```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <ctime>
using namespace std;
// max values for row and column
const int MAXROW = 10;
const int MAXCOLUMN = 10;
int numGuess = 0;
int numLaser = 0;
int numBaffles = 0;
int DIFFICULTY = 100;
int ZERO = 0;
// direction for laser
enum Direction {LEFT, RIGHT, UP, DOWN};
// difficulty levels
enum Level {BEGINNER = 4, INTERMEDIATE = 7, ADVANCED = 10};
enum Grid {LEFTBAFFLE, RIGHTBAFFLE, EMPTY, LEFTFOUND, RIGHTFOUND};
Grid board [MAXROW][MAXCOLUMN];
// functions used
void menu(char);
void setDifficulty(string);
void setBoard (Grid Grid[][MAXCOLUMN], Level gameLevel);void printBoard(Grid Grid[][MAXCOLUMN], bool foundOnly);
void printScore (int numLaserShot, int numGuess);
void getStartPosition(int laserID, int& row, int& column, Direction& direction);
void getExitPosition (int& laserID, int row, int column, Direction direction);
int trackLaser(Grid Grid[][MAXCOLUMN], int laserID);
bool makeGuess (Grid Grid[][MAXCOLUMN], int row, int column, char direction);
// MAIN
int main() {
string difficulty = " ";
char selection = ' ';
bool validGuess;
// introduce game
cout << "Welcome to Baffle Game!" << endl;
cout << "The goal of this game is to find all the baffles with the lowest score" << endl;
cout << "Score is determined by guesses and laser shots" << endl;
cout << "One guess will add 2 points and one laser shot will add 1 point" << endl << endl;
// select difficulty
cout << "Please enter your desired difficulty" << endl;
cout << "beginner, intermediate, or advanced" << endl;
cout << "beginner = 4 baffles, intermediate = 7 baffles, and advanced = 10 baffles" << endl;
do {
cin >> difficulty;
setDifficulty(difficulty);} while (difficulty != "beginner" && difficulty != "intermediate" && difficulty != "advanced");
// select from menu
do {
cout << "Enter L to do a laser shot" << endl;
cout << "Enter G to guess the position of a baffle" << endl;
cout << "Enter S to print the number of laser shots and guesses you have attempted ";
cout << " along with your current score" << endl;
cout << "Enter P to print out the board" << endl;
cout << "Enter Q to quit playing" << endl;
cout << "Enter C for cheater mode" << endl;
cout << "Cheater mode shows the locations of all the baffles on the board" << endl;
cin >> selection;
menu(selection);
```

```
} while(selection != 'Q' && selection != 'q' && numBaffles != DIFFICULTY);
cout << "YOU WIN!" << endl << endl;
cout << "You're final score was:" << endl;
printScore(numLaser, numGuess);
return 0;
}
// MENU
void menu(char input) {
int row;
int column;
int laserShotNum = 1;
int laserShot = -1;switch(input) {
// laser shot
case 'l':
case 'L':
cout << "Laser shot selected" << endl;
cout << "Enter an integer 0 - 39 to start the money shot: ";
cin >> laserShot;
laserShot = trackLaser(board, laserShot);
printBoard(board,true);
cout << "Laser shot #" << laserShotNum << " exited the box at " << laserShot << endl;
cout << " " << endl;
numLaser++;
laserShotNum++; // increment numLaserShot
break;
// guess
case 'g':
case 'G':
cout << "Guess selected" << endl << endl;
cout << "row: ";
cin >> row;
cout << "column: ";
cin >> column;
cout << endl;
makeGuess(board, row, column);
break;
// print score
case 's':case 'S':
cout << "Print score selected" << endl << endl;
printScore(numLaser, numGuess);
break;
// print board
case 'p':
case 'P':
cout << "Print board selected" << endl << endl;
printBoard(board, true);
break;
// quit
case 'q':
case 'Q':
cout << "Better luck next time" << endl << endl;
return 0;
// you filthy cheater
case 'c':
case 'C':
cout << "Entering cheater mode" << endl << endl;
printBoard(board, false);
```

```
break;
default:
cout << "Invalid input please enter an option from the menu" << endl;
}
}
// SET DIFFICULTY
void setDifficulty(string difficulty) {
if (difficulty == "beginner") {
cout << "beginner selected" << endl << endl;DIFFICULTY = 4;
setBoard(board, BEGINNER);
} else if (difficulty == "intermediate") {
cout << "intermediate selected" << endl << endl;
DIFFICULTY = 7;
setBoard(board, INTERMEDIATE);
} else if (difficulty == "advanced") {
cout << "advanced selected" << endl << endl;
DIFFICULTY = 10;
setBoard(board, ADVANCED);
} else {
cout << "please enter beginner, intermediate, or advanced" << endl;
}
}
// PRINT BOARD
void printBoard(Grid board[][MAXCOLUMN], bool foundOnly) {
cout << " ";
for (int i = MAXROW; i < MAXROW + MAXCOLUMN; i++) {
//top row numbers
cout << " " << setw(2) << i;
}
cout << endl;
for (int row = 0; row < MAXROW; row++) {
//left numbers
cout << setw(2) << (MAXROW - row - 1) << " |";for (int col = 0; col < MAXCOLUMN; col++) {
if (foundOnly == true) {
switch(board[row][col]) {
case EMPTY:
case LEFTBAFFLE:
case RIGHTBAFFLE:
cout << " _ ";
break;
case LEFTFOUND:
cout << " \\ ";
break;
case RIGHTFOUND:
cout << " / ";
break;
}
} else if (foundOnly == false) {
switch(board[row][col]) {
case EMPTY:
cout << " _ ";
break;
case LEFTBAFFLE:
cout << " \\ ";
break;
case RIGHTBAFFLE:
cout << " / ";
```

```
break;case LEFTFOUND:
cout << " \\ ";
break;
case RIGHTFOUND:
cout << " / ";
break;
}
}
cout << "|";
}
// right side numbers
cout << " " << row + (MAXROW + MAXCOLUMN);
cout << endl;
}
cout << " ";

for (int i = (2 * (MAXROW + MAXCOLUMN)) - 1; i >= (2 * MAXROW) + MAXCOLUMN; i--) {
// bottom numbers
cout << " " << setw(2) << i;
}
cout << endl << endl;
}
// PRINT SCORE
void printScore (int numLaserShot, int numGuess) {

cout << setw(20) << left << "Number of shots:" << numLaserShot << endl;
cout << setw(20) << left << "Number of guesses:" << numGuess << endl;
int totalScore = 0;
totalScore += (2 * numGuess) + numLaserShot;
cout << setw(20) << left << "Current score:" << totalScore << endl << endl;
}
// SET BOARD
void setBoard (Grid board[][MAXCOLUMN], Level gameLevel) {
// initialize board
for (int row = 0; row < MAXROW; row++) {
for (int col = 0; col < MAXCOLUMN; col++) {
board[row][col] = EMPTY;
}
}
// seed game for beginner play
if(gameLevel == BEGINNER) {
int row = 0;
int column = 0;
for (int i = 0; i < BEGINNER; i++) {
row = (rand() % 10);
column = (rand() % 10);
int randomBaffle = (rand() % 1000);if (randomBaffle <= 500) {
board[row][column] = Grid(LEFTBAFFLE);
} else if (randomBaffle > 500) {
board[row][column] = Grid(RIGHTBAFFLE);
}
}
}
// seed game for intermediate play
if (gameLevel == INTERMEDIATE) {
int row = 0;
int column = 0;
```

```
for (int i = 0; i < INTERMEDIATE; i++) {
row = (rand() % 10);
column = (rand() % 10);
int randomBaffle = (rand() % 1000);
if (randomBaffle <= 500) {
board[row][column] = Grid(LEFTBAFFLE);
} else if (randomBaffle > 500) {
board[row][column] = Grid(RIGHTBAFFLE);
}
}
}
// seed game for advanced play
if (gameLevel == ADVANCED) {
int row = 0;
int column = 0;for (int i = 0; i < ADVANCED; i++) {
row = (rand() % 10);
column = (rand() % 10);
int randomBaffle = (rand() % 1000);
if (randomBaffle <= 500) {
board[row][column] = Grid(LEFTBAFFLE);
} else if (randomBaffle > 500) {
board[row][column] = Grid(RIGHTBAFFLE);
}
}
}
}
// GET START POSITION OF LASER
void getStartPosition(int laserID, int& row, int& column, Direction& direction) {
// converts the laserID to an index in our array while determining direction
// 30 - 39
if (laserID >= (2 * MAXROW) + MAXCOLUMN) {
direction = UP;
row = MAXROW - 1;
column = laserID - ((2 * MAXROW) + MAXCOLUMN);
}
// 20 - 29
else if (laserID >= MAXROW + MAXCOLUMN) {
direction = LEFT;
row = laserID - MAXROW - MAXCOLUMN;
column = MAXCOLUMN - 1;
}
// 10 - 19else if (laserID >= MAXROW) {
direction = DOWN;
row = ZERO;
column = laserID - MAXROW;
}
// 0 - 9
else {
direction = RIGHT;
row = MAXROW - laserID - 1;
column = ZERO;
}
}
// GET EXIT POSITION OF LASER
void getExitPosition (int& laserID, int row, int column, Direction direction) {
// get exit position
if (direction == UP) {
```

```
laserID = column + MAXROW;
}
else if (direction == LEFT) {
laserID = MAXROW - row - 1;
}
else if (direction == DOWN) {
laserID = column + ((2 * MAXROW) + MAXCOLUMN);
}
else if (direction == RIGHT) {laserID = row + MAXROW + MAXCOLUMN;
}
}
// TRACK LASER
int trackLaser(Grid board[][MAXCOLUMN], int laserID) {
int trackRow;
int trackColumn;
Direction trackDirection;
getStartPosition(laserID, trackRow, trackColumn, trackDirection);
while (trackRow < MAXROW && trackColumn < MAXCOLUMN && trackRow >= ZERO && trackColumn
>= ZERO) {
if(board[trackRow][trackColumn] == RIGHTBAFFLE) {
switch(trackDirection) {
case UP:
trackDirection = RIGHT;
break;
case DOWN:
trackDirection = LEFT;
break;
case LEFT:
trackDirection = DOWN;
break;
default:
trackDirection = UP;}
}
if(board[trackRow][trackColumn] == LEFTBAFFLE) {
switch(trackDirection) {
case UP:
trackDirection = LEFT;
break;
case DOWN:
trackDirection = RIGHT;
break;
case LEFT:
trackDirection = UP;
break;
default:
trackDirection = DOWN;
}
}
switch (trackDirection) {
case UP:
trackRow--;
break;
case DOWN:
trackRow++;
break;
case LEFT:
trackColumn--;
```

```
break;default:
trackColumn++;
}
}
getExitPosition(laserID, trackRow, trackColumn, trackDirection);
return laserID;
}
// MAKE GUESS
bool makeGuess (Grid board[][MAXCOLUMN], int row, int column) {
// convert user row and column values to array indexes
if(row < MAXROW){
row = (MAXROW - 1) - row;
} else if(row >= (MAXROW + MAXROW) && row <= ((MAXROW + (2 * MAXROW)) - 1)){
row = (MAXROW + (MAXROW - 1)) - row;
}
if(column >= MAXCOLUMN && column <= ((MAXCOLUMN * 2) - 1)){
column = column - MAXCOLUMN;
}
else if(column >= (MAXCOLUMN * 3) && column <= ((MAXCOLUMN * 4 ) - 1)){
column = ((MAXCOLUMN * 4 ) - 1) - column;
}
if (row < ZERO || row >= 2 * (MAXROW + MAXCOLUMN)) {
cout << "invalid first coordinate" << endl;return false;
}
if (column < ZERO || column >= 2 * (MAXROW + MAXCOLUMN) || column == row) {
cout << "invalid second coordinate" << endl;
return false;
}
// check if a move has already been made
if (board[row][column] == LEFTFOUND || board[row][column] == RIGHTFOUND) {
cout << "You have already attempted a guess at this position" << endl;
return false;
}
// checking if a guess is found
else if (board[row][column] == LEFTBAFFLE || board[row][column] == RIGHTBAFFLE) {
numGuess++;
cout << "you hit a baffle on guess " << numGuess << endl;
if (board[row][column] == LEFTBAFFLE) {
board[row][column] = LEFTFOUND;
}
if (board[row][column] == RIGHTBAFFLE) {
board[row][column] = RIGHTFOUND;
}
numBaffles++;
numGuess++;
return true;} else {
numGuess++;
cout << "this is guess number " << numGuess << " you did not hit a baffle" << endl;
return false;
}
}
```