << score;<br>myfile << shipsSunk<<endl;<br>myfile << HitPercent<< endl;<br>myfile <<totalShots<<endl;</td></tr>
<tr><td>myfile</td><td colspan="2">myfile << ships[i][0];<br>myfile << ships[i][1];<br>myfile << ships[i][2] << endl;</td></tr>
</table>

### **totalShots():**

Reads through the users save file and returns how many shots the user has taken.

Inputs:

String load(name of the file to load)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| lines | - | integer | state |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| totalShots | - | int | state |

<table>
<tr><td rowspan="4"></td><td colspan="2">if (myfile.is_open())</td></tr>
<tr><td colspan="2">for(int lines = 0; getline(myfile,line) && lines < 12; lines++)</td></tr>
<tr><td>if(lines == 11)</td><td>If(lines<11|| lines >11)</td></tr>
<tr><td>int totalShots = atoi(line.c_str())</td><td rowspan="2">Does nothing</td></tr>
<tr><td>totalShots</td><td>Return totalShots</td></tr>
</table>

**Grid::Load()**

Calls a text file specified by the user and checks if that file exists. If the text file does not exist displays a message telling the user to retry. If the file exists it reads through the file line by line. For the first 7 lines it inputs the values into the addShip() function to place the loaded ships on the grid.  It reads the 8th line in the file which contains the grid state and sets the current grid state so that it is the same as the grid loaded(shoots the current grid in appropriate places).
The last line in the file is the users score which it reads and inputs into the score box on the windows form.

Inputs:

String load(name of the file to load)

Updates:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| lines | - | integer | state |

Outputs:

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| score | - | integer | state |
| addShip | - | integer array | state |
| shoot | - | integer | state |

Range Checks:

```
if(myfile.is_open()){
if((line[1] - 48) == 0){line[1] =58}
if((line[2] - 48) == 0){line[2] =58}
if((line[k]-48) == 2 || (line[k]-48) == 3){shoot(j,i); k++;}
else if((line[k]-48) == 0 || (line[k]-48) == 1){k++;}
else {return -1;}
```

<table>
<tr><td rowspan="4"></td><td colspan="3">if (myfile.is_open())</td></tr>
<tr><td colspan="3">for(lines = 0; getline(myfile,line) &amp;&amp; lines &lt; 8; lines++)</td></tr>
<tr><td colspan="3">if(lines &lt; 7)</td></tr>
<tr><td>if((line[1] - 48) == 0)</td><td>if((line[2] - 48) == 0)</td><td rowspan="3">addShip(lines,line[0]-48,line[1]-48,line[2]-48);</td></tr>
<tr><td></td><td>line[1] = 58;</td><td>line[2] = 58</td></tr>
<tr><td>addShip</td><td colspan="2">addShip(lines,line[0]-48,line[1]-48,line[2]-48);</td></tr>
</table>

<table>
<tr><td rowspan="4"></td><td colspan="2">if (myfile.is_open())</td></tr>
<tr><td colspan="2">if(lines == 7)</td></tr>
<tr><td colspan="2">for (int i = 1; i &lt; 11; i++) &amp;&amp; for (int j = 1; j &lt; 11; j++)</td></tr>
<tr><td>if((line[k]-48) == 2 || (line[k]-48) == 3)</td><td>if((line[k]-48) == 0 || (line[k]-48) == 1)</td></tr>
<tr><td>shoot()</td><td>shoot(j,i); k++;</td><td>K++;</td></tr>
</table>

**Grid Class**

The grid class uses two private 2-dimensional vectors in order store the data needed for the class to function.  The first 2D vector represents the grid on which the ships are placed, and shots are fired.  The second 2D vector represents the seven ships, with ships[x][0] representing the $x^{th}$ ship's orientation, ship[x][1] representing the top-left x position, and ship[x][2] representing the top-left y position.

**Grid::AIsetup()**

This method will setup the AI grid and will randomize locations on where ships can be placed. It will also randomize the direction that the ships will be set on. The boundaries will be set from 1 to 10 so that the ships that are placed on the two-dimensional array will not be placed out of bounds. Additionally, it will use the C++ time clock to completely randomize starting coordinates. Lastly, the origin of the ship will be on the top-left sector. This means that the carrier which is 5 blocks will have its pivot point set on the most left or the most upper block. Therefore, the pivot point cannot be set at any point past x = 6 on the left-right direction and y = 6 on the up-down direction. This is the same for all the other ships.

<u>INPUTS</u>

NONE

<u>UPDATES</u>

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| addShip(0,rand()%2,xCoor,yCoor) | - | item | State |
| addShip(1,rand()%2,xCoor,yCoor) | - | item | State |
| addShip(2,rand()%2,xCoor,yCoor) | - | item | State |
| addShip(3,rand()%2,xCoor,yCoor) | - | item | State |
| addShip(4,rand()%2,xCoor,yCoor) | - | item | State |
| addShip(5,rand()%2,xCoor,yCoor) | - | item | State |
| addShip(6,rand()%2,xCoor,yCoor) | - | item | State |

where xCoor and yCoor are randomized from 1 to 10

<u>OUTPUTS</u>

NONE

<u>RANGE CHECKS:</u>

| | True | False |
|---|---|---|
| !addShip(6,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | Carrier Created |
| !addShip(5,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | 4 block ship made |
| !addShip(4,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | 3 block ship made |
| !addShip(3,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | First 2 block ship made |
| !addShip(2,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | Second 2 block ship made |
| !addShip(1,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | First 1 block ship made |
| !addShip(0,rand()%2,xCoor,yCoor) | Randomize a new xCoor, yCoor | Second 1 block ship made |

**int* Grid::MediumAIshoot()**
This method will activate an algorithm that will determine where the computer will want to shoot on the grid. It will first randomize a coordinate within the grid to shoot at. It will activate the shoot method in the class, which will activate a comparison to determine whether the shot missed, hit or sunk a target. If a coordinate has been hit, the algorithm will search for that position and proceed to fire one of the 4 positions surrounding that spot. If a two consecutive spots have been hit, it will proceed to fire down the same line until the ship has been sunk. Once the ship has been sunk, it will completely randomize a new position if there are no hit markers with non-sunk ships. The algorithm will repeat until either the user or computer wins. Additionally the method will return an array indicating that positions it has last hit. This is needed to properly make the isSunk method work. Whenever a shot has been fired, it will update the user's 2-dimensional grid by changing a unfired coordinate or ship coordinate to a missed coordinate or hit coordinate.

INPUTS

NONE

UPDATES

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| shoot(i-1,j) | - | item | State |
| shoot(i+1,j); | - | item | State |
| shoot(i,j+1) | - | item | State |
| shoot(i,j-1) | - | item | State |
| shoot(xCoor, yCoor) | - | item | State |

OUTPUTS

| Name | Ext_Value | Type | Origin |
|---|---|---|---|
| returner | - | array | DigitalOutput |

RANGE CHECKS:

| | True | False |
|---|---|---|
| if ((getBlock(i+1,j) == 2) && ((getBlock(i-1,j) == 0) \|\| (getBlock(i-1,j) == 1))) | Shoots in the left position in two consecutive spaces | No change |
| if ((getBlock(i-1,j) == 2) && ((getBlock(i+1,j) == 0) \|\| (getBlock(i+1,j) == 1))){ | Shoots in the right position in two consecutive spaces | No change |
| if ((getBlock(i,j-1) == 2) && ((getBlock(i,j+1) == 0) \|\| (getBlock(i,j+1) == 1))){ | Shoots in the upper position in two consecutive spaces | No change |
| if ((getBlock(i,j+1) == 2) && ((getBlock(i,j-1) == 0) \|\| (getBlock(i,j-1) == 1))){ | Shoots in the lower position in two consecutive spaces | No change |
| if ((getBlock(i+1,j) == 0) \|\| (getBlock(i+1,j) == 1)){ | Shoots in the right position off a hit marker | No change |
| if ((getBlock(i-1,j) == 0) \|\| (getBlock(i-1,j) == 1)){ | Shoots in the left position off a hit marker | No change |
| if ((getBlock(i,j+1) == 0) \|\| (getBlock(i,j+1) == 1)){ | Shoots in the upper position off a hit marker | No change |
| if ((getBlock(i,j-1) == 0) \|\| (getBlock(i,j-1) == 1)){ | Shoots in the lower position off a hit marker | No change |
| !(getBlock(xCoor,yCoor)==0) && !(getBlock(xCoor,yCoor)==1) | If found that the block contains a hit or miss marker, then it will randomize another coordinate | It will proceed to fire at that coordinate |

## Game Logic Functions

### a_PlacePos_textChanged()

Tracks if the user enters a valid input for textBox3. If input is valid, the "Place Ship" button becomes enabled(clickable), otherwise the inputs for textBox3 and textBox4 turn red.

```
Function textBox3_textChanged(User Input)
if textBox3.text equals 1,2,3,4,5,6,7,8,9,10 then
        textBox3.foreCOlor = Black
        textBox4.foreColor = Black
        button2.Enabled = true
Else
        textBox3.foreCOlor = Red
        textBox4.foreColor = Red
        button2.Enabled = false
end if
end function
```

### b_PlacePos_textChanged()

Tracks if the user enters a valid input for textBox4. If input is valid, the "Place Ship" button becomes enabled(clickable), otherwise the inputs for
textBox3 and textBox4 turn red.

```
Function textBox4_textChanged(User Input)
if textBox4.text equals 1,2,3,4,5,6,7,8,9,10 then
        textBox4.foreCOlor = Black
        textBox3.foreColor = Black
        button2.Enabled = true
Else
        textBox4.foreCOlor = Red
        textBox3.foreColor = Red
        button2.Enabled = false
end if
end function
```

### a_AttackPos_textChanged

Tracks if the user enters a valid input for textBox1. If input is valid, the "Place Ship" button becomes enabled(clickable), otherwise the inputs for textBox1 and textBox2 turn red.

```
Function textBox1_textChanged(User Input)
if textBox1.text equals 1,2,3,4,5,6,7,8,9,10 then
        textBox1.foreColor = Black
        textBox2.foreColor = Black
        button1.Enabled = true
Else
        textBox1.foreCOlor = Red
        textBox2.foreColor = Red
        button1.Enabled = false
end if
end function
```

### b_AttackPos_textChanged

Tracks if the user enters a valid input for textBox2. If input is valid, the "Place Ship" button becomes enabled(clickable), otherwise the inputs for textBox1 and textBox2 turn red.

```
Function textBox2_textChanged(User Input)
if textBox2.text equals 1,2,3,4,5,6,7,8,9,10 then
        textBox1.foreCOlor = Black
        textBox2.foreColor = Black
        button1.Enabled = true
Else
        textBox1.foreCOlor = Red
        textBox2.foreColor = Red
        button1.Enabled = false
end if
end function
```

### IsVerticalCB (IsVerticalCB_CheckedChanged() )

This function uses a simple boolean equation to change the value that the function returns

### Help( HelpButton_click() )

Creates a copy of Form2 called help and then displays it to the user.

### Restart( ResetButton_click() )

Calls the Application::Restart() function to restart the form and then calls Form1::Show() to display the restarted game to the user.

### Save( SaveButton_click() )

Checks saveText->Text to see if the file name is correct. If the filename is invalid it displays a message to the user asking for a valid filename. If the filename is valid it also checks textBox7->Text to see what the users score is then calls the save() function and displays a message confirming that the file was saved.

### Load( LoadButton_click())

Checks saveText->Text to see if the file name is correct. If the filename is invalid it displays a message to the user asking for a valid filename. If the filename is valid it calls the load() function and displays a message confirming that the file was loaded. It also enables or disables specified buttons, labels and text boxes on the windows form. After the game has been loaded this button will draw rectangles overtop of the grid where shots have been placed. If it was a hit then a red box is drawn, if it was a miss a grey box is drawn. This function also stores the players statistics to variables in game as they are loaded.

### ShootShipbutton( ShootShipButton_click() )

When the button is pressed, it will first check the textboxes to check if the values are within the range. If the values are in range, it will call upon the shoot function from game logic which will update the grid. It will then call the AI function and update the grid again. This function also uses a series of if statements along with the getBlock function and isSunk function to update the status bars and the score. It also uses a conditional statement to check if any of the players win after each time the shoot button is pressed.

### PlaceShipbutton (PlaceShipButton_click() )

When the button is pressed, it will first check the textboxes to check if the values that the user inputted are within the range. It uses conditional statements with the boolean value of ‘isVertical’ checkbox to determine whether the ship is placed vertically and horizontally. It will then call upon the addShip function to add the ships to the grid and update the grid. It also contains conditional statements to disable the placeship function after all the ships have been placed.

### StatsButton_click()

Confirms that the users name is valid. If the name is valid it loads all stored statistics for that users previous games.

**Stats_click():**

displays the stats page where the user can view previous statistics of a specific player