Missouri

Final Evaluation of the semester project

Authors

Chen Shih, Giwa Haheeb Rilwan, and Phan Thanh-Long

Table of contents

1. Applied tools and techniques

1.1 Tools

To develop the game, the C programming language is used.

GitHub is heavily utilized in the whole development process for version control, coordination of efforts among developers, and file management.

The game is developed and tested on Windows 10 environment, other environments are untested and not supported.

1.2 Techniques:

1.2.1 User defined header file and built-in Libraries

For the purpose of modularization, a file called “library .h” is created to define and contain the functions for the main .c file. Furthermore, functions in built-in libraries are called in the program as well in order to perform certain executions.

1.2.2 Exploiting array to create grid systems

The grid system is made by a 10 by10 array.

- *int grid[10][10]*

  A hidden array which stores the information of players' game boards.
- *char display[10][10]*

  Visualization of the player’s game board.
- *char Mask[10][10]*

  Mask out the opponent's board and show only part of it.

1.2.3 Utilization of getch():

Getch() method pauses the Output Console until a key is pressed. It does not use any buffer to store the input character. The entered character is immediately returned without waiting for the enter key. Moreover, the entered character does not show up on the console. [1]

In Missouri, getch() receives inputs from the player and passes them to other functions for further processing. For example, choose game mode, difficulty level, and coordinates of the target.

1.2.4 rand() %

rand() % is the random number generator [2], it is used in computer's attack algorithm to randomize an attack coordinate, and it is also used to choose one template randomly out of the ones in the library.

1.2.5 system("cls")

The function "system("cls")" clears previous outputs to keep the console clean.

1.2.6 sleep()

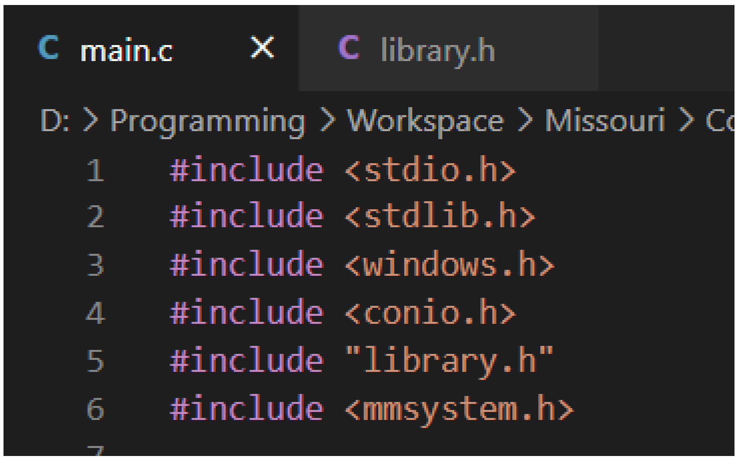
C programming language provides sleep() function in order to wait for a current thread for a specified time. Of course, the CPU and other processes will run without a problem. [3]

In Missouri, Sleep() function pauses the program to give the user some time to respond. Otherwise, the user wouldn't be able to see certain changes on the screen. For instance, the visual effect of a ship being attacked.

1.2.7 PlaySound()

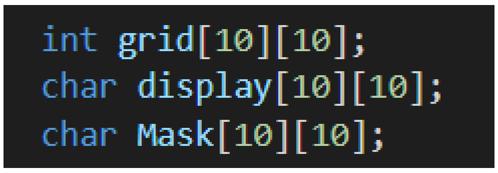
The PlaySound function plays a sound specified by the given file name, resource, or system event. [4]

PlaySound() is exploited in Missouri to create sound effects. An example is shown in fig 5. In this example, the program plays the audio file "Victory.wav" when a winner is announced.



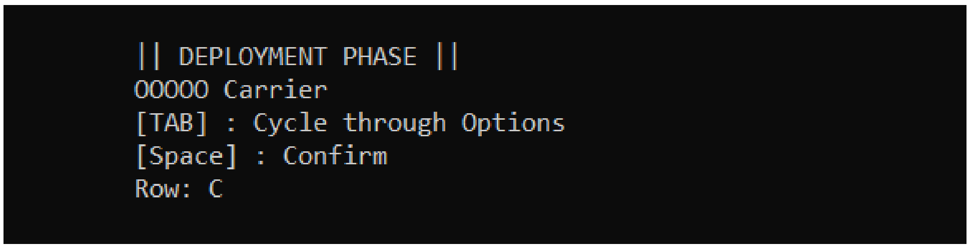
```c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>
#include "library.h"
#include <mmsystem.h>
```

Fig. 1: Utilization of .h file



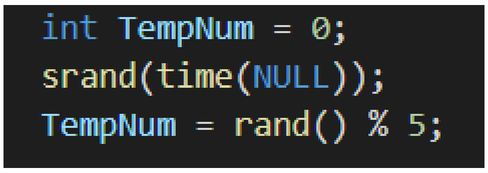
```c
int grid[10][10];
char display[10][10];
char Mask[10][10];
```

Fig. 2: Elements of a grid



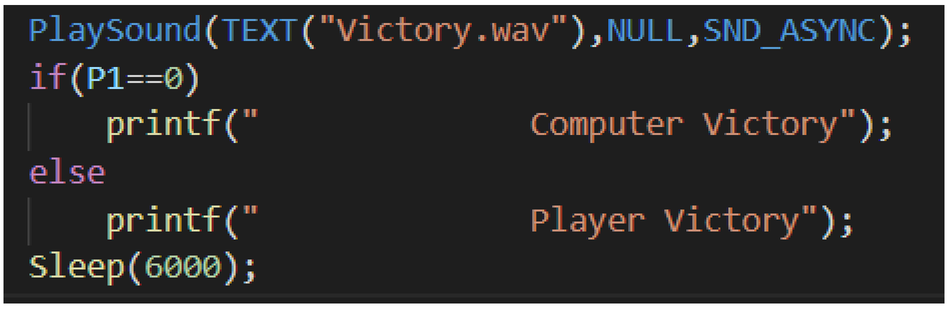
```
|| DEPLOYMENT PHASE ||
OOOOO Carrier
[TAB] : Cycle through Options
[Space] : Confirm
Row: C
```

Fig. 3: Visualization of cycling through options using getch()



```c
int TempNum = 0;
srand(time(NULL));
TempNum = rand() % 5;
```

Fig. 4: Example of rand() used



```
PlaySound(TEXT("Victory.wav"),NULL,SND_ASYNC);
if(P1==0)
    printf("              Computer Victory");
else
    printf("              Player Victory");
Sleep(6000);
```

Fig. 5: Example of PlaySound() used

2. Module structure

For modularity and ease of maintenance, the codes are divided into two parts, one is <main.c> and the other is a header file called <library.h>.

<main.c> is basically the skeleton of the program, it defines the structure and courses of workflow. All the functions in <main.c> can be considered as muscles which carry out desired actions. <library.h>, on the other hand, defines the functions and contains necessary information and variables.

This structure brings convenience of modification and troubleshooting. Functions are developed separately and integrated into the main program after complete tests, which increases the independence and effectively avoids confusion while coding. Since functions are rather independent of one another, problems are easier to be identified and fixed without the concerns of interference.

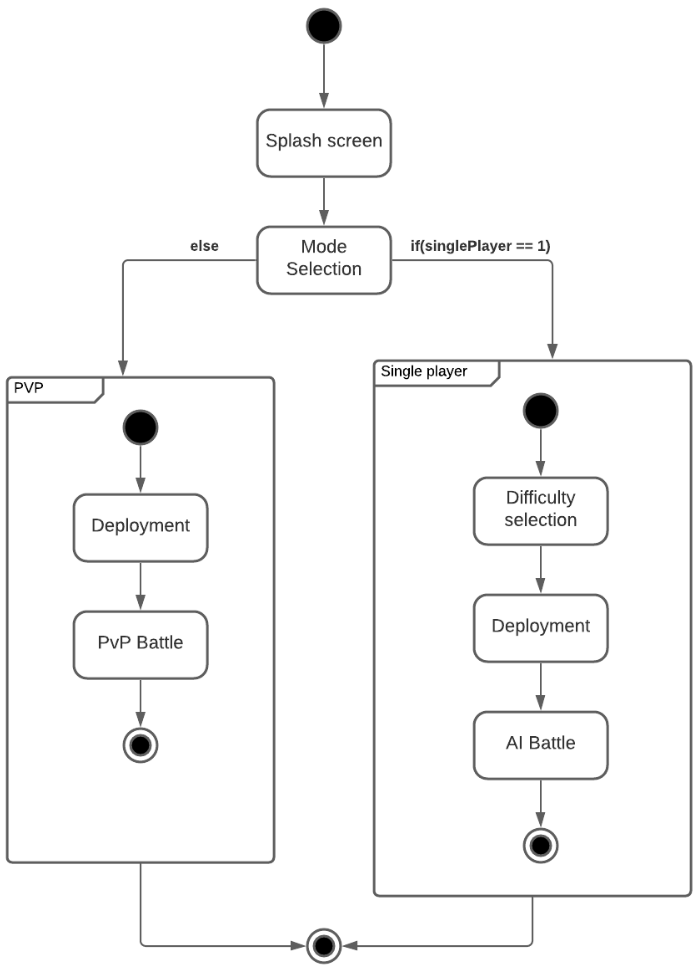


Fig 6: Overall State Machine Diagram of the Program

3. Major design decisions

In addition to default requirements, some designs are added to the game in order to improve user experience while others are applied to avoid error-prone situations, which brings the quality of the program to next level.

3.1 User interface

A list of instructions and hints are provided for the players to follow, so that even a beginner can get the hang of it quickly. The program refreshes the console constantly whenever there are updates, and it erases old outputs as well in order to keep the screen neat and organized. Moreover, pauses are made for players to respond to the game.

3.2 Mistake proofing mechanism

Instead of entering different sets of commands and inputs, players can interact with the console by simply pressing [Tab] and [Space]. This design not only brings the ease of control, but also saves developers a lot of hassles because it avoids invaild inputs which can lead to errors.

3.3 Sound effects

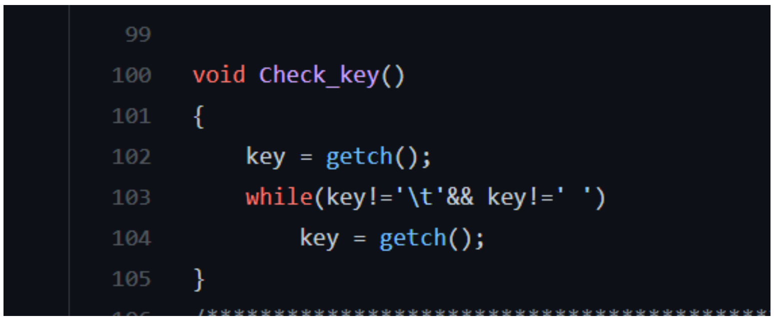
Sound effect is one of the essential elements for any successful games! Along with the pleasant visual interface, the sound effects are cherry on top.

4. Implementations in details

This section covers all the major functions in Missouri.

4.1 Check_key()

The Check_key() function consists of a while loop and a getch() function. First, the input value will be assigned to the variable “key”. Subsequently, it enters the while loop and stays until the key ‘ ’ or ‘ \ t ’ is pressed. This function effectively avoids invalid inputs.



```

void Check_key()
{
    key = getch();
    while(key!='\t'&& key!=' ')
        key = getch();
}
/*********************************************
```

Fig. 7: The Code of check_key()

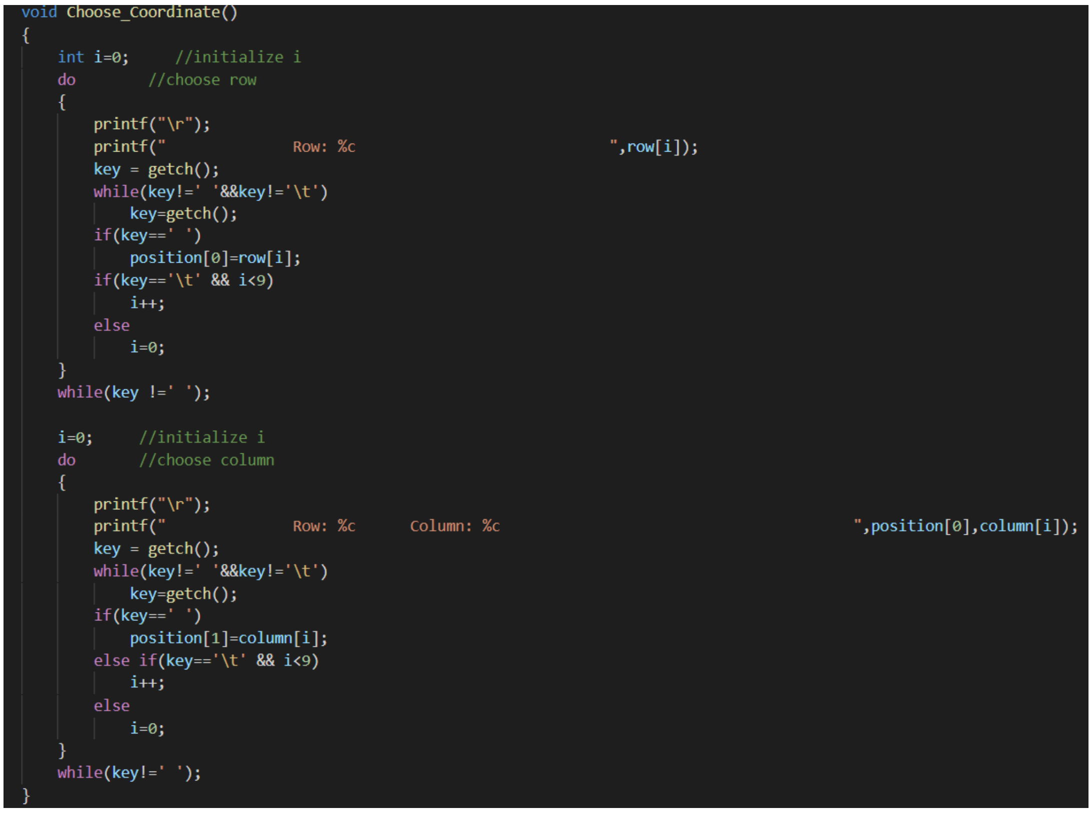
4.2 Choose_Coordinate() & Choose_Direction()

Choose_Coordinate() during ship deployment enables players to select position and heading direction of the ships. Also, this is applicable to battle mode, selection of targets can be achieved by using the same function.

- **Choose_Coordinate()**
- This function enables players to input letters (A-J) for rows, numbers from (0-9) for columns.
- There are 2 one dimensional arrays being used, one contains capital letters from A to J and the other has integer numbers 0 to 9. By cycling through these two arrays, the program can get the selected row and column for further processing.

Choose_Direction()

- With this function, players are enabled to choose the heading direction of their ships .
- In this case, a one dimensional char array which contains ”U”, “R”, “D”, and ”L” is applied.



```
void Choose_Coordinate()
{
    int i=0;     //initialize i
    do       //choose row
    {
        printf("\r");
        printf("           Row: %c                          ",row[i]);
        key = getch();
        while(key!=' '&&key!='\t')
            key=getch();
        if(key==' ')
            position[0]=row[i];
        if(key=='\t' && i<9)
            i++;
        else
            i=0;
    }
    while(key !=' ');

    i=0;     //initialize i
    do       //choose column
    {
        printf("\r");
        printf("           Row: %c      Column: %c                          ",position[0],column[i]);
        key = getch();
        while(key!=' '&&key!='\t')
            key=getch();
        if(key==' ')
            position[1]=column[i];
        else if(key=='\t' && i<9)
            i++;
        else
            i=0;
    }
    while(key!=' ');
}
```

Fig. 8: The Code of Choose_Coordinate()

4.3 Deployment()

Each ship type has its own code for ship deployment, and they are executed sequentially in the following order: Carrier (OOOOO ship), Battleship (OOOO ship), Cruiser (OOO ship) and lastly Destroyer (OO ship).

Each ship deployment phase works as described below:

- Initial comment stating which type of ship
- Choose_Coordinate() is called to input the starting position of the ship.
- Choose_Direction() is called to input the direction of the ship.
- int grid [10][10]:

  In order to handle the ship deployment, a two dimensional integer array has been used. It serves as an internal and dynamic map in the program to record the position of ships. Initially, the array is filled with 0s. Once a ship is deployed, the value of corresponding elements will change from 0 to 1. The surrounding area is marked with 2. In the following steps, players can only put the ships on where 0s are, otherwise a message will pop up on the console to remind the player, and the program reverts back to the previous step.
- char display [10][10]:

  Another two dimensional char array called “display” is applied to update and visualize the map. An empty spot on the map is marked with “-”, and “O”s signify ships deployed. In hard mode, “ * ” stands for reefs, ships are not allowed to be there.

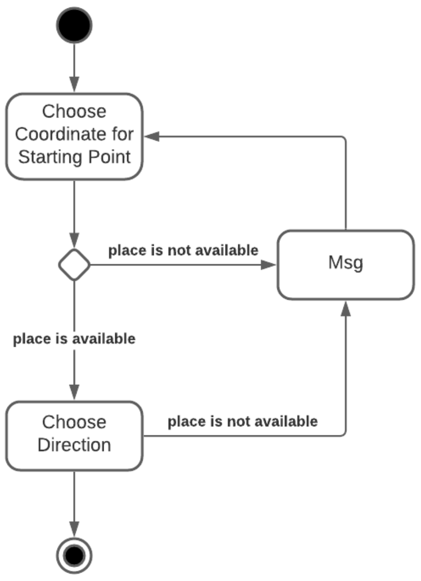


Fig. 9: Activity Diagram of deployment()

4.4 PvP_battle()

The activity diagram demonstrates how Player vs Player Mode works. To realize it, a while loop is used. The game will keep cycling between 2 states, player 1’s turn (case 0) and player 2’s turn (case 1). Because of the while loop, it keeps going until Win = 1 is triggered when player 1’s number of parts (P1) or player 2’s number of parts (P2) = 0.

When a part ‘O’ is destroyed, the corresponding value of the coordinate (grid[i][j]) turns from 1 to 2. When a shot is missed, the coordinate of that shot turns from 0 to 2. For visualizations, the destroyed parts are marked with “X”, and missed shots become blank space.

In PvP mode, in each turn, the program asks the player whether a ship has just been sunk or not.

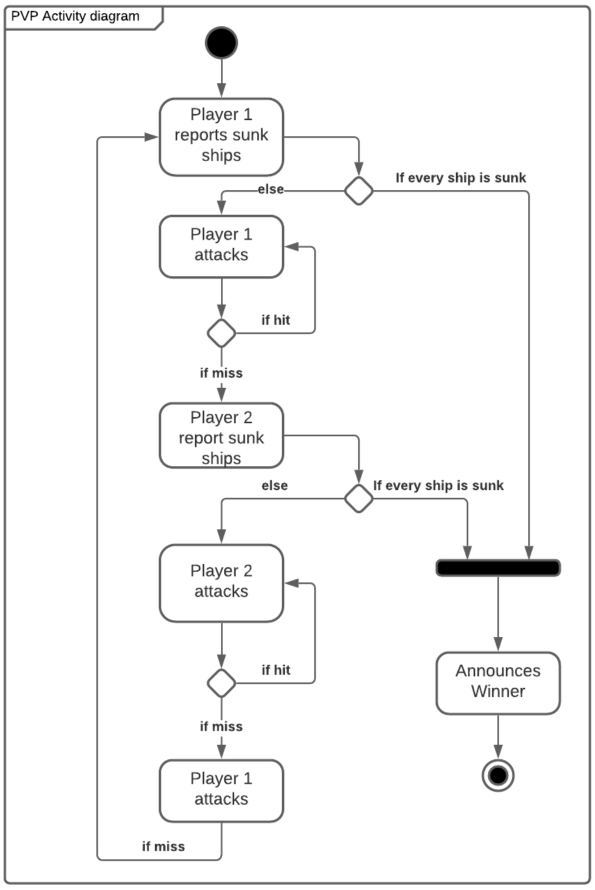


Fig. 10: Activity Diagram of PvP_Battle()

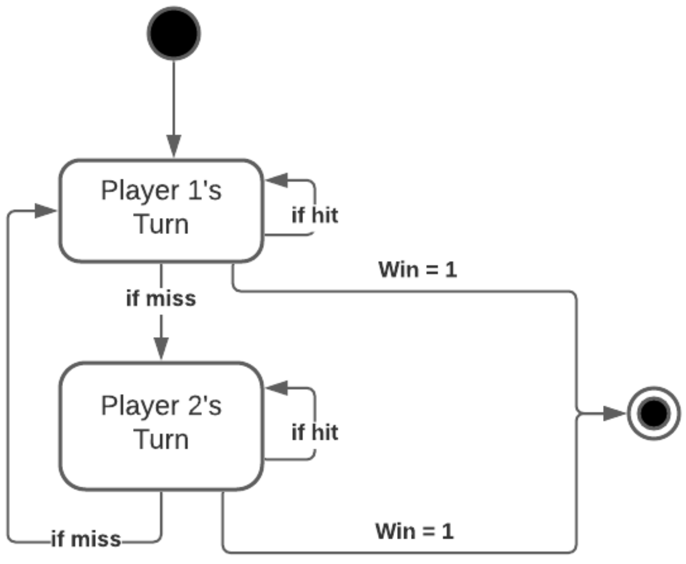
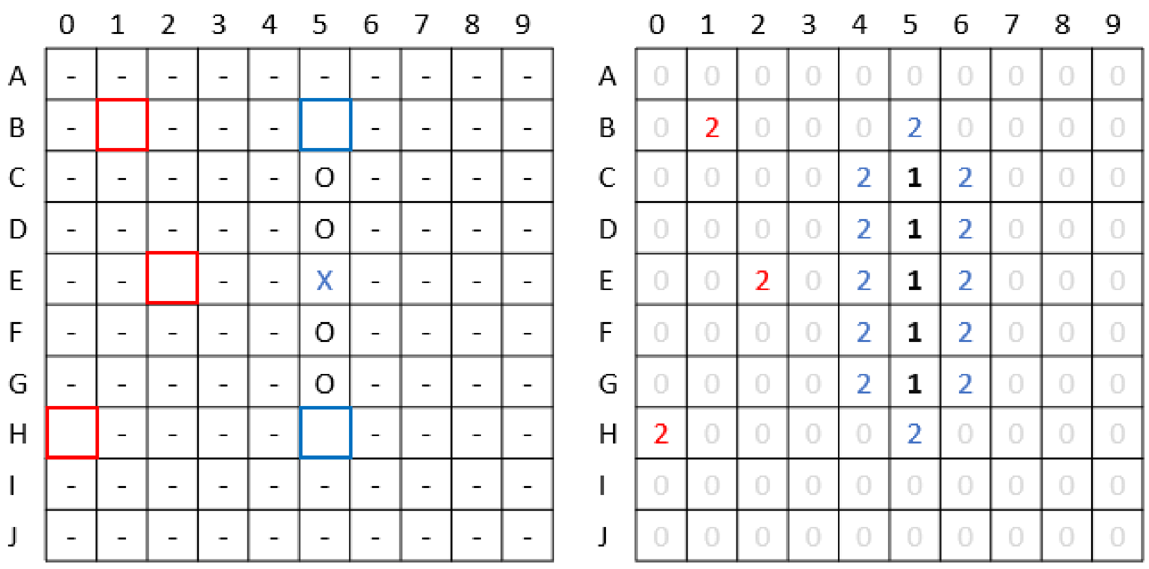


Fig. 11: State Machine Diagram of PvP_Battle()

4.5 AI_battle()

AI in Missouri aims at providing players with more challenges and excitements. The optimized algorithm enables the bot to launch attacks tactically and change the strategy based on situations.

This is achieved by utilizing an dynamic array called “smart_grid” which serves as an internal map for the program. The “smart_grid” is a 10 by 10 two dimensional array, and any changes in array “Player1.grid” will reflect on “smart_grid”. All the elements of the array are 0s by default, once a piece of ship on the human player’s grid is found, the corresponding element in the array will turn into 1. The algorithm is able to identify the boundary of ships, and those areas are marked with 2 instead. A miss will result in a change of value from 0 to 2 in the array.



	0	1	2	3	4	5	6	7	8	9
A	-	-	-	-	-	-	-	-	-	-
B	-		-	-	-		-	-	-	-
C	-	-	-	-	-	O	-	-	-	-
D	-	-	-	-	-	O	-	-	-	-
E	-	-		-	-	X	-	-	-	-
F	-	-	-	-	-	O	-	-	-	-
G	-	-	-	-	-	O	-	-	-	-
H		-	-	-	-		-	-	-	-
I	-	-	-	-	-	-	-	-	-	-
J	-	-	-	-	-	-	-	-	-	-

	0	1	2	3	4	5	6	7	8	9
A	0	0	0	0	0	0	0	0	0	0
B	0	2	0	0	0	2	0	0	0	0
C	0	0	0	0	2	**1**	2	0	0	0
D	0	0	0	0	2	**1**	2	0	0	0
E	0	0	2	0	2	**1**	2	0	0	0
F	0	0	0	0	2	**1**	2	0	0	0
G	0	0	0	0	2	**1**	2	0	0	0
H	2	0	0	0	0	2	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0

Fig. 12: Explanation of the AI attack algorithm

AI acts in certain patterns depending on the dynamic circumstance on the battlefield. At the beginning, the AI chooses random spots, if it is a miss shot, the AI remembers this coordinate and chooses another random spot without repeating the same mistakes in following turns until a bingo.

After a few rounds of trial and error, a piece of a ship is found. The AI starts looking around vertically and then horizontally, if necessary, to eliminate the rest of it. After destroying the ship, the AI returns to random mode. Note that AI exploits “smart_grid” to avoid wasteful shots. Moreover, it puts vertical search at higher priority, horizontal search is carried out only if there is nothing found in North and South. All these tactics make the bot a smart player.

However, as more ships are sunk after time, this method becomes less effective, and the program needs more time to find an empty spot or available targets. Therefore, it is time to change the strategy. Initially a fleet is composed of 30 “O"s, and the destroyed ones turn into “X”. When less than 10 “O”s remain on the human player’s board, the bot starts to scan from the upper left corner to the lower right corner of the whole map to find out the hidden ships.

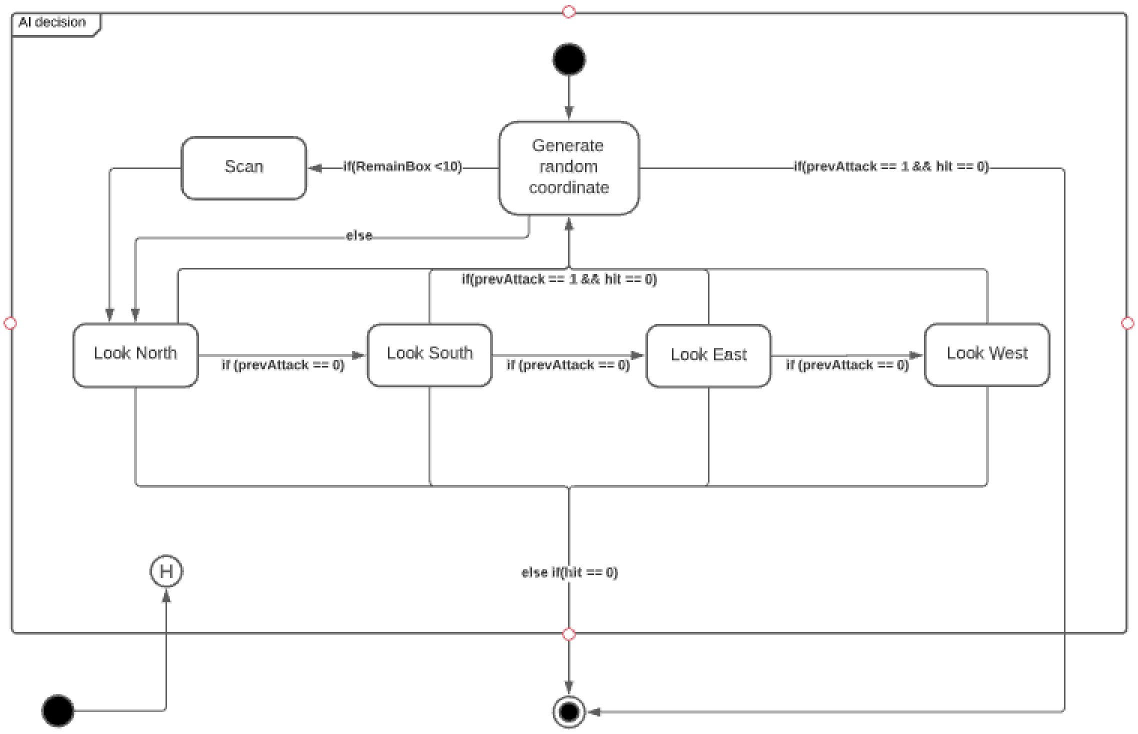
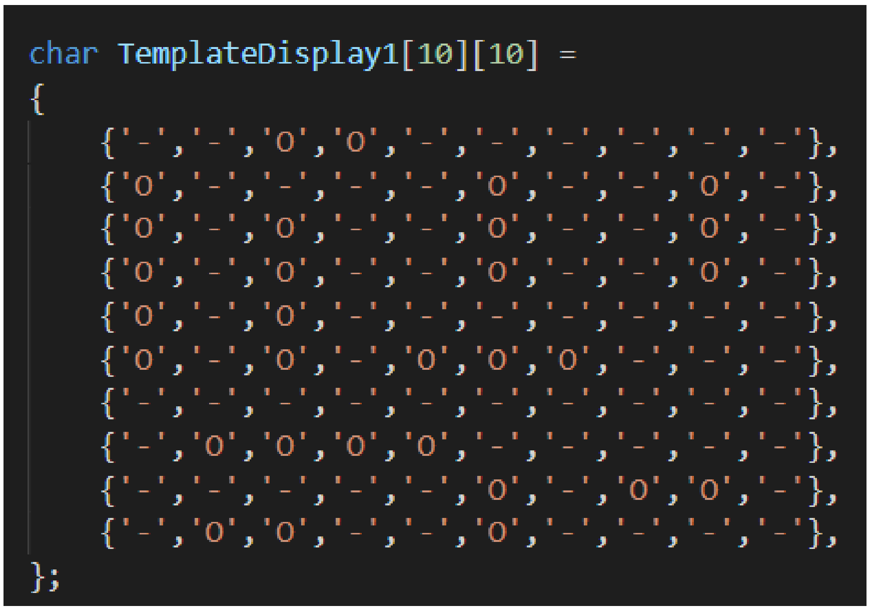


Fig. 13: State Machine Diagram of AI attack algorithm

4.6 Generate_map

5 ship deployment templates are prepared for the Computer, and 5 hard mode maps with obstacles are prepared for the player.

To store each template, 2 arrays are used. TemplateGridX[10][10] and TemplateDisplayX[10][10] are used for the ship deployment templates, and ReefTemplateGridX[10][10] and ReefTemplateDisplayX[10][10] are used for the obstacle maps, with X are numbers from 1 to 5.



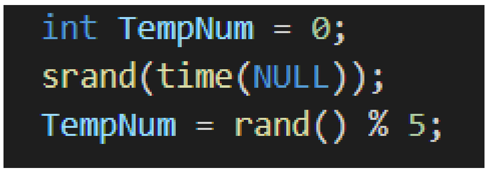
```
char TemplateDisplay1[10][10] =
{
    {'-','-','O','O','-','-','-','-','-','-'},
    {'O','-','-','-','-','O','-','-','O','-'},
    {'O','-','O','-','-','O','-','-','O','-'},
    {'O','-','O','-','-','O','-','-','O','-'},
    {'O','-','O','-','-','-','-','-','-','-'},
    {'O','-','O','-','O','O','O','-','-','-'},
    {'-','-','-','-','-','-','-','-','-','-'},
    {'-','O','O','O','O','-','-','-','-','-'},
    {'-','-','-','-','-','O','-','O','O','-'},
    {'-','O','O','-','-','O','-','-','-','-'},
};
```

Fig. 14: Example of TemplateDisplay1.

Like described in 1.2.2,

- The “grid” array is used to store the value of each coordinate.
- The “display” array is used to sketch the map to help players visualize what is going on.

After that, a random number generator is utilized to randomly choose a number from 0 to 4, this number is the case in which the corresponding template is used. For example, if the random number generator is 0, grid 1 is used.



```
int TempNum = 0;
srand(time(NULL));
TempNum = rand() % 5;
```

Fig. 15: Using rand() to randomize a case.

5. Our takeaways from the project

The design phase and the code structure were done together as a group. In the programming phase, although each member was responsible for a function individually, all major functions were often done with assistance from other member(s).

5.1 Chen Shih

For me, the key takeaways from this project are teamworks, communication skills, time management skills, and coding techniques. I consider myself as the helmsman of our team, who schedules meetings, assigns tasks, and keeps track of the progress. Missouri is my first formal project, although I already have two years of experience in programming. The idea of designing and developing a program from scratch with teammates is just amazing. Over the course we encountered different technical problems, and it gets frustrating at times, especially when debugging. Nevertheless, I genuinely enjoy the process of solving problems and the sense of achievement, which I believe is the most important part of learning.

5.2 Giwa Haheeb Rilwan

My contribution helped make the team project a success. I encouraged team members to brainstorm and I provided ideas and inputs for the assignments. Doing the project, I worked on developing some coding functions such as Check_Key(), Choose_Coodinate() & Choose_Direction(), and also took part in the Deployment () function. This project is really helpful in improving in my coding skill as well as modeling. Every step during a project is challenging and messy but with the help of my team members will be able to achieve our project goals.

5.3 Phan Thanh-Long

In this project I am most proud of coming up with the “probing shot” part of the attack algorithm for the Computer (after first hit, keep going North until miss, then go South until miss, if still no hit is scored (the ship isn’t placed vertically), go East until miss, then go West until miss). While I am responsible for generate_map(), I also took part in other functions, especially deployment() and AI_Battle().

My main problem is usually implementing my abstract idea into code. Sometimes I have a good idea of the logic that we are going to use and make good diagrams but I find it difficult to translate them into code. It got better at the end of the project, as I have more practice.

6. Summary and Future outlook for Missouri

The development of Missouri started with the Software Requirement Specification from the outset. All the essential requirements and design elements were listed out. This is followed by designing the overall structure of codes. All the major functions are developed and tested independently before they are merged into the main function. Subsequently, several minor changes were made to fulfill new requirements and to implement additional features, in the end, the final iteration is packaged and submitted for verification.

With a modular code structure, Missouri game has lots of room for growth in the foreseeable future . Given time, some improvements can be made to the game, such as:

- Improve AI to make it more efficient.
- Add flavors to the game by giving each ship a name and a history based on real world naval ships.
- A function to randomly deploy the ships for the computer.
- Improve sunk_ship() function and make reports about sunk ships automatically.

8. Source

[1]"getch() function in C with Examples - GeeksforGeeks", *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/getch-function-in-c-with-examples/. [Accessed: 04- Jul- 2021].

[2]"rand() and srand() in C/C++ - GeeksforGeeks", *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/. [Accessed: 04- Jul- 2021].

[3]İ. Baydan, "What Is sleep() function and How To Use It In C Program? – POFTUT", *Poftut.com*. [Online]. Available: https://www.poftut.com/what-is-sleep-function-and-how-to-use-it-in-c-program/. [Accessed: 04- Jul- 2021].

[4]"PlaySound function (Windows)", *docs.microsoft.com*. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/dd743680(v=vs.85). [Accessed: 04- Jul- 2021].

Source for the audio tracks: www.zapsplat.com