**Battleship**

Project1

CSC- 17A – 43950

Sergio Hurtado

04 May, 2015

Board of Contents

1.Introduction………………………………………….3  
Rule and Gameplay  
Programming Issues

2. Development………………………..………………3-4  
Strategy

3. Variables list…………………………….……..……5-6

4. Topic Covered (Checklist)…………………….……...7

5. Libraries included……………………………………..7

6. Pseudo Code……………………………………….8-9

7. Flowchart…………………….…………………….10-14

8. Main Code………………………………………....15-17

1. Introduction

**Rules and Gameplay**

The players can choose the size of the board which 8x8 to 10x10. After the player choose the size, two boards created . One is for player, the other is for AI. Both of them have 5 ships which are 1 5-unit ship, 1 4-unit ship,1 3-unit ship, and 2 2-unit ships. After the player enters the coordinates to place the ship, the board will refresh and use numbers to label the ships. The coordinates of AI’s ships will be place randomly. Player needs to hit all the ships to win the game. In the game, “X” means hit and “O” means miss. The program will check for valid input. The program will also scan the board to check whether there are any ships on the board. If there aren’t any, the game ends.

**Program issues**

The toughest part of the program was writing the code for the AI. The AI’s fire part is complicated because it is very difficult to make the AI do things that a human player would do. For example, the AI should fire around a point where there was a hit, and continue fire if there are 2 hits in a row. If one side is “O” or the side of the board, the AI needs to check the other side too. Also, the AI board shown to the player isn’t the AI’s actual board. It’s just a clear board and after the player fires, the program will compare the coordinate to the real board. Then it records and shows “O”, “X”, or invalid input. The validation part was also very complex. I had to get input as string and allow the user to enter A1A5 format to choose attack. I was able to check length, but had to use ascii code to translate after checking length.

2. Strategy:

Battleship needs three boards. Using two dimensional arrays really simplified the board manipulation. Also, A-J were used to label the rows and 0-9 to label the columns. It makes the players enter the coordinates clearly and prevent them get confused. It is possible the boards which are larger than 10x10, but there are only 0-9 for digits, and anything more would be confusing. I tried to let the player to choose which ship they want to place first, but there are 2 2-unit ships, so I need to use a Boolean to remember the first 2-unit ship. I figured, it would be easier to just have the player place the ships in order. After the player’s place ship part, I need to randomize the AI ships’ coordinates. Avoid overflowing the array by using things such as srand()%num-5, where num is the size of the board. After generating the coordinate, the program will random to place it horizontally or vertically. If the ship overlaps, it will try to place it in other way. If it is still invalid, the program will randomize the coordinates again.

Once the game board is set up, I use a switch to separate the player’s fire turn and AI’s fire turn. If the game isn’t over, the program will go to AI’s turn. If the game ends, the program will jump to other case. Also, I put a do-while loop outside the switch and repeat until the game is over.

For the AI’s fire part, I let the AI fire randomly until it hits. After AI hits, the program will record the coordinate and check the four coordinates beside it until it gets second hit. After a second hit, AI will fire that direction until it misses, touches the side, or overlaps. Then it will fire the opposite side until misses, oversizes, or overlaps again. After it finishes these steps, it will go back to random fire mode.

3. Variables list

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Variable Name** | **Descripbion** |  | **Line** |
| int | num | size of 2d dynamic array | main | 22 |
|  | turn=1 | menu turns | main | 23 |
|  | over | Check if over | main | 24 |
|  | count | Loop control | usrPlce | 8 |
|  | max | Check larger | usrPlce | 9 |
|  | min | Smaller | usrPlce | 10 |
|  | turn | in function aifire return var | aiFire | 11 |
|  | hplan | hit plan after first hit (cross) | aiFire | 13 |
|  | unit[5] | units of the ships in an array | Player | 17 |
|  | x1, x2, y1, y2 | 2 coordinates to place ship | Player | 18 |
|  | hx, hy, x, y | current, pre fire coordinates | AI | 22 |
|  | oppcombo | opposite side combo | AI | 24 |
|  | combo |  | AI | 25 |
| float | pbtl=0 | both total number of fire | getStates | 8 |
|  | pac | accuracy | getStates | 9 |
|  | hit | player hit counter | Player | 23 |
|  | miss | player miss counter | Player | 24 |
|  | hit | ai hit miss counter | AI | 34 |
|  | miss | ai miss counter | AI | 35 |
| char | temp |  | init | 8 |
|  | row |  | board | 34 |
|  | \*\*pb | player board | Player | 13 |
|  | \*\*fake | ai fake tales | AI | 12 |
|  | \*\*real | ai real tales | AI | 13 |
|  | cx, cy | ai coordinates in char | AI | 27 |
| string | temp |  | intro | 8 |
|  | place | player place input type | usrPlace | 11 |
|  | fire | player fire input type | pFire | 8 |
| bool | invalid=false | num validation | getNum | 6 |
|  | valid | string validation | usrPlce | 12 |
|  | digit | isdigit validation | usrPlce | 13 |
|  | valid | ai place validation | aiPlace | 10 |
|  | over=true | game over Boolean | pFire | 9 |
|  | valid | player fire validation | pFire | 10 |
|  | valid | ai fire validation | main | 430 |
|  | over | ai over boolean | AI | 18 |
|  | done | finish fire | AI | 19 |
|  | cross[4] | cross 4 boxes around hit | AI | 20 |
|  | crossdone | if true back to random | AI | 21 |
|  | goback | invalid back to random | AI | 22 |
|  | finish | combo finish Boolean | AI | 24 |
|  | hit | hit Boolean | AI | 28 |
|  | oneend | one side finish | AI | 29 |
|  | combohit | keep fire the same direction | AI | 30 |
| fstream | io | i/o file in function intro | intro | 5 |
|  | io | i/o file in function init | init | 7 |
|  | io | i/o file in function getstates | getStates | 7 |
| time\_t | start, end | Delay ai fire | pause | 8 |
| PlayerB | pb | Player board | main | 25 |
| PlayerG | pG | Player game data | main | 26 |
| PlayerS | \*ps | Player States | main | 27 |
| AIB | aib | AI board | main | 28 |
| AIG | ai | AI game data | main | 29 |
| AIS | \*as | AI game states | main | 30 |

4. Topics Covered (Checklist)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Chapter | type | code | cpp | Line |
| Memory Allocation | char\*\* | pb.pb = new char\*[num]; | init | 45 |
|  |  | for(int i=0;i<num;i++){ | init | 48 |
|  |  | pb.pb[i] = new char[num]; | init | 50 |
|  |  | } | init | 51 |
| delete 2d dynamic arr |  | delete[] pb.pb[i]; | main | 71 |
|  |  | delete[] pb.pb; | main | 75 |
| function with struct |  | void init(PlayerT &, AIT &, int, PlayerG &, PlayerS &, AIG &, AIS &); | Prototypes | 15 |
|  |  | void board(char \*\*, char \*\*, char \*\*, int); | Prototypes | 16 |
| pointer notation |  | if(\*(\*(pb+pG.y1)+k)==' '){ | usrPlace | 78 |
| structure pointer | PlayerS | PlayerS \*ps | main | 27 |
| delete struct pointer |  | delete ps, as; | main | 78 |
| cctype | isdigit | if(isdigit(place[1]) && isdigit(place[3])){ | usrPlace | 35 |
| Reading from binary file | io | io.open("unit.txt", ios::in | ios::binary); | init | 10 |
| Writing to binary file | io | io.open("rank.txt", ios::out | ios::binary); | getStates | 20 |
| struct |  | struct PlayerB | Player | 11 |
|  | array | int unit[5]; | Player | 17 |

5. Libraries included

* <cstdlib>
* <iostream>
* <ctime>
* <fstream>
* <iomanip>
* <cctype>
* Player.h
* AI.h
* LibraryInc.h
* Prototypes.h

6. Pseudo Code

Get the size of the board

Initialize

Reset board

Output board

do

{

Input 2 coordinates to place ship

}while (invalid)

place other ship and check validation

do

{

AI random ship coordinates

}while (invalid)

**case1(player fire)**

Player enter coordinate to fire

check validation

check hit/miss and add count

display board again

check game over (no number s on the board)

if(true) case3

else case2

**case2 (AI fire)**

do{

if (no hit/ no combo) random hit

if (hit) check cross 4

if(all invalid) go back to random

if(hit) combo++, add count

else add count

if (cross 4 coordinates hit) continue fire that direction

if(invalid) jump to next statement, oppcombo++, combo=0

if(miss) oppcombo++, combo=0, add count

if(hit) combo++, add count

if(oppcombo>0) check the opposite side

if(invalid) go back to random

if(miss) oppcombo=0, add count

if(hit) oppcombo+1

}while (not fire)

check game over

if (true) go to case 4

else go to case1

**case3**

Player win, turn=5

**case 4**

Player lose, turn=5

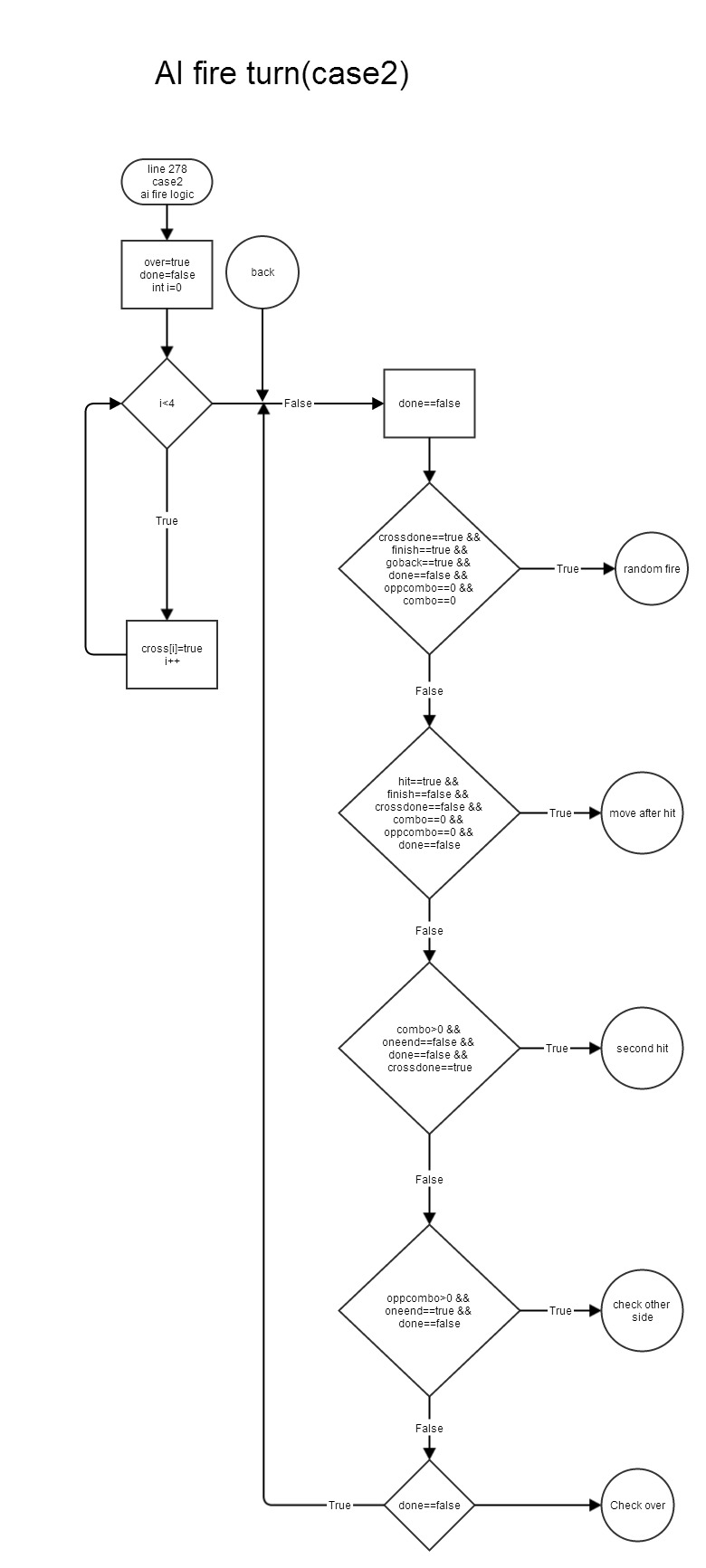
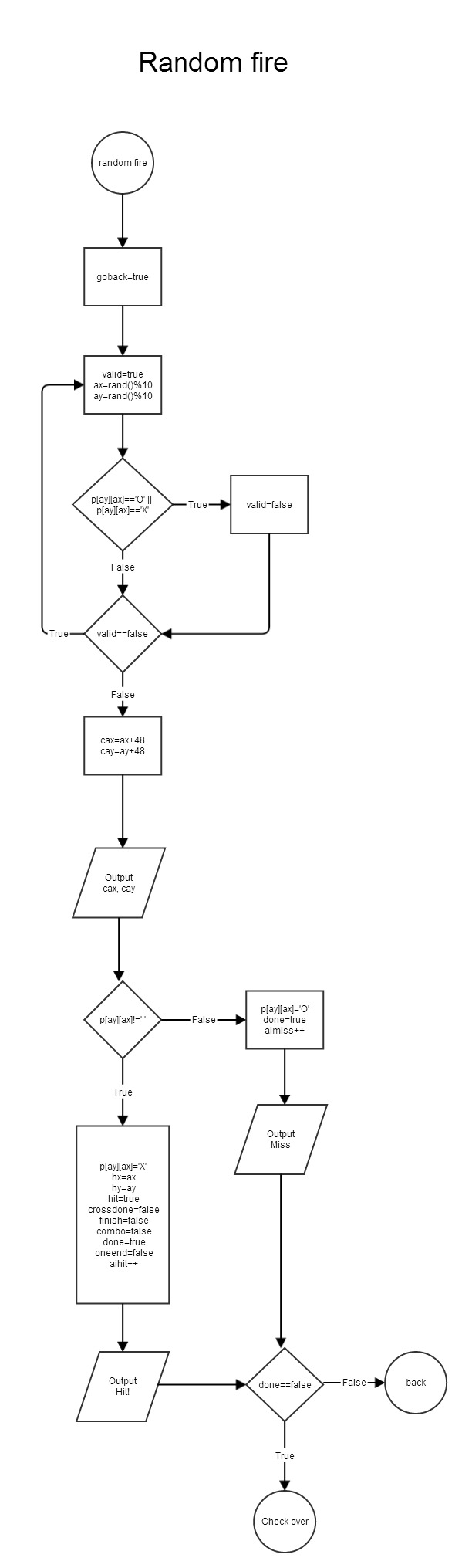
if(turn<5) keep looping the case

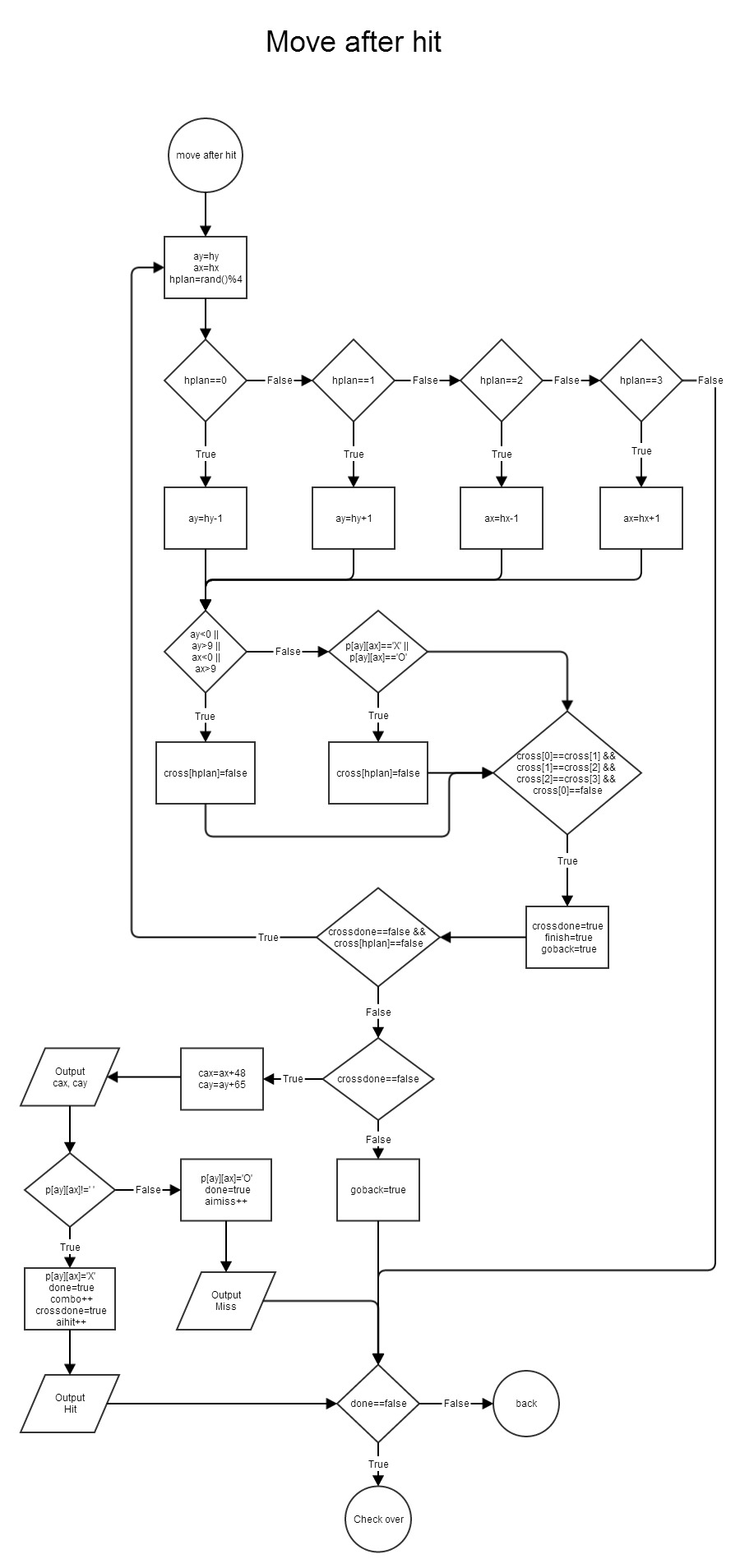
scan player and ai board

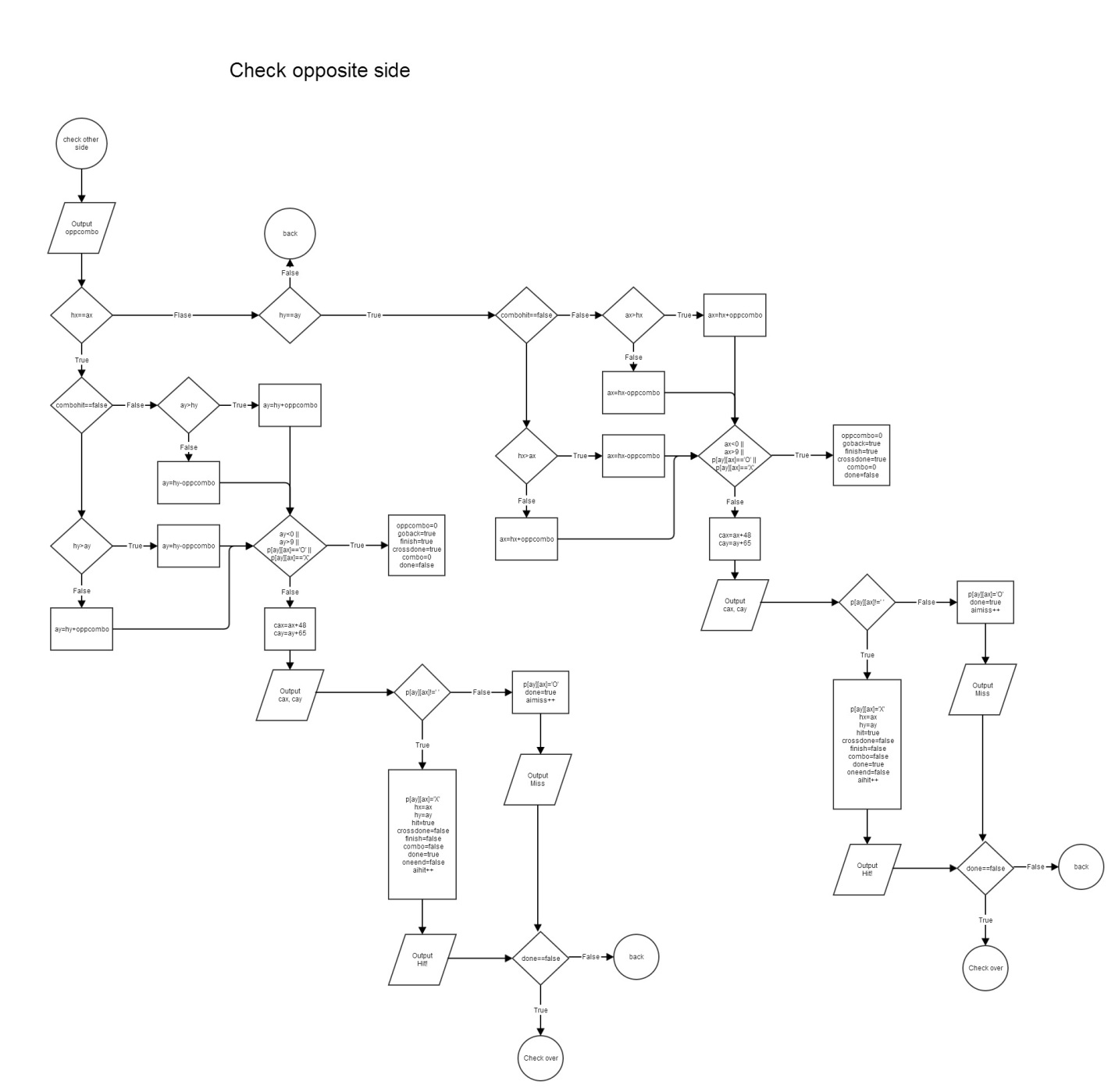
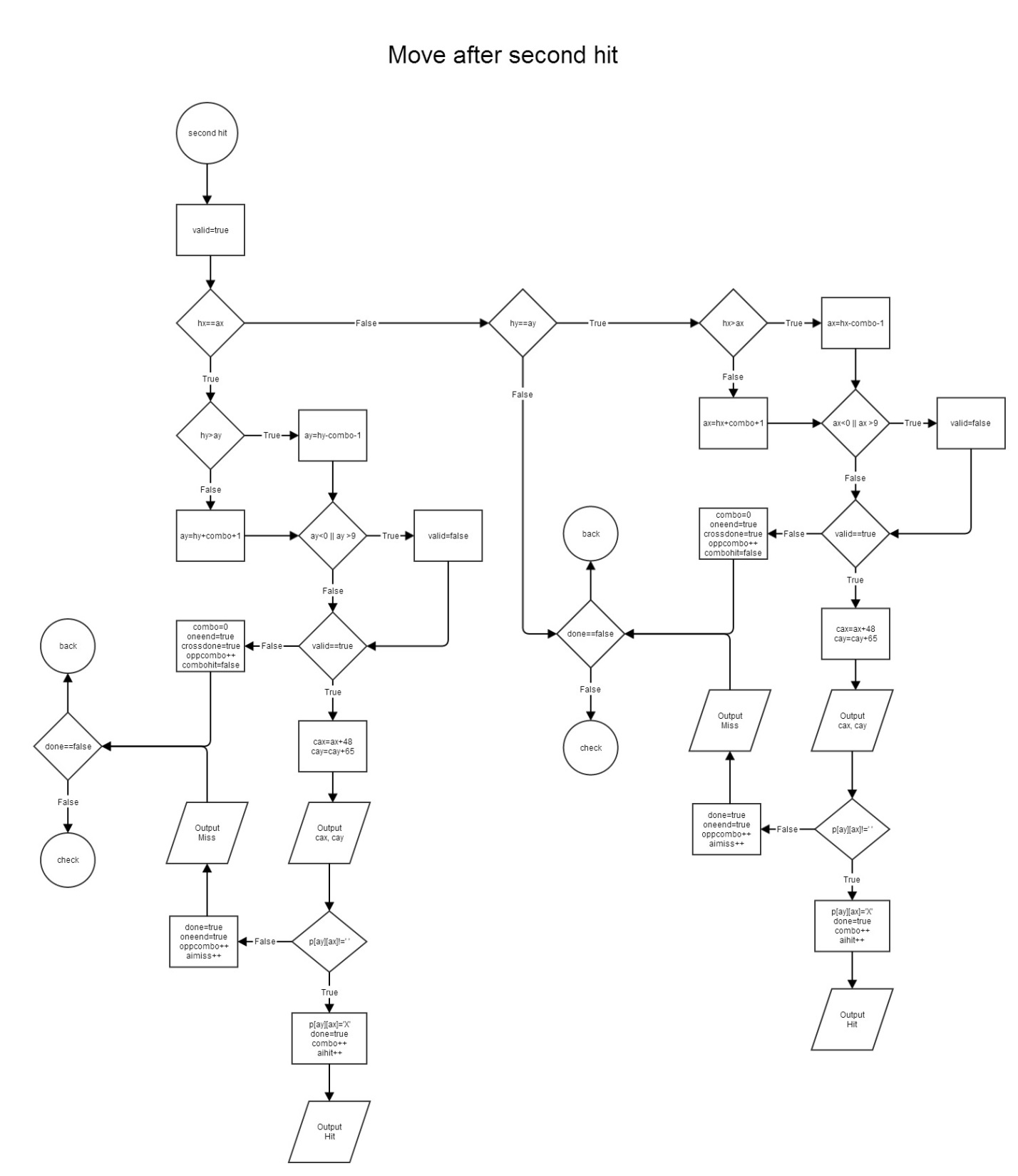
calculate accuracy hit/(hit+miss)

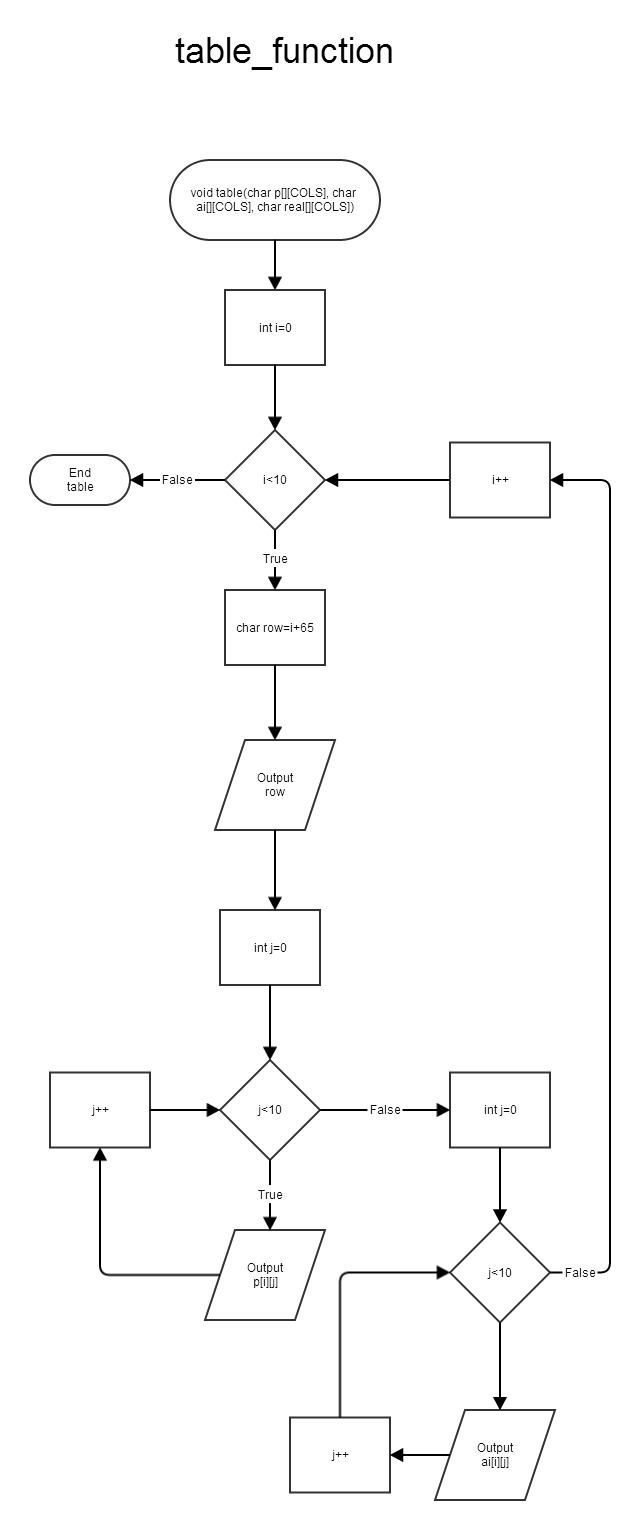
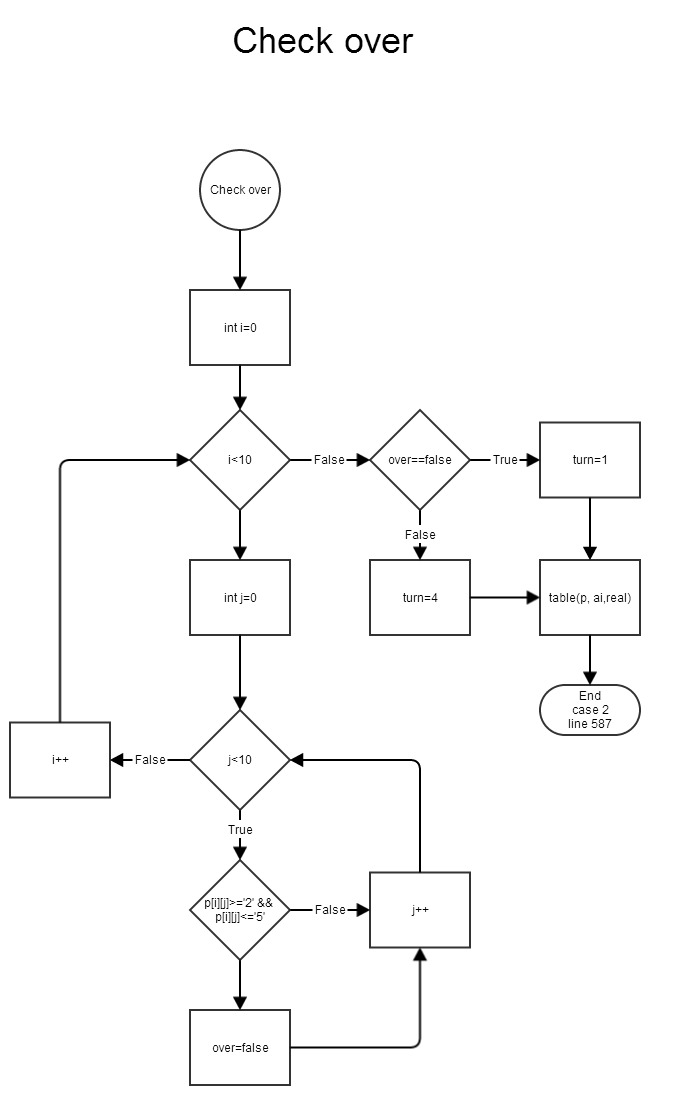
display player rank

output to rank.txt

7. Flowchart







8. Code – main

/\*

\* File: main.cpp

\* Author: Sergio Hurtado

\* Created on April 22, 2015, 8:03 AM

\*/

//User Libraries

#include "LibraryInc.h"

#include "Player.h"

#include "AI.h”

//Global Constants

//Prototypes

#include "Prototypes.h"

//Execution Begins Here

int main(int argc, char\*\* argv)

{

intro();

int num = getNum();

int turn = 1;

int over;

PlayerT pb;

PlayerG pG;

PlayerS \*ps = new PlayerS;

AIT ait;

AIG ai;

AIS \*as = new AIS;

cout << fixed << showpoint << setprecision(2);

//reset

srand(static\_cast<unsigned int>(time(0)));

init(pb, ait, num, pG, ps, ai, as);

board(pb.pb, ait.fake, ait.real, num);

//player prepare

usrPlce(pb.pb, ait.fake, ait.real, pG, num);

aiplace(pb.pb, ait.fake, ait.real, pG.unit, pG.x1, pG.y1, num);

//game start

do

{

switch(turn)

{

case 1:

{

turn = pfire(pb.pb, ait.fake, ait.real, pG, ps, num);

break;

}

case 2:

{

turn = aifire(pb.pb, ait.fake, ait.real, ai, as, num);

break;

}

case 3:

{

cout << "You win!\n\n";turn = 5;

break;

}

case 4:

{

cout << "You lose!\n\n";turn = 5;

break;

}

}

}while(turn<5);

getstates(ps);

//free memory

for (int i = 0;i<num;i++)

{

delete[] pb.pb[i];

delete[] ait.real[i];

delete[] ait.fake[i];

}

delete[] pb.pb;

delete[] ait.real;

delete[] ait.fake;

delete ps, as;

return 0;

}