#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <time.h>

using namespace std;

#define BOARD\_SIZE\_b 6

#define BOARD\_SIZE\_a 10

#define BOARD\_SIZE\_p 15

#define TRUE 0

#define FALSE 1

// I refuse to have to read in the game board for every function that calls it..

// Global "lost" variable is kind of stupid, but, it works...

char board\_b[BOARD\_SIZE\_b][BOARD\_SIZE\_b];

char game\_board\_b[BOARD\_SIZE\_b][BOARD\_SIZE\_b];

char board\_a[BOARD\_SIZE\_a][BOARD\_SIZE\_a];

char game\_board\_a[BOARD\_SIZE\_a][BOARD\_SIZE\_a];

char board\_p[BOARD\_SIZE\_p][BOARD\_SIZE\_p];

char game\_board\_p[BOARD\_SIZE\_p][BOARD\_SIZE\_p];

int lost = 0;

/\* Function prototypes \*/

void display\_welcome();

void build\_board\_b();

void build\_board\_a();

void build\_board\_p();

void build\_gboard\_b();

void build\_gboard\_a();

void build\_gboard\_p();

void create\_mines\_b();

void create\_mines\_a();

void create\_mines\_p();

void print\_board\_b();

void print\_board\_a();

void print\_board\_p();

void print\_fullboard\_b();

void print\_fullboard\_a();

void print\_fullboard\_p();

void start\_b();

void start\_a();

void start\_p();

int play\_game\_b();

int play\_game\_a();

int play\_game\_p();

void play\_again\_b();

void play\_again\_a();

void play\_again\_p();

int check\_win\_game\_b();

int check\_win\_game\_a();

int check\_win\_game\_p();

void check\_for\_mine\_b(int, int);

void check\_for\_mine\_a(int, int);

void check\_for\_mine\_p(int, int);

int check\_for\_nearby\_mines\_b(int, int);

int check\_for\_nearby\_mines\_a(int, int);

int check\_for\_nearby\_mines\_p(int, int);

// Main function

int main ()

{

display\_welcome();

printf("\n If you are ready to play Minesweeper...... Just press ENTER. :)");

getchar();

system("cls");//clear screen

printf("\n\n Please Choose Your Level you want to play from the list below. \n ( Please Enter The No. Preeceding Your Choice ) \n\n");

printf(" 1. Beginner Level \n\n 2. Amateur Level \n\n 3. Professional Level \n\n 4. Exit Minesweeper \n\n");

int choice;

scanf ("%d",&choice);

switch (choice)

{

case 1:

system("cls");

cout<<"Beginner Level Starting...";

start\_b();

break;

case 2:

system("cls");

cout<<"Amateur Level Starting...";

start\_p();

break;

case 3:

system("cls");

cout<<"Professional Level Starting...";

start\_p();

break;

default:

system("cls");

cout<<"Exiting Minesweeper... \n";

return 0;

}

return 0;

}

/\* Build board used for created random mines \*/

void build\_board\_b()

{

int i, j;// Assign char for board elements

for(i = 0; i < BOARD\_SIZE\_b; i++)

for(j = 0; j < BOARD\_SIZE\_b; j++)

board\_b[i][j] = 'o';

// Place mines in this board, it remains hidden from user until the game has finished.

create\_mines\_b();

}

void build\_board\_a()

{

int i, j;// Assign char for board elements

for(i = 0; i < BOARD\_SIZE\_a; i++)

for(j = 0; j < BOARD\_SIZE\_a; j++)

board\_a[i][j] = 'o';

// Place mines in this board, it remains hidden from user until the game has finished.

create\_mines\_a();

}

void build\_board\_p()

{

int i, j;// Assign char for board elements

for(i = 0; i < BOARD\_SIZE\_p; i++)

for(j = 0; j < BOARD\_SIZE\_p; j++)

board\_p[i][j] = 'o';

// Place mines in this board, it remains hidden from user until the game has finished.

create\_mines\_p();

}

/\* Build game board for user input \*/

void build\_gboard\_b()

{

int i, j;

int row, col;

cout<<" Creating Beginner Game Board... \n Ready... \n Set... \n PLAY MINESWEEPER !!!! \n\n";

// Assign char 'o' for all board elements

for(i = 0; i < BOARD\_SIZE\_b; i++)

for(j = 0; j < BOARD\_SIZE\_b; j++)

game\_board\_b[i][j] = 'o';

// Print board

for(col = 0; col < BOARD\_SIZE\_b; col++)

cout<<(col + 1)<<" ";

cout<<"\n\n";

for(row = 0; row < BOARD\_SIZE\_b; row++)

{

for(col = 0; col < BOARD\_SIZE\_b; col++)

{

cout<<game\_board\_b[row][col]<<" ";

}

cout<<" "<<(row + 1);

cout<<"\n";

}

}

void build\_gboard\_a()

{

int i, j;

int row, col;

printf(" Creating Beginner Game Board... \n Ready... \n Set... \n PLAY MINESWEEPER !!!! \n\n");

// Assign char 'o' for all board elements

for(i = 0; i < BOARD\_SIZE\_a; i++)

for(j = 0; j < BOARD\_SIZE\_a; j++)

game\_board\_a[i][j] = 'o';

// Print board

for(col = 0; col < BOARD\_SIZE\_a; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_a; row++)

{

for(col = 0; col < BOARD\_SIZE\_a; col++)

{

printf("%c ", game\_board\_a[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

void build\_gboard\_p()

{

int i, j;

int row, col;

printf(" Creating Beginner Game Board... \n Ready... \n Set... \n PLAY MINESWEEPER !!!! \n\n");

// Assign char 'o' for all board elements

for(i = 0; i < BOARD\_SIZE\_p; i++)

for(j = 0; j < BOARD\_SIZE\_p; j++)

game\_board\_p[i][j] = 'o';

// Print board

for(col = 0; col < BOARD\_SIZE\_p; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_p; row++)

{

for(col = 0; col < BOARD\_SIZE\_p; col++)

{

printf("%c ", game\_board\_p[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

/\* Create random places in the array for mines \*/

void create\_mines\_b()

{

int i, random;

// Seeding srand() with time(0) so that mine locations aren't the same every time the game is played.

srand(time(0));

for (i = 0; i < BOARD\_SIZE\_b; i++)

{

random = rand() % (BOARD\_SIZE\_b);

board\_b[random][i] = '\*';

}

}

void create\_mines\_a()

{

int i, random;

// Seeding srand() with time(0) so that mine locations aren't the same every time the game is played.

srand(time(0));

for (i = 0; i < BOARD\_SIZE\_a; i++)

{

random = rand() % (BOARD\_SIZE\_a);

board\_a[random][i] = '\*';

}

}

void create\_mines\_p()

{

int i, random;

// Seeding srand() with time(0) so that mine locations aren't the same every time the game is played.

srand(time(0));

for (i = 0; i < BOARD\_SIZE\_p; i++)

{

random = rand() % (BOARD\_SIZE\_p);

board\_p[random][i] = '\*';

}

}

/\* Print the game board \*/

void print\_board\_b()

{

int row, col;

system("cls");

for(col = 0; col < BOARD\_SIZE\_b; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_b; row++)

{

for(col = 0; col < BOARD\_SIZE\_b; col++)

{

printf("%c ", game\_board\_b[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

void print\_board\_a()

{

int row, col;

system("cls");

for(col = 0; col < BOARD\_SIZE\_a; row++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_a; row++)

{

for(col = 0; col < BOARD\_SIZE\_a; col++)

{

printf("%c ", game\_board\_a[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

void print\_board\_p()

{

int row, col;

system("cls");

for(col = 0; col < BOARD\_SIZE\_p; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_p; row++)

{

for(col = 0; col < BOARD\_SIZE\_p; col++)

{

printf("%c ", game\_board\_p[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

/\* Print the full board showing mines \*/

void print\_fullboard\_b()

{

int row, col;

system("cls");

for(col = 0; col < BOARD\_SIZE\_b; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_b; row++)

{

for(col = 0; col < BOARD\_SIZE\_b; col++)

{

printf("%c ", board\_b[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

void print\_fullboard\_a()

{

int row, col;

system("cls");

for(col = 0; col < BOARD\_SIZE\_a; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_a; row++)

{

for(col = 0; col < BOARD\_SIZE\_a; col++)

{

printf("%c ", board\_a[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

void print\_fullboard\_p()

{

int row, col;

system("cls");

for(col = 0; col < BOARD\_SIZE\_p; col++)

printf("%d ", col + 1);

printf("\n\n");

for(row = 0; row < BOARD\_SIZE\_p; row++)

{

for(col = 0; col < BOARD\_SIZE\_p; col++)

{

printf("%c ", board\_p[row][col]);

}

printf(" %d ", row + 1);

printf("\n");

}

}

/\* Take user input for playing of the game \*/

int play\_game\_b()

{

int r\_selection = 0, c\_selection = 0,

nearbymines = 0, nearbymines2 = 0,

nearbymines3 = 0, nearbymines4 = 0,

nearbymines5 = 0, nearbymines6 = 0,

nearbymines7 = 0, nearbymines8 = 0,

i = 0;

// Recieves data from the user, first the row number, then the column number...

// I could think of other ways to do it, but this one seemed easiest.

do {

printf("\nMake a selection (ie. row [ENTER] col): \n");

printf("Row--> ");

scanf("%d", &r\_selection);

printf("Col--> ");

scanf("%d", &c\_selection);

} while(r\_selection < 1 || r\_selection > BOARD\_SIZE\_b || c\_selection < 1 || c\_selection > BOARD\_SIZE\_b);

// ^ Checks for any invalid input statements from user.

check\_for\_mine\_b(r\_selection - 1, c\_selection - 1);

if(lost == 1)

return -1;

// Checks for nearby mines at every direction from user input location.

//Assigns that location the number of mines found nearby, updating the board.

nearbymines = check\_for\_nearby\_mines\_b(r\_selection - 1, c\_selection - 1);

game\_board\_b[r\_selection - 1][c\_selection - 1] = (char)( ((int)'0') + nearbymines );

// The following checks for mines nearby elements

// in the array with no mines.. it's a continuous

// loop until either a mine is found, or we

// reach the end of the array & it cannot be checked any further.

// It also changes the game\_board[] with '0' if no mines are found. Very useful piece of code.

// It checks all elements left, right, up, down and all diagonal directions.

// By running a function that checks these same directions.

// A bit much to follow, though. I'm sure there's a much better way. I just don't know it yet.

if(nearbymines == 0)

{

if(c\_selection != BOARD\_SIZE\_b)

{

i = 0;

while(nearbymines == 0 && (c\_selection - 1 + i) < BOARD\_SIZE\_b)

{

// This is checking elements to the right

nearbymines = check\_for\_nearby\_mines\_b(r\_selection - 1, (c\_selection - 1 + i));

if(nearbymines != -1)

{

game\_board\_b[r\_selection - 1][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines );

i++;

}

}

if(r\_selection != 1)

{

i = 0;

while(nearbymines5 == 0 && (c\_selection - 1 + i) < BOARD\_SIZE\_b && (r\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-uright

nearbymines5 = check\_for\_nearby\_mines\_b((r\_selection - 1 - i), (c\_selection - 1 + i));

if(nearbymines5 != -1)

{

game\_board\_b[(r\_selection - 1) - i][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines5);

i++;

}

}

}

if(r\_selection != BOARD\_SIZE\_b)

{

i = 0;

while(nearbymines6 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_b && (c\_selection - 1 + i) < BOARD\_SIZE\_b )

{

// This is checking elements to the diagonal-dright

nearbymines6 = check\_for\_nearby\_mines\_b((r\_selection - 1 + i), (c\_selection - 1 + i));

if(nearbymines != -1)

{

game\_board\_b[(r\_selection - 1) + i][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines6);

i++;

}

}

}

}

if(r\_selection != BOARD\_SIZE\_b)

{

i = 0;

while(nearbymines2 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_b)

{

// This is checking elements heading down

nearbymines2 = check\_for\_nearby\_mines\_b((r\_selection - 1 + i), c\_selection - 1);

if(nearbymines2 != -1)

{

game\_board\_b[(r\_selection - 1) + i][c\_selection - 1] = (char) ( ((int)'0') + nearbymines2 );

i++;

}

}

if(c\_selection != BOARD\_SIZE\_b)

{

i = 0;

while(nearbymines7 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_b && (c\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-dleft

nearbymines7 = check\_for\_nearby\_mines\_b((r\_selection - 1 + i), (c\_selection - 1 - i));

if(nearbymines != -1)

{

game\_board\_b[(r\_selection - 1) + i][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines7);

i++;

}

}

}

}

if(r\_selection != 1)

{

i = 0;

while(nearbymines3 == 0 && (r\_selection - i) > 0)

{

// This is checking elements heading up

nearbymines3 = check\_for\_nearby\_mines\_b((r\_selection - 1 - i), c\_selection - 1);

if(nearbymines3 != -1)

{

game\_board\_b[(r\_selection - 1) - i][c\_selection - 1] = (char) ( ((int)'0') + nearbymines3 );

i++;

}

}

if(c\_selection != BOARD\_SIZE\_b)

{

while(nearbymines8 == 0 && (c\_selection - 1 - i) > 0 && (r\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-uleft

nearbymines8 = check\_for\_nearby\_mines\_b((r\_selection - 1 - i), (c\_selection - 1 - i));

if(nearbymines8 != -1)

{

game\_board\_b[(r\_selection - 1) - i][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines8);

i++;

}

}

}

}

if(c\_selection != 1)

{

i = 0;

while(nearbymines4 == 0 && (c\_selection - i) > 0)

{

// This is checking elements to the left

nearbymines4 = check\_for\_nearby\_mines\_b(r\_selection - 1, (c\_selection - 1 - i));

if(nearbymines4 != -1)

{

game\_board\_b[r\_selection - 1][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines4 );

i++;

}

}

}

}

// Handles a player winning.

if(check\_win\_game\_b() == TRUE)

{

system("cls");

print\_fullboard\_b();

printf("\n\nYou've won the game!! Congrats!!\n\n");

play\_again\_b();

}

return 0;

}

int play\_game\_a()

{

int r\_selection = 0, c\_selection = 0,

nearbymines = 0, nearbymines2 = 0,

nearbymines3 = 0, nearbymines4 = 0,

nearbymines5 = 0, nearbymines6 = 0,

nearbymines7 = 0, nearbymines8 = 0,

i = 0;

// Recieves data from the user, first the row number, then the column number...

// I could think of other ways to do it, but this one seemed easiest.

do {

printf("\nMake a selection (ie. row [ENTER] col): \n");

printf("Row--> ");

scanf("%d", &r\_selection);

printf("Col--> ");

scanf("%d", &c\_selection);

} while(r\_selection < 1 || r\_selection > BOARD\_SIZE\_a || c\_selection < 1 || c\_selection > BOARD\_SIZE\_a);

// ^ Checks for any invalid input statements from user.

check\_for\_mine\_a(r\_selection - 1, c\_selection - 1);

if(lost == 1)

return -1;

// Checks for nearby mines at every direction from user input location.

//Assigns that location the number of mines found nearby, updating the board.

nearbymines = check\_for\_nearby\_mines\_a(r\_selection - 1, c\_selection - 1);

game\_board\_a[r\_selection - 1][c\_selection - 1] = (char)( ((int)'0') + nearbymines );

// The following checks for mines nearby elements

// in the array with no mines.. it's a continuous

// loop until either a mine is found, or we

// reach the end of the array & it cannot be checked any further.

// It also changes the game\_board[] with '0' if no mines are found. Very useful piece of code.

// It checks all elements left, right, up, down and all diagonal directions.

// By running a function that checks these same directions.

// A bit much to follow, though. I'm sure there's a much better way. I just don't know it yet.

if(nearbymines == 0)

{

if(c\_selection != BOARD\_SIZE\_a)

{

i = 0;

while(nearbymines == 0 && (c\_selection - 1 + i) < BOARD\_SIZE\_a)

{

// This is checking elements to the right

nearbymines = check\_for\_nearby\_mines\_a(r\_selection - 1, (c\_selection - 1 + i));

if(nearbymines != -1)

{

game\_board\_a[r\_selection - 1][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines );

i++;

}

}

if(r\_selection != 1)

{

i = 0;

while(nearbymines5 == 0 && (c\_selection - 1 + i) < BOARD\_SIZE\_a && (r\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-uright

nearbymines5 = check\_for\_nearby\_mines\_a((r\_selection - 1 - i), (c\_selection - 1 + i));

if(nearbymines5 != -1)

{

game\_board\_a[(r\_selection - 1) - i][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines5);

i++;

}

}

}

if(r\_selection != BOARD\_SIZE\_a)

{

i = 0;

while(nearbymines6 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_a && (c\_selection - 1 + i) < BOARD\_SIZE\_a )

{

// This is checking elements to the diagonal-dright

nearbymines6 = check\_for\_nearby\_mines\_a((r\_selection - 1 + i), (c\_selection - 1 + i));

if(nearbymines != -1)

{

game\_board\_a[(r\_selection - 1) + i][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines6);

i++;

}

}

}

}

if(r\_selection != BOARD\_SIZE\_a)

{

i = 0;

while(nearbymines2 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_a)

{

// This is checking elements heading down

nearbymines2 = check\_for\_nearby\_mines\_a((r\_selection - 1 + i), c\_selection - 1);

if(nearbymines2 != -1)

{

game\_board\_a[(r\_selection - 1) + i][c\_selection - 1] = (char) ( ((int)'0') + nearbymines2 );

i++;

}

}

if(c\_selection != BOARD\_SIZE\_a)

{

i = 0;

while(nearbymines7 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_a && (c\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-dleft

nearbymines7 = check\_for\_nearby\_mines\_a((r\_selection - 1 + i), (c\_selection - 1 - i));

if(nearbymines != -1)

{

game\_board\_a[(r\_selection - 1) + i][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines7);

i++;

}

}

}

}

if(r\_selection != 1)

{

i = 0;

while(nearbymines3 == 0 && (r\_selection - i) > 0)

{

// This is checking elements heading up

nearbymines3 = check\_for\_nearby\_mines\_a((r\_selection - 1 - i), c\_selection - 1);

if(nearbymines3 != -1)

{

game\_board\_a[(r\_selection - 1) - i][c\_selection - 1] = (char) ( ((int)'0') + nearbymines3 );

i++;

}

}

if(c\_selection != BOARD\_SIZE\_a)

{

while(nearbymines8 == 0 && (c\_selection - 1 - i) > 0 && (r\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-uleft

nearbymines8 = check\_for\_nearby\_mines\_a((r\_selection - 1 - i), (c\_selection - 1 - i));

if(nearbymines8 != -1)

{

game\_board\_a[(r\_selection - 1) - i][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines8);

i++;

}

}

}

}

if(c\_selection != 1)

{

i = 0;

while(nearbymines4 == 0 && (c\_selection - i) > 0)

{

// This is checking elements to the left

nearbymines4 = check\_for\_nearby\_mines\_a(r\_selection - 1, (c\_selection - 1 - i));

if(nearbymines4 != -1)

{

game\_board\_a[r\_selection - 1][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines4 );

i++;

}

}

}

}

// Handles a player winning.

if(check\_win\_game\_a() == TRUE)

{

system("cls");

print\_fullboard\_a();

printf("\n\nYou've won the game!! Congrats!!\n\n");

play\_again\_a();

}

return 0;

}

int play\_game\_p()

{

int r\_selection = 0, c\_selection = 0,

nearbymines = 0, nearbymines2 = 0,

nearbymines3 = 0, nearbymines4 = 0,

nearbymines5 = 0, nearbymines6 = 0,

nearbymines7 = 0, nearbymines8 = 0,

i = 0;

// Recieves data from the user, first the row number, then the column number...

// I could think of other ways to do it, but this one seemed easiest.

do {

printf("\nMake a selection (ie. row [ENTER] col): \n");

printf("Row--> ");

scanf("%d", &r\_selection);

printf("Col--> ");

scanf("%d", &c\_selection);

} while(r\_selection < 1 || r\_selection > BOARD\_SIZE\_p || c\_selection < 1 || c\_selection > BOARD\_SIZE\_p);

// ^ Checks for any invalid input statements from user.

check\_for\_mine\_p(r\_selection - 1, c\_selection - 1);

if(lost == 1)

return -1;

// Checks for nearby mines at every direction from user input location.

//Assigns that location the number of mines found nearby, updating the board.

nearbymines = check\_for\_nearby\_mines\_p(r\_selection - 1, c\_selection - 1);

game\_board\_p[r\_selection - 1][c\_selection - 1] = (char)( ((int)'0') + nearbymines );

// The following checks for mines nearby elements

// in the array with no mines.. it's a continuous

// loop until either a mine is found, or we

// reach the end of the array & it cannot be checked any further.

// It also changes the game\_board[] with '0' if no mines are found. Very useful piece of code.

// It checks all elements left, right, up, down and all diagonal directions.

// By running a function that checks these same directions.

// A bit much to follow, though. I'm sure there's a much better way. I just don't know it yet.

if(nearbymines == 0)

{

if(c\_selection != BOARD\_SIZE\_p)

{

i = 0;

while(nearbymines == 0 && (c\_selection - 1 + i) < BOARD\_SIZE\_p)

{

// This is checking elements to the right

nearbymines = check\_for\_nearby\_mines\_p(r\_selection - 1, (c\_selection - 1 + i));

if(nearbymines != -1)

{

game\_board\_p[r\_selection - 1][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines );

i++;

}

}

if(r\_selection != 1)

{

i = 0;

while(nearbymines5 == 0 && (c\_selection - 1 + i) < BOARD\_SIZE\_p && (r\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-uright

nearbymines5 = check\_for\_nearby\_mines\_p((r\_selection - 1 - i), (c\_selection - 1 + i));

if(nearbymines5 != -1)

{

game\_board\_p[(r\_selection - 1) - i][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines5);

i++;

}

}

}

if(r\_selection != BOARD\_SIZE\_p)

{

i = 0;

while(nearbymines6 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_p && (c\_selection - 1 + i) < BOARD\_SIZE\_p )

{

// This is checking elements to the diagonal-dright

nearbymines6 = check\_for\_nearby\_mines\_p((r\_selection - 1 + i), (c\_selection - 1 + i));

if(nearbymines != -1)

{

game\_board\_p[(r\_selection - 1) + i][(c\_selection - 1) + i] = (char) ( ((int)'0') + nearbymines6);

i++;

}

}

}

}

if(r\_selection != BOARD\_SIZE\_p)

{

i = 0;

while(nearbymines2 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_p)

{

// This is checking elements heading down

nearbymines2 = check\_for\_nearby\_mines\_p((r\_selection - 1 + i), c\_selection - 1);

if(nearbymines2 != -1)

{

game\_board\_p[(r\_selection - 1) + i][c\_selection - 1] = (char) ( ((int)'0') + nearbymines2 );

i++;

}

}

if(c\_selection != BOARD\_SIZE\_p)

{

i = 0;

while(nearbymines7 == 0 && (r\_selection - 1 + i) < BOARD\_SIZE\_p && (c\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-dleft

nearbymines7 = check\_for\_nearby\_mines\_p((r\_selection - 1 + i), (c\_selection - 1 - i));

if(nearbymines != -1)

{

game\_board\_p[(r\_selection - 1) + i][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines7);

i++;

}

}

}

}

if(r\_selection != 1)

{

i = 0;

while(nearbymines3 == 0 && (r\_selection - i) > 0)

{

// This is checking elements heading up

nearbymines3 = check\_for\_nearby\_mines\_p((r\_selection - 1 - i), c\_selection - 1);

if(nearbymines3 != -1)

{

game\_board\_p[(r\_selection - 1) - i][c\_selection - 1] = (char) ( ((int)'0') + nearbymines3 );

i++;

}

}

if(c\_selection != BOARD\_SIZE\_p)

{

while(nearbymines8 == 0 && (c\_selection - 1 - i) > 0 && (r\_selection - 1 - i) > 0)

{

// This is checking elements to the diagonal-uleft

nearbymines8 = check\_for\_nearby\_mines\_p((r\_selection - 1 - i), (c\_selection - 1 - i));

if(nearbymines8 != -1)

{

game\_board\_p[(r\_selection - 1) - i][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines8);

i++;

}

}

}

}

if(c\_selection != 1)

{

i = 0;

while(nearbymines4 == 0 && (c\_selection - i) > 0)

{

// This is checking elements to the left

nearbymines4 = check\_for\_nearby\_mines\_p(r\_selection - 1, (c\_selection - 1 - i));

if(nearbymines4 != -1)

{

game\_board\_p[r\_selection - 1][(c\_selection - 1) - i] = (char) ( ((int)'0') + nearbymines4 );

i++;

}

}

}

}

// Handles a player winning.

if(check\_win\_game\_p() == TRUE)

{

system("cls");

print\_fullboard\_p();

printf("\n\nYou've won the game!! Congrats!!\n\n");

play\_again\_p();

}

return 0;

}

/\* Check whether user input has selected a mine \*/

void check\_for\_mine\_b(int r\_select, int c\_select)

{

if(board\_b[r\_select][c\_select] == '\*')

{

printf("\nYou've hit a mine! You lose!\n");

getchar(); getchar();

lost = 1;

}

}

void check\_for\_mine\_a(int r\_select, int c\_select)

{

if(board\_a[r\_select][c\_select] == '\*')

{

printf("\nYou've hit a mine! You lose!\n");

getchar(); getchar();

lost = 1;

}

}

void check\_for\_mine\_p(int r\_select, int c\_select)

{

if(board\_p[r\_select][c\_select] == '\*')

{

printf("\nYou've hit a mine! You lose!\n");

getchar(); getchar();

lost = 1;

}

}

//Another ridiculous function to find nearby mines.

//I know, I know...it's messy, and needs a rewrite.

int check\_for\_nearby\_mines\_b(int r\_select, int c\_select)

{

int nearby\_mine\_count = 0;

if(board\_b[r\_select][c\_select] == '\*')

return -1;

// Check for mines below and to the right.

if(r\_select < (BOARD\_SIZE\_b - 1) && c\_select < (BOARD\_SIZE\_b - 1))

{

// Check for mine below

if(board\_b[r\_select + 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine to the right.

if(board\_b[r\_select][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dright.

if(board\_b[r\_select + 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check whether the columns to the left can be checked

if(c\_select != 0)

{

// Check for mine diagonal-dleft

if(board\_b[r\_select + 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine to the left

if(board\_b[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

// Check whether the rows above can be checked

if(r\_select != 0)

{

// Check for mine above

if(board\_b[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-uright

if(board\_b[r\_select - 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check whether columns to the left can be checked

if(c\_select != 0)

{

// Check for mine diagonal-uleft

if(board\_b[r\_select - 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

}

}

// Check if selection is in last row

if(r\_select == (BOARD\_SIZE\_b - 1) && c\_select != (BOARD\_SIZE\_b - 1))

{

// Check for mine above

if(board\_b[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-uright

if(board\_b[r\_select - 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

}

// Check if selection is in last column

if(c\_select == (BOARD\_SIZE\_b - 1) && r\_select != (BOARD\_SIZE\_b - 1))

{

// Check for mine to the left

if(board\_b[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dleft

if(board\_b[r\_select + 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

// Check whether selection is last in element

if(r\_select == (BOARD\_SIZE\_b - 1) && c\_select == (BOARD\_SIZE\_b - 1))

{

// Check for mine to the left

if(board\_b[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dleft

if(board\_b[r\_select - 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine above

if(board\_b[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

}

return nearby\_mine\_count;

}

int check\_for\_nearby\_mines\_a(int r\_select, int c\_select)

{

int nearby\_mine\_count = 0;

if(board\_a[r\_select][c\_select] == '\*')

return -1;

// Check for mines below and to the right.

if(r\_select < (BOARD\_SIZE\_a - 1) && c\_select < (BOARD\_SIZE\_a - 1))

{

// Check for mine below

if(board\_a[r\_select + 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine to the right.

if(board\_a[r\_select][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dright.

if(board\_a[r\_select + 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check whether the columns to the left can be checked

if(c\_select != 0)

{

// Check for mine diagonal-dleft

if(board\_a[r\_select + 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine to the left

if(board\_a[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

// Check whether the rows above can be checked

if(r\_select != 0)

{

// Check for mine above

if(board\_a[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-uright

if(board\_a[r\_select - 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check whether columns to the left can be checked

if(c\_select != 0)

{

// Check for mine diagonal-uleft

if(board\_a[r\_select - 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

}

}

// Check if selection is in last row

if(r\_select == (BOARD\_SIZE\_a - 1) && c\_select != (BOARD\_SIZE\_a - 1))

{

// Check for mine above

if(board\_a[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-uright

if(board\_a[r\_select - 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

}

// Check if selection is in last column

if(c\_select == (BOARD\_SIZE\_a - 1) && r\_select != (BOARD\_SIZE\_a - 1))

{

// Check for mine to the left

if(board\_a[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dleft

if(board\_a[r\_select + 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

// Check whether selection is last in element

if(r\_select == (BOARD\_SIZE\_a - 1) && c\_select == (BOARD\_SIZE\_a - 1))

{

// Check for mine to the left

if(board\_a[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dleft

if(board\_a[r\_select - 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine above

if(board\_a[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

}

return nearby\_mine\_count;

}

int check\_for\_nearby\_mines\_p(int r\_select, int c\_select)

{

int nearby\_mine\_count = 0;

if(board\_p[r\_select][c\_select] == '\*')

return -1;

// Check for mines below and to the right.

if(r\_select < (BOARD\_SIZE\_p - 1) && c\_select < (BOARD\_SIZE\_p - 1))

{

// Check for mine below

if(board\_p[r\_select + 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine to the right.

if(board\_p[r\_select][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dright.

if(board\_p[r\_select + 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check whether the columns to the left can be checked

if(c\_select != 0)

{

// Check for mine diagonal-dleft

if(board\_p[r\_select + 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine to the left

if(board\_p[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

// Check whether the rows above can be checked

if(r\_select != 0)

{

// Check for mine above

if(board\_p[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-uright

if(board\_p[r\_select - 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

// Check whether columns to the left can be checked

if(c\_select != 0)

{

// Check for mine diagonal-uleft

if(board\_p[r\_select - 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

}

}

// Check if selection is in last row

if(r\_select == (BOARD\_SIZE\_p - 1) && c\_select != (BOARD\_SIZE\_p - 1))

{

// Check for mine above

if(board\_p[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-uright

if(board\_p[r\_select - 1][c\_select + 1] == '\*')

nearby\_mine\_count++;

}

// Check if selection is in last column

if(c\_select == (BOARD\_SIZE\_p - 1) && r\_select != (BOARD\_SIZE\_p - 1))

{

// Check for mine to the left

if(board\_p[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dleft

if(board\_p[r\_select + 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

}

// Check whether selection is last in element

if(r\_select == (BOARD\_SIZE\_p - 1) && c\_select == (BOARD\_SIZE\_p - 1))

{

// Check for mine to the left

if(board\_p[r\_select][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine diagonal-dleft

if(board\_p[r\_select - 1][c\_select - 1] == '\*')

nearby\_mine\_count++;

// Check for mine above

if(board\_p[r\_select - 1][c\_select] == '\*')

nearby\_mine\_count++;

}

return nearby\_mine\_count;

}

/\* Check if user has won game \*/

int check\_win\_game\_b()

{

int row, col;

for(row = 0; row < BOARD\_SIZE\_b; row++)

for(col = 0; col < BOARD\_SIZE\_b; col++)

{

if(game\_board\_b[row][col] == 'o' && board\_b[row][col] != '\*')

return FALSE;

}

return TRUE;

}

int check\_win\_game\_a()

{

int row, col;

for(row = 0; row < BOARD\_SIZE\_a; row++)

for(col = 0; col < BOARD\_SIZE\_a; col++)

{

if(game\_board\_a[row][col] == 'o' && board\_a[row][col] != '\*')

return FALSE;

}

return TRUE;

}

int check\_win\_game\_p()

{

int row, col;

for(row = 0; row < BOARD\_SIZE\_p; row++)

for(col = 0; col < BOARD\_SIZE\_p; col++)

{

if(game\_board\_p[row][col] == 'o' && board\_p[row][col] != '\*')

return FALSE;

}

return TRUE;

}

// Ask user if they wish to play again.

void play\_again\_b()

{

char ans;

printf("\n\n Would You Like To Play Again? \n Press 'Y' or 'y' for YES \n Press 'N' or 'n' for NO \n\n");

scanf(" %c", &ans);

if(toupper(ans) == 'Y')

{

system("cls");

start\_b();

}

else

{

printf("\n\nThanks for playing! Bye.");

(void) getchar();

exit(EXIT\_SUCCESS);

}

}

void play\_again\_a()

{

char ans;

printf("\n\n Would You Like To Play Again? \n Press 'Y' or 'y' for YES \n Press 'N' or 'n' for NO \n\n");

scanf(" %c", &ans);

if(toupper(ans) == 'Y')

{

system("cls");

start\_a();

}

else

{

printf("\n\nThanks for playing! Bye.");

(void) getchar();

exit(EXIT\_SUCCESS);

}

}

void play\_again\_p()

{

char ans;

printf("\n\n Would You Like To Play Again? \n Press 'Y' or 'y' for YES \n Press 'N' or 'n' for NO \n\n");

scanf(" %c", &ans);

if(toupper(ans) == 'Y')

{

system("cls");

start\_p();

}

else

{

printf("\n\nThanks for playing! Bye.");

(void) getchar();

exit(EXIT\_SUCCESS);

}

}

// Displays the welcome message, and the GNU License

void display\_welcome()

{

puts("-----------------------WELCOME TO MINESWEEPER!---------------------------\n\n");

puts("--------------------------Instructions-----------------------------------\n");

puts("1. Type coordinates when prompted");

puts("2. If you hit a mine, you die");

puts("3. Do not uncover coordinates you think are mines");

puts("\n\n");

puts("----------------------------GOOD LUCK------------------------------------\n\n");

}

void start\_b()

{

lost = 0; // User hasn't lost yet

// Build both game boards (one for the user to see and the one with the mines).

build\_board\_b();

build\_gboard\_b();

// Start playing game

do

{

play\_game\_b();

print\_board\_b();

} while(lost != 1); // While the user hasn't lost, loop.

// Once user is lost, print the board with all the mines.

print\_fullboard\_b();

// Play again?

play\_again\_b();

}

void start\_a()

{

lost = 0; // User hasn't lost yet

// Build both game boards (one for the user to see and the one with the mines).

build\_board\_a();

build\_gboard\_a();

// Start playing game

do

{

play\_game\_a();

print\_board\_a();

} while(lost != 1); // While the user hasn't lost, loop.

// Once user is lost, print the board with all the mines.

print\_fullboard\_a();

// Play again?

play\_again\_a();

}

void start\_p()

{

lost = 0; // User hasn't lost yet

// Build both game boards (one for the user to see and the one with the mines).

build\_board\_p();

build\_gboard\_p();

// Start playing game

do

{

play\_game\_p();

print\_board\_p();

} while(lost != 1); // While the user hasn't lost, loop.

// Once user is lost, print the board with all the mines.

print\_fullboard\_p();

// Play again?

play\_again\_p();

}