

# **Group Project 1**

Title

**Space Invaders**

Course

**CIS/CSC-17B**

Section

**40502**

Date

**April 27th, 2020**

Authors

**Brandon Sanchez / Matthew Borja**

**Dolores / Christopher Alexman**

**Martin Perina / Malay Patel**

# Introduction

Space Invaders was an extremely popular arcade game when it was released in 1978. The game was originally created by the Japanese video game developer Tomohiro Nishikado. Even though it is an old game, many different ports have been made to play on modern PC's. The player controls a spaceship at the bottom of the screen that can be moved left and right. To win, the player must kill all of the enemies on the screen with their laser. A player loses if they get hit by enemy lasers three times during the playthrough.



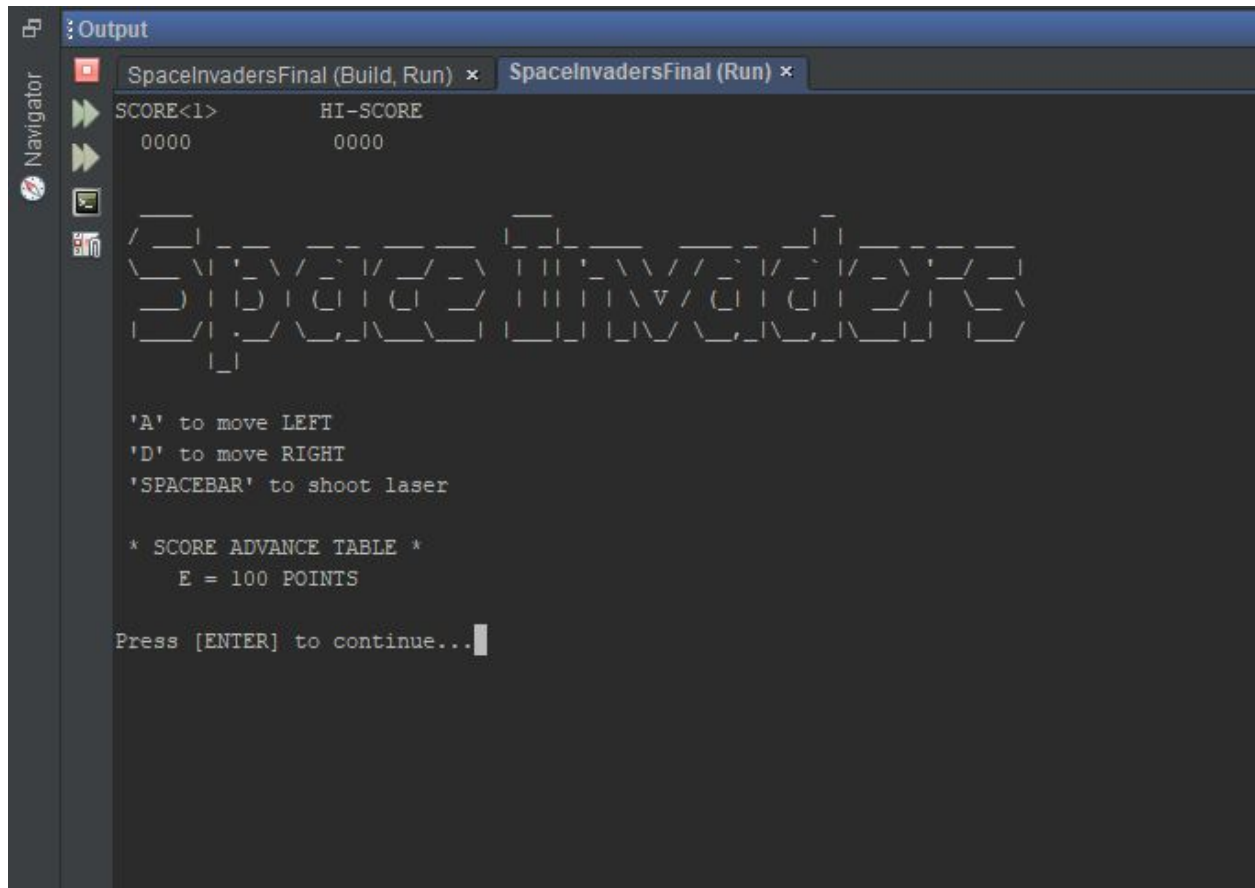
# Checklist

This program meets the criteria for the first project. It meets the base requirement of making a game using the following concepts below. There is an aspect of user admin and server client relations and the model view controller design style.

Those ideas include:

- model view controller
- objects in C++
- Reading/Writing Files
- User-Admin
- Client-Server

# Gameplay with Sample IO



The screenshot shows a development environment with a dark theme. On the left is a 'Navigator' sidebar with icons for file explorer, search, and other tools. The main area is titled 'Output' and contains the game's start screen. The start screen has a title bar with two tabs: 'SpacInvadersFinal (Build, Run) x' and 'SpacInvadersFinal (Run) x'. The game content is displayed in a monospaced font. At the top, it shows 'SCORE<1>' and 'HI-SCORE' both with values of '0000'. Below this is the title 'SpacInvaders' in a large, stylized, blocky font. The controls are listed: 'A' to move LEFT, 'D' to move RIGHT, and 'SPACEBAR' to shoot laser. A scoring scale is shown: '\* SCORE ADVANCE TABLE \*' followed by 'E = 100 POINTS'. At the bottom, it says 'Press [ENTER] to continue...' with a cursor.

```
SCORE<1>      HI-SCORE
0000          0000

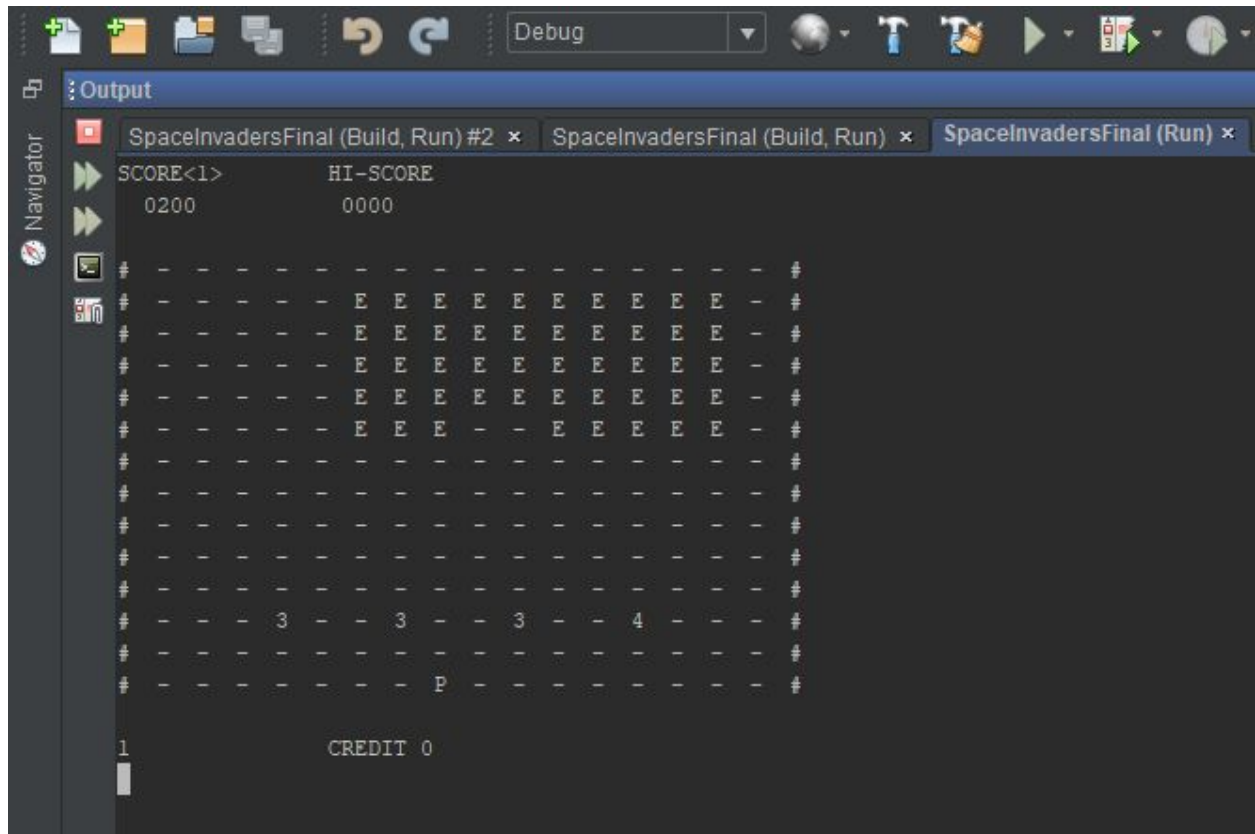
SpacInvaders

'A' to move LEFT
'D' to move RIGHT
'SPACEBAR' to shoot laser

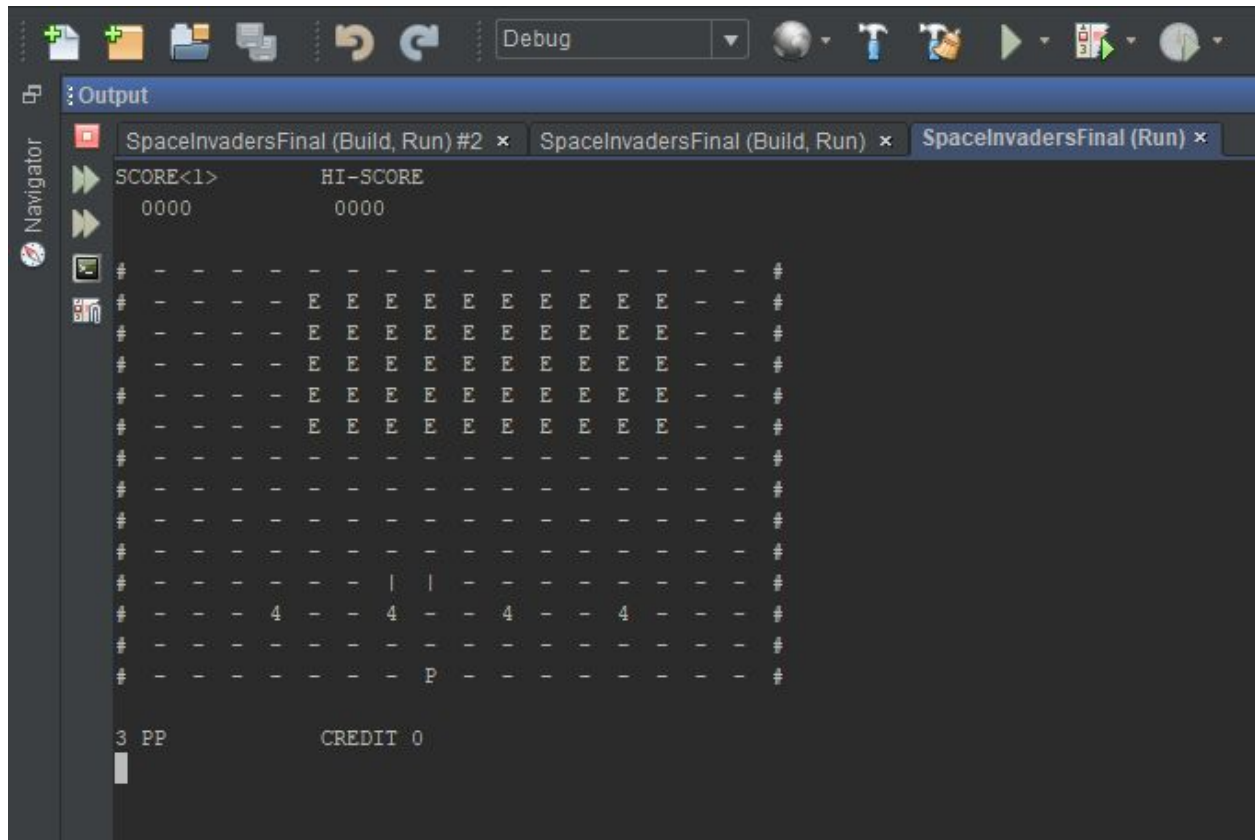
* SCORE ADVANCE TABLE *
  E = 100 POINTS

Press [ENTER] to continue...
```

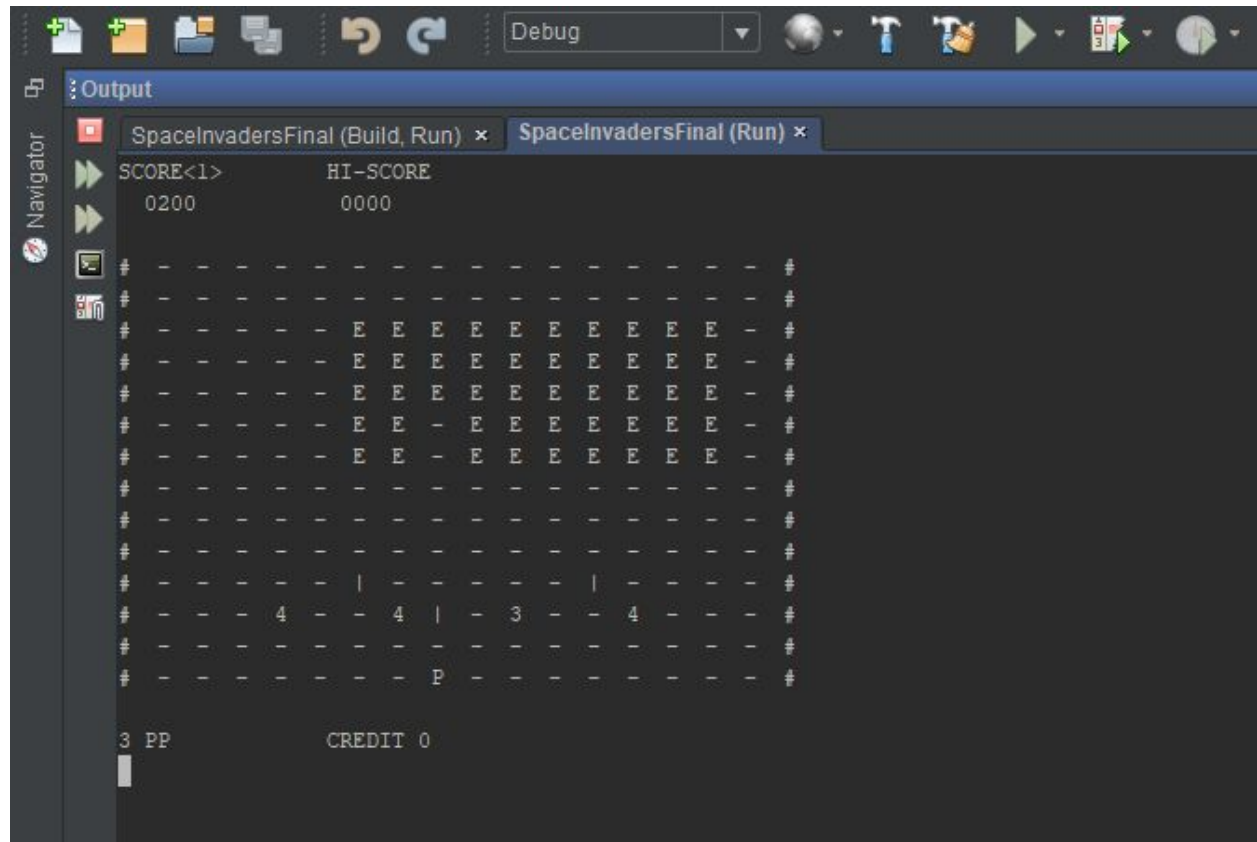
This is the start screen when the game gets loaded, it shows the controls and scoring scale. A moves the player left while D moves the player right and Spacebar shoots the player's laser.



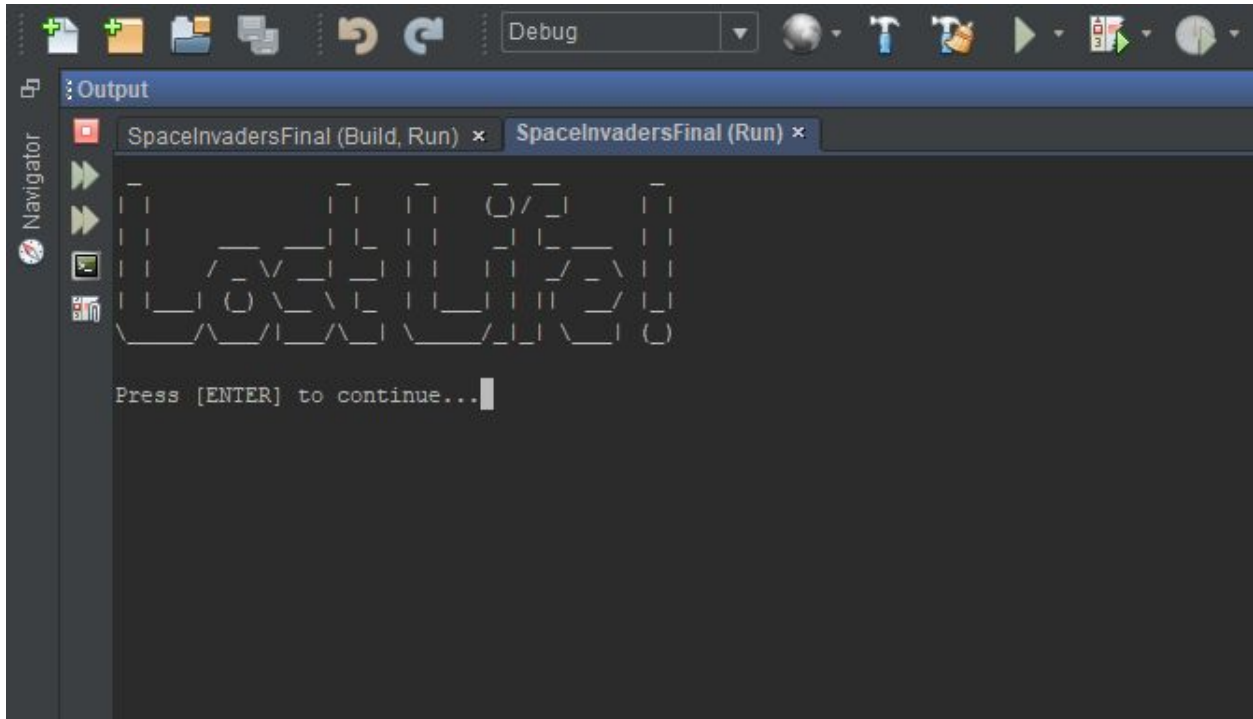
This shows how the enemies move as a group even when some of them are missing.



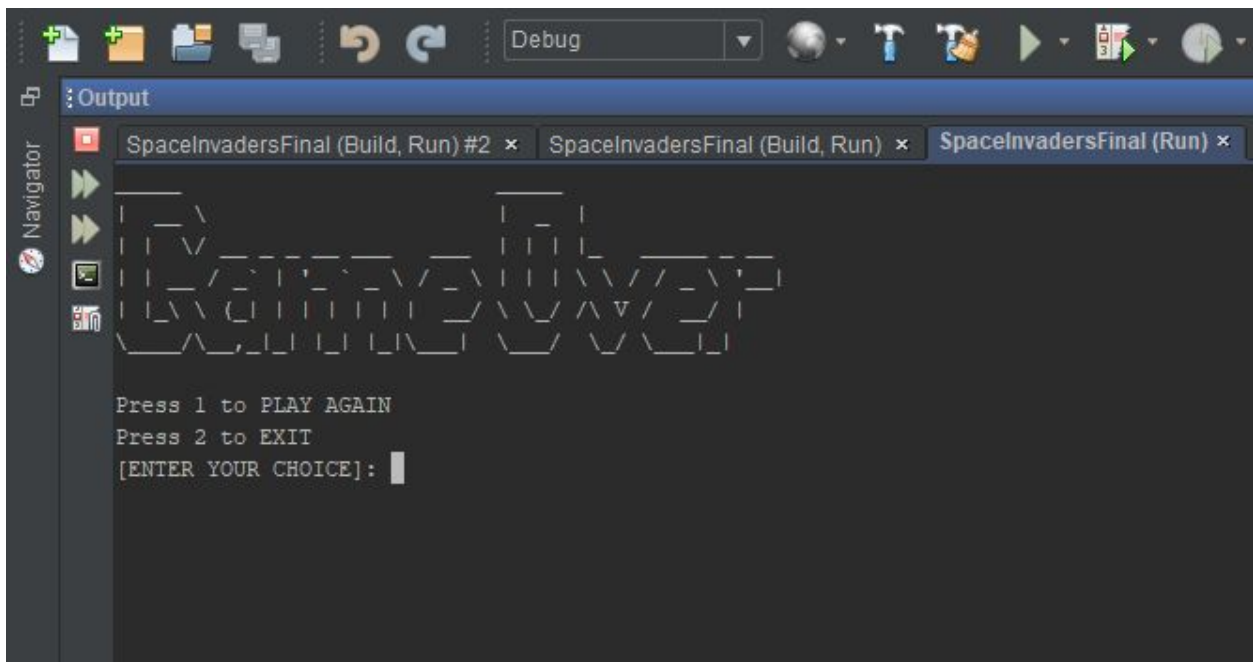
This was taken after a few frames passed, the enemies moved to the right as a group and two of them shot lasers at the player. The choice of which enemy shoots is random.



The player can also shoot their laser while the enemy lasers are on the field.

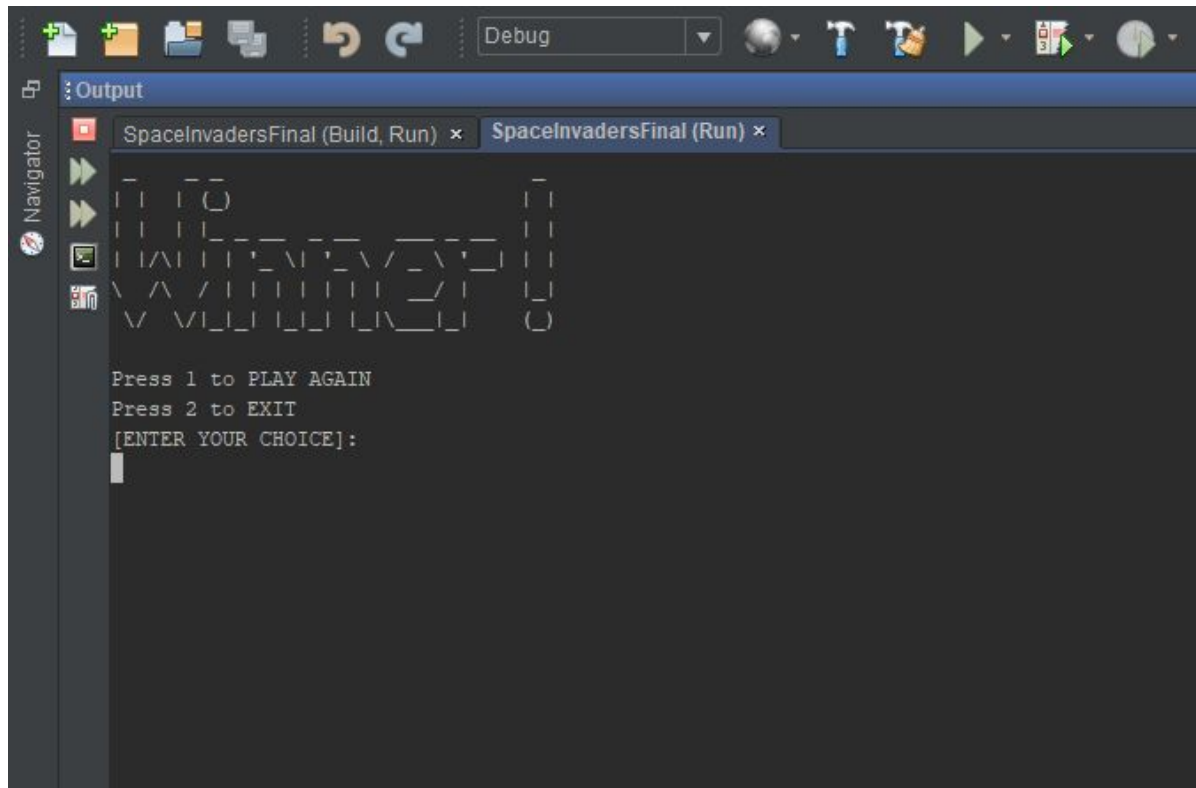


When the player is hit by an enemy laser this screen shows up. After a couple seconds it goes away and the board is shown again with the remaining enemies respawned.



When the player loses all of their lives this screen shows up asking if the player would like to continue with the same number of enemies but with their lives reset to three.





This screen indicates that all the enemies were defeated and the player won the game.

```
update_4-26_matthew
48, 16
*****
Matthew 100
*****
Brandon 100
*****
Chris 100
*****
SCORE<1>    HI-SCORE
5000        5000

HI-SCORER REGISTRATION
ENTER NAME (10 letters max):      dadaddddaaaaaadddd      d
                                ddda aaa
```

This shows up after the player wins to show what the current high scores are and who they belong to. If the player gets a score higher than those high scores then they are prompted to input their name as shown.

## Development Summary

**Lines of Code: # 1679**

### Chris

I chose the game Space Invaders because I wanted to see how well a game with a decently high refresh rate would translate with C++ into an output console. It is also a classic arcade game so the chance to have a stab at what the mechanics behind it might be was a nice opportunity. The thing that my group and I struggled with the most was just working through the logic and debugging of the code. There were moving enemies and lasers so there was a lot to keep track of. I was the one who primarily wrote the board class and its functions. It was a good

role and really challenged my logical thinking in the details and the big picture. I learned that time management and communication are the most important things for a group project. I also feel that I got a better grasp on classes by doing this project.

## Brandon

I chose Space Invaders for this project because it was the only game I have played out of the pool. It was my first time working on a large piece of code so working with a group was the biggest challenge for me. I started the project off by creating a simple function that would move the character 'X' across the board. The rest of the group added significantly on to this. I implemented the functions to check when a player wins/loses. I liked how we got the game working but I heavily dislike the flashing that often occurs when outputting the board. I learned a lot more about Github and how to work on code as a team.

## Matt

I chose Space Invaders because I was familiar with it, remember playing it once or twice in an arcade when I was younger, and I felt most confident out of the options in contributing to the creation of the code. I struggled most with the architecture of the file operations for this project. Beginning with the leaderboard class, I coded parts of the iconic user interface of Space Invaders, that being the scores, lives, and credit display above and below the board. In the final stage of development, I worked on the file operations of the hi-score system. I am proud of our rendition of Space Invaders, however am perturbed by the bloat of 'a', 'd' and space keystrokes when outside of the game board. What I am proud to take away from this project is a better understanding regarding the writing and reading of a variable number of records from binary files.

# Malay

One of my original game choices to propose was Galaga, so when Space Invaders was nominated I thought that it was close enough. I definitely struggled with enemy movement and I think the group did too since after we got that finished the rest of the project went smoothly. Originally I worked on enemy movement but I could not figure it out so I asked the group for help, then I worked on Dynamically spawning the enemy and changing their speed as they died out. I learned more about dynamically creating memory and various ways to parse through a 2D-Array.

# Martin

The game is an old classic arcade game, I thought that programming it would be an interesting challenge. As a group I believe our biggest struggle was debugging the enemy movement and lasers as well as any trailing problems. I was pleased with my finished contributions, those being the displayed banners upon losing a life, winning, and losing as well as partly managing the main gameplay loop. Not that I disliked it, but I think that the main loop could be updated for the next step of the project. It could be more streamlined and not rely on outside functions for exiting. I learned more about implementing game logic into programming as a whole, especially logic loops.

## Goals for Project 2

- Using pictures or some other form of representing the game objects
- Including buttons for user input
- Improve main gameplay loop (loop for number of enemies and player lives)
- Overall , make a graphical display instead of terminal

The plan going forward will be to be more consistent with communication and expectation. We will try to distribute the tasks a little more evenly and make weekly efforts to make progress and keep each other accountable. We will also start even earlier so we don't run into a bottleneck at the end with trying to turn everything in at the same time.

Given we have 8 weeks it wouldn't be unreasonable, if we started this week, to finish the majority of everything by week 6 and not run into finals week on top of these projects as well.

# Pseudocode

*main.cpp*

*Create board object*

*Display start screen*

*Create the initial board*

*While the game isn't over*

*Clear the screen*

*Move player laser*

*Move board left, down, or right*

*Move enemy laser(s)*

*Output leaderboard top*

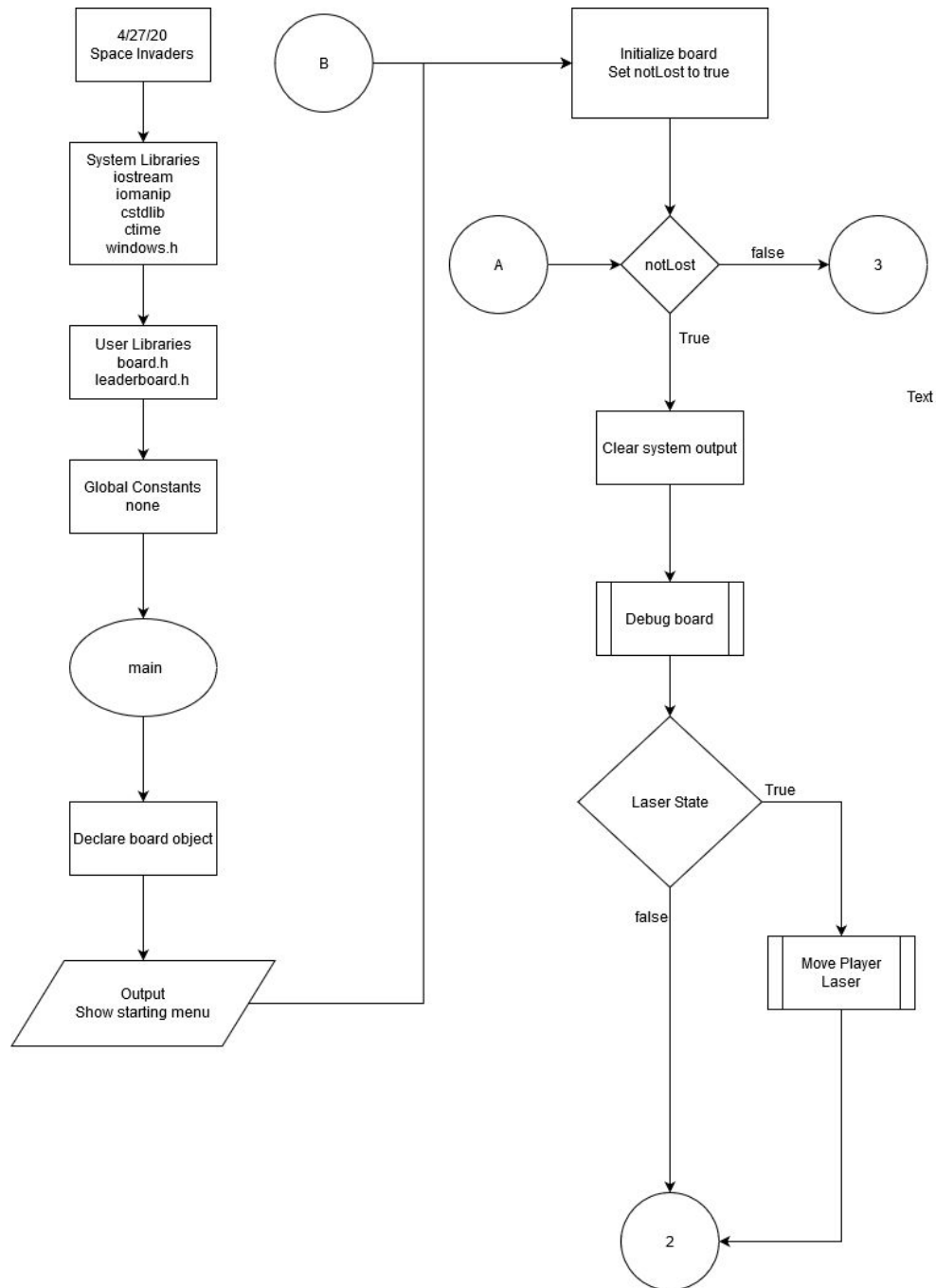
*Output the game board array*

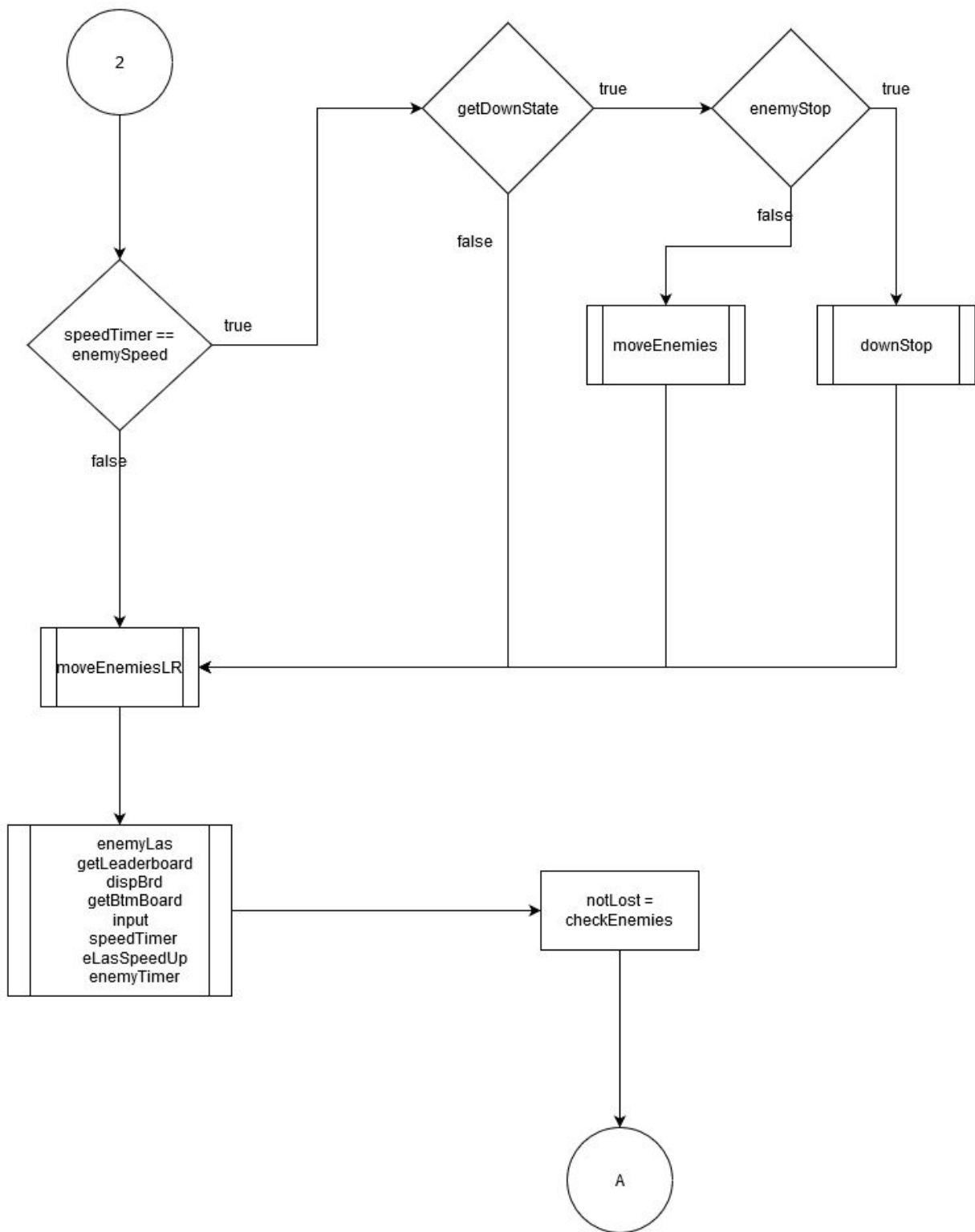
*Output the leaderboard bottom*

*Increment all the timing counters*

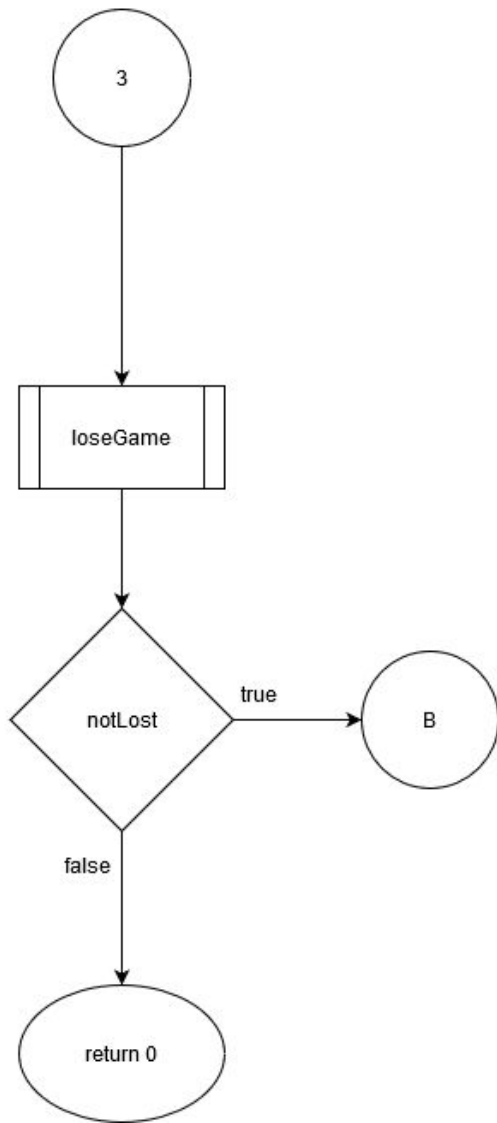
*Check if game is won or lost*

# Flowchart









# Code

header.h

```
// -----  
// This is the header file for the project with all the libraries.  
// -----  
#include <iostream>      // Input - Output Library  
#include <iomanip>        // Various formatting  
#include <cstdlib>        // srand to set the seed  
#include <ctime>          // time for rand  
#include <fstream>        // needed for file operations  
#include <windows.h>      // for getasynkeystate function, Sleep();  
  
using namespace std;
```

record.h

```
// -----  
// This header file holds a structure that is used for the high score file.  
// -----  
const int NAME_SIZE = 11;    // maximum size hi-score name can be (-1 for '/0')  
  
struct Record{  
    char name[NAME_SIZE];  
    int score;  
};
```

leaderboard.h

```
// -----  
// This class holds all of the information for the leaderboard information that  
// goes on the top and bottom of the game board. It is able to add points  
// when the player hits an enemy with a laser and display a current high score  
// from a file.  
// -----  
const int TOP5 = 5;  
  
class Leaderboard  
{  
private:  
    Record recorder;           //Holds hi-score information  
    int size;                  //Size of all hi-score records from file.  
    int index;                  // variable to hold an index  
    int scoreOne;               //Player One's score.  
    int hiScore;                //High Score.  
    int credit;                 //Credits player has.  
public:  
    Leaderboard();              //Default constructor.  
    void addScore(int);          //Add score for kills.  
    void setHiScore();           //Set Hi-Score.  
    int getScore(){ return scoreOne; } //Display Player One's score.  
    int getHiScore(){ return hiScore; } //Display Hi-Score.  
    void getLeaderboard();        //Display scores neatly.  
    void getBtmBoard();           //Display lives & credits  
    void displayLives();          //Determine how many lives to display.  
    int lives;                   //The number of lives the player has.  
    bool hiScores();              // high scores function  
    void displayHiscres();         // display high scores function  
    void registration();           //Get name from user if hi-score reached.  
    void setScoreOne(int);         //For resetting the game  
    void lostLife() { --lives; }   //Lose a life  
    int getLives() { return lives; } //Get current number of lives  
};  
  
Leaderboard::Leaderboard(){      //Default constructor.  
    index=0;  
    scoreOne=0;  
    hiScore=0;  
    lives=3;  
    credit=0;  
}
```

```

void Leaderboard::setScoreOne(int s) { scoreOne = s; } //lets the board class reset the score

void Leaderboard::addScore(int score){           //Accepts score for the kill and
    scoreOne+=score;                             //Adds score to scoreOne.
}

void Leaderboard::setHiScore(){
    if(hiScore<=scoreOne)
        hiScore=scoreOne;
    if(hiScores()){
        registration();
    }
}

void Leaderboard::getLeaderboard(){             //Display the top UI.
    cout<<"SCORE<1>"<<"\tHI-SCORE\n ";
    cout<<setw(4)<<setfill('0')<<scoreOne<<"\t\t "<<setw(4)<<setfill('0')<<hiScore<<endl<<endl;
}

void Leaderboard::getBtmBoard(){                //Display lives & credits below
    cout<<endl<<lives;                          //game board.
    displayLives();                             //Call separate function to display lives
    cout<<"\t\tCREDIT "<<credit<<endl;
}

void Leaderboard::displayLives(){
    switch(lives){
        case 3: cout<<" PP";                    //Determine how many lives to
            break;                               //display based on lives left
        case 2: cout<<" P";                      //in Space Invaders Theme.
            break;
        case 1: cout<<" ";
            break;
    }
}

```

```

bool Leaderboard::hiScores(){
    fstream hiscoreFile;
    hiscoreFile.open("hiscores.dat", ios::in | ios::binary); //Check if file exists.
    if(hiscoreFile.fail()){
        hiscoreFile.open("hiscores.dat", ios::out | ios::binary); //create a new file.
        cout<<"NO HI-SCORES RECORDED YET."<<endl;
        cout<<"NEW HI-SCORE!"<<endl;
        hiscoreFile.close();
        return true;
    }
    else{
        long numBytes;
        hiscoreFile.seekg(0L, ios::end);
        numBytes=hiscoreFile.tellg();
        hiscoreFile.seekg(0L, ios::beg);
        hiscoreFile.close();
        displayHiscores();
        return true;
    }
}

void Leaderboard::displayHiscores(){
    fstream hiscoreFile;
    Record temp[size];
    hiscoreFile.open("hiscores.dat", ios::in | ios::binary);
    hiscoreFile.read(reinterpret_cast<char*>(&temp), sizeof(temp));
    for(int count=0; count<size; count++){
        cout<<"*****"<<endl;
        cout<<setw(12)<<temp[count].name<<setw(6)<<temp[count].score<<endl;
    }
    cout<<"*****"<<endl;
}

void Leaderboard::registration(){
    fstream hiscoresFile;
    char name[NAME_SIZE];
    size++;
    getLeaderboard();
    //reevaluateScores();
    cout<<"\t\tHI-SCORER REGISTRATION\n";

    cout<<"ENTER NAME (10 letters max): ";
    cin.getline(recorder.name, NAME_SIZE);
    recorder.score=hiScore;
    hiscoresFile.open("hiscores.dat", ios::out | ios::app | ios::binary);
    hiscoresFile.write(reinterpret_cast<char*>(&recorder), sizeof(recorder));
}

```

board.h

```
// board.h ////////////////////////////////////////

// -----
// This class holds everything that goes on the game board itself like the
// player, enemies, lasers, shields, and boundaries. The board is able to be
// printed based on how fast the computer running it can handle.
// -----

class Board
{
private:
    // board data
    int row;                // row size of 2D array
    int col;                // col size of 2D array
    char **board;           // dynamic 2D array for game board
    bool gameOver;         // bool to determine if the game is over
    char emptyChar;         // char for empty spaces on the game board
    char boundaryChar;      // char for boundaries on the sides of the board

    // player data
    char plyrChar;          // char for the player on the board
    int plyrX, plyrY;       // x and y of the player in the 2D array

    // enemy data
    char enemyChar;         // char for enemy on the board
    char destroyedEnemyChar; // char for when the enemy is destroyed
    int enemyPoints;        // point value per enemy
    int numEnemies;         // total number of enemies on board
    int moveR;              // number of times the enemies need to move right
    int moveL;              // number of times the enemies need to move left
    int enemySpeed;         // how quickly the enemies move, higher = slower
    int speedTime;          // variable to control enemy speed, increments
    bool downState;         // bool if the enemies need to move down or not
                           // X Y position for the two enemy lasers
    int lasX1, lasX2, lasX3, lasX4, lasX5, lasX6, lasX7, lasX8, lasX9, lasX10;
    int lasY1, lasY2, lasY3, lasY4, lasY5, lasY6, lasY7, lasY8, lasY9, lasY10;
    int netLR;              // the net enemy movement, used for enemy lasers
    int numEnemyLas;        // number of enemy lasers on board at once, max 2
    int eLas1X, eLas1Y;     // enemy las 1's X Y position in 2D array
    int eLas2X, eLas2Y;     // enemy las 2's X Y position in 2D array
    bool eLas1State, eLas2State; // bool for if las 1 or 2 is on the board
    int eLasSpeed;          // speed for the lasers

    // shield data
    char shield1;           // char for the first shield on the board
    char shield2;           // char for the second shield on the board
    char shield3;           // char for the third shield on the board
    char shield4;           // char for the fourth shield on the board
    char sHlth4;            // shield health of '4'
    char sHlth3;            // shield health of '3'
    char sHlth2;            // shield health of '2'
    char sHlth1;            // shield health of '1'

```



```

    int lasX, lasY;                // x and y of the player's laser, vertically bound
    bool lasStateP;                // state of laser, if its on board or not

    // loseGame
    int choice;                    // variable to hold choice

public:
    Board();                       // constructor
    ~Board();                      // destructor

    // create object for the Leaderboard class in the board class
    Leaderboard ldrBrd;

    // class functions
    void startMenu();              // menu to introduce game
    void createBoard();            // starting board conditions
    void dispBrd();                // output the board itself
    bool getGameState() { return gameOver; } // return if game is over or not

    // player functions
    void input();                  // keyboard input of player
    bool getLaserStateP() { return lasStateP; } // if player laser is on the board
    void plyrLas();                // set X Y of laser from the player X Y
    void movPlyrLas();             // move the player laser

    // enemy functions
    void moveEnemiesLR();          // move enemies Left or Right
    void moveEnemiesD();           // move enemies Down
    void speedTimer() { speedTime++; } // increment speed timer
    int getSpeedTimer() { return speedTime; } // get speed timer for main
    int getEnemySpeed() { return enemySpeed; } // get enemy speed for main
    bool checkRow(int, char);      // check if row has enemies in it
    bool checkCol(int);            // check if column has enemies
    void checkSide(char);          // check if sides have enemies
    bool getDownState() { return downState; } // get down state for main
    void setEnemyLasers();         // set up all the enemy lasers X Y
    void enemyTimer();             // how fast enemies are based on how many are left
    void enemyLas();               // randomize which enemies shoot a laser
    void moveEnemyLas1(int, int);  // move enemy laser 1
    void moveEnemyLas2(int, int);  // move enemy laser
    void eLasSpeedUp() { eLasSpeed++; } // keeps track of enemy laser speed
    void checkEnemies();           // double check how many enemies left
    bool enemyStop();              // stop enemies on row before shields
    void downStop() { downState = false; } // stop enemes from moving down

    // game functions
    void loseLife();               // lose a life
    void winGame();                // you win the game
    void loseGame();               // you lose the game

    // debug function
    void debug();                  // housekeeping for the player icon with row movement
};

```



```

// constructor to set size of game board, initialize dynamic 2D array
Board::Board()
{
    // board variables
    row = 14;                // height of board
    col = 18;                // width of board

    // initialize dynamic 2D array
    board = new char*[row];
    for(int i=0; i<row; i++)
        board[i] = new char[col];

    // player variables
    plyrX = plyrY = 0;

    // character variables for the board
    emptyChar = '-';
    boundaryChar = '#';
    plyrChar = 'P';
    lasChar = '|';
    enemyChar = 'E';
    destroyedEnemyChar = 'x';

    // shield variables
    shield1 = shield2 = shield3 = shield4 = '4';    // starting health for shield
    sHlth4 = '4';
    sHlth3 = '3';
    sHlth2 = '2';
    sHlth1 = '1';

    // enemy variables
    numEnemies = 50;
    enemyPoints = 100;
    moveR = 3;
    moveL = 0;
    enemySpeed = 15;
    speedTime = 0;
    downState = false;
    netLR = 0;
    lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
    lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;

    // laser variables
    lasX = lasY = 0;
    lasStateP = false;
    numEnemyLas = 0;
    eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
    eLas1State = eLas2State = false;
    eLasSpeed = 0;

```



```

// this functions sets up the initial conditions of the game board with:
// player, side barriers, enemies, blockades, empty space
void Board::createBoard()
{
    int e = numEnemies; // to dynamically place in enemies

    for(int i=0; i < row; i++){
        for(int j=0; j < col; j++){
            if(i == 13 && j == 8)
            {
                board[i][j] = plyrChar;
                plyrX = i;
                plyrY = j;
                lasX = plyrX;
                lasY = plyrY;
            }
            else if(i == 11 && j == 4)
            {
                board[i][j] = shield1;
            }
            else if(i == 11 && j == 7)
            {
                board[i][j] = shield2;
            }
            else if(i == 11 && j == 10)
            {
                board[i][j] = shield3;
            }
            else if(i == 11 && j == 13)
            {
                board[i][j] = shield4;
            }
            else if (i > 0 && i < 6 && j > 3 && j < 14 && e > 0)
            {
                board[i][j] = enemyChar;
                --e;
            }
            else if(j==0 || j==col-1)
            {
                board[i][j] = boundaryChar;
            }
            else
            {
                board[i][j] = emptyChar;
            }
        }
    }

    setEnemyLasers(); // initial enemy laser positions
}

```

```

// this function simply outputs the current state of the board, or the 2D array
// at the end it clears the screen just to display itself again
void Board::dispBrd()
{
    for(int i=0; i < row; i++){
        for(int j=0; j < col; j++){
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
}

// this function looks for input from the player to move Left and Right, and
// shoot lasers; it has boundary check and laser delay
void Board::input()
{
    if(board[plyrX][plyrY] != plyrChar)
        board[plyrX][plyrY] = plyrChar;

    if(GetAsyncKeyState(0x41) && (board[plyrX][plyrY-1] != '#')){ // 'a'
        board[plyrX][plyrY] = emptyChar;
        board[plyrX][plyrY-1] = plyrChar;
        plyrY--;
    }
    else if(GetAsyncKeyState(0x44) && (board[plyrX][plyrY+1] != '#')){ // 'd'
        board[plyrX][plyrY] = emptyChar;
        board[plyrX][plyrY+1] = plyrChar;
        plyrY++;
    }
    else if(GetAsyncKeyState(0xC0) && lasStateP == false){ // 'SPACEBAR'
        plyrLas();
    }
}

// this function sets the X Y position for the player laser to the player's
// current position
void Board::plyrLas()
{
    lasStateP = true;

    if(board[plyrX][plyrY] == plyrChar)
    {
        lasX = plyrX;
        lasY = plyrY;
    }
}

```

```

// it considers every type of character that could be above it
void Board::movPlyrLas()
{
    // if outside of the board, stop laser, set empty char
    if(lasX-1 < 0)
    {
        board[lasX][lasY] = emptyChar;
        lasX = lasY = 0;
        lasStateP = false;
    }
    // if empty, move laser up one, put empty space where it was
    else if(board[lasX-1][lasY] == emptyChar)
    {
        if(board[lasX][lasY] == plyrChar)
            board[lasX-1][lasY] = lasChar;
        else
        {
            board[lasX][lasY] = emptyChar;
            board[lasX-1][lasY] = lasChar;
        }

        lasX -= 1;
    }
    // if a shield, decrement the enemy shield, keep track of each shield
    // position X Y
    else if(board[lasX-1][lasY] == shield1 || board[lasX-1][lasY] == shield2 ||
            board[lasX-1][lasY] == shield3 || board[lasX-1][lasY] == shield4)
    {
        if(board[lasX-1][lasY] == shield1)
        {
            switch(board[lasX-1][lasY])
            {
                case '4':
                    board[lasX-1][lasY] = shield1 = sHlth3;
                    board[lasX][lasY] = emptyChar;
                    break;
                case '3':
                    board[lasX-1][lasY] = shield1 = sHlth2;
                    board[lasX][lasY] = emptyChar;
                    break;
                case '2':
                    board[lasX-1][lasY] = shield1 = sHlth1;
                    board[lasX][lasY] = emptyChar;
                    break;
                case '1':
                    board[lasX-1][lasY] = shield1 = emptyChar;
                    board[lasX][lasY] = emptyChar;
                    break;
            }
        }
    }
}

```

```

else if(board[lasX-1][lasY] == shield2)
{
    switch(board[lasX-1][lasY])
    {
        case '4':
            board[lasX-1][lasY] = shield2 = sHlth3;
            board[lasX][lasY] = emptyChar;
            break;
        case '3':
            board[lasX-1][lasY] = shield2 = sHlth2;
            board[lasX][lasY] = emptyChar;
            break;
        case '2':
            board[lasX-1][lasY] = shield2 = sHlth1;
            board[lasX][lasY] = emptyChar;
            break;
        case '1':
            board[lasX-1][lasY] = shield2 = emptyChar;
            board[lasX][lasY] = emptyChar;
            break;
    }
}
else if(board[lasX-1][lasY] == shield3)
{
    switch(board[lasX-1][lasY])
    {
        case '4':
            board[lasX-1][lasY] = shield3 = sHlth3;
            board[lasX][lasY] = emptyChar;
            break;
        case '3':
            board[lasX-1][lasY] = shield3 = sHlth2;
            board[lasX][lasY] = emptyChar;
            break;
        case '2':
            board[lasX-1][lasY] = shield3 = sHlth1;
            board[lasX][lasY] = emptyChar;
            break;
        case '1':
            board[lasX-1][lasY] = shield3 = emptyChar;
            board[lasX][lasY] = emptyChar;
            break;
    }
}
else if(board[lasX-1][lasY] == shield4)

```

```

else if(board[lasX-1][lasY] == shield4)
{
    switch(board[lasX-1][lasY])
    {
        case '4':
            board[lasX-1][lasY] = shield4 = sHlth3;
            board[lasX][lasY] = emptyChar;
            break;
        case '3':
            board[lasX-1][lasY] = shield4 = sHlth2;
            board[lasX][lasY] = emptyChar;
            break;
        case '2':
            board[lasX-1][lasY] = shield4 = sHlth1;
            board[lasX][lasY] = emptyChar;
            break;
        case '1':
            board[lasX-1][lasY] = shield4 = emptyChar;
            board[lasX][lasY] = emptyChar;
            break;
    }
}
lasStateP = false;
lasX = lasY = 0;
}
// if its an enemy, replace las and enemy with empty char, add score, stop
// laser on board, decrement number of enemies on board, reset enemy
// laser position to enemy above it using net movement
else if(board[lasX-1][lasY] == enemyChar)
{
    board[lasX-1][lasY] = destroyedEnemyChar;
    board[lasX][lasY] = emptyChar;
    board[lasX-1][lasY] = emptyChar;

    ldrBrd.addScore(enemyPoints);
    lasX = lasY = 0;
    lasStateP = false;
    --numEnemies;

    switch(lasY - 3 + (-1*netLR))
    {
        case 1:
            --lasX1;
            break;
        case 2:
            --lasX2;
            break;
        case 3:
            --lasX3;
            break;
        case 4:
            --lasX4;
            break;
        case 5:

```



```

        case 4:
            --lasX4;
            break;
        case 5:
            --lasX5;
            break;
        case 6:
            --lasX6;
            break;
        case 7:
            --lasX7;
            break;
        case 8:
            --lasX8;
            break;
        case 9:
            --lasX9;
            break;
        case 10:
            --lasX10;
            break;
    }
}
// if its a laser they cancel out, both empty spots
else if(board[lasX-1][lasY] == lasChar)
{
    if(lasX-1 == eLas1X && lasY == eLas1Y)
    {
        board[lasX-1][lasY] = emptyChar;
        eLas1X = eLas1Y = 0;
        eLas1State = false;
    }
    else if(lasX-1 == eLas2X && lasY == eLas2Y)
    {
        board[lasX-1][lasY] = emptyChar;
        eLas2X = eLas2Y = 0;
        eLas2State = false;
    }
    board[lasX-1][lasY] = emptyChar;
    board[lasX][lasY] = emptyChar;
    lasX = lasY = 0;
    lasStateP = false;
}
// otherwise stop the laser
else
{
    lasX = lasY = 0;
    lasStateP = false;
}
}

```



```

// this function moves the group of enemies left or right one spot at a time
// using the timer, it check if each row even has an enemy otherwise it wont
// move the row, it also uses a temp variables to hold the end array spot
// so it can rotate the whole role, it also adjusts the enemy laser positions,
// it also check the side to see if the enemies can move left or right more
void Board::moveEnemiesLR()
{
    if(moveR > 0)
    {
        for(int i = 0; i < row-3; i++)
        {
            if(checkRow(i, 'R'))
            {
                int temp = board[i][col-2];
                for(int j = col-2; j > 0; --j)
                {
                    board[i][j] = board [i][j-1];
                }
                board [i][1] = temp;
            }
        }

        ++lasY1, ++lasY2, ++lasY3, ++lasY4, ++lasY5, ++lasY6, ++lasY7, ++lasY8, ++lasY9, ++lasY10;

        if(moveR == 1)
            checkSide('R');

        if(moveR == 1)
        {
            moveR = 0;
            moveL = 5;
            downState = true;
        }
        else if(moveR > 1)
            moveR--;

        netLR++;
    }
    else if(moveL > 0)
    {
        for(int i = 0; i < row-3; i++)
        {
            if(checkRow(i, 'L'))
            {
                int temp = board[i][1];
                for(int j = 1; j < col-2; j++)
                {
                    board[i][j] = board [i][j+1];
                }
                board [i][col-2] = temp;
            }
        }

        --lasY1, --lasY2, --lasY3, --lasY4, --lasY5, --lasY6, --lasY7, --lasY8, --lasY9, --lasY10;
    }
}

```

```

--lasY1, --lasY2, --lasY3, --lasY4, --lasY5, --lasY6, --lasY7, --lasY8, --lasY9, --lasY10;

if(moveL == 1)
    checkSide('L');

if(moveL == 1)
{
    moveR = 5;
    moveL = 0;
    downState = true;
}
else if(moveL > 1)
    moveL--;

netLR--;
}
speedTime = 0;
}

// this function check the row to see if there is an enemy or other thing in the
// row so it can move it, it corrects if there is a laser character
bool Board::checkRow(int row, char direction)
{
    bool state = false;
    int enemy = 0;

    if(direction == 'R')
    {
        for(int i = 1; i < col-2; i++)
        {
            if(board[row][i] == enemyChar)
                enemy++;
            else if((board[row][i] == shield1 || board[row][i] == shield2 ||
                board[row][i] == shield3 || board[row][i] == shield4) &&
                enemy > 0)
            {
                if(board[row][i-1] == enemyChar)
                    board[row][i] = emptyChar;
                else
                {
                    board[row][i-1] = board[row][i];
                    board[row][i] = emptyChar;
                }
            }
        }
    }
    if(enemy > 0)
    {
        for(int i = 1; i < col-2; i++)
        {
            if(board[row][i] == lasChar)
            {
                if(board[row][i-1] == enemyChar)

```

```

if(enemy > 0)
{
    for(int i = 1; i < col-2; i++)
    {
        if(board[row][i] == lasChar)
        {
            if(board[row][i-1] == enemyChar)
            {
                board[row][i] = emptyChar;
                lasStateP = false;
            }
            else if(board[row][i-1] == emptyChar)
            {
                board[row][i-1] == lasChar;
                board[row][i] == enemyChar;
            }
        }
    }
}

else if (direction == 'L')
{
    for(int i = col-2; i > 1; --i)
    {
        if(board[row][i] == enemyChar)
            enemy++;
        else if((board[row][i] == shield1 || board[row][i] == shield2 ||
            board[row][i] == shield3 || board[row][i] == shield4) &&
            enemy > 0)
        {
            if(board[row][i+1] == enemyChar)
                board[row][i] = emptyChar;
            else
            {
                board[row][i+1] = board[row][i];
                board[row][i] = emptyChar;
            }
            enemy++;
        }
    }
}

if(enemy > 0)
{
    for(int i = 1; i < col-2; i++)
    {
        if(board[row][i] == lasChar)
        {
            if(board[row][i+1] == enemyChar)
            {
                board[row][i] = emptyChar;
                lasStateP = false;
            }
            else if(board[row][i+1] == emptyChar)

```

```

        board[row][i] = emptyChar;
        lasStateP = false;
    }
    else if(board[row][i+1] == emptyChar)
    {
        board[row][i+1] == lasChar;
        board[row][i] == enemyChar;
    }
    }
}

if(enemy > 0)
    state = true;

return state;
}

// this board check the sides next to the boundaries to see if there are enemies
// so the moveLR function knows if it can move the enemies another spot left
// or right
void Board::checkSide(char direction)
{
    int enemy = 0;

    if(direction == 'R')
    {
        for(int i = 0; i < row; i++)
        {
            if(board[i][col-2] == enemyChar)
            {
                enemy++;
            }
        }
        if(enemy == 0)
            moveR++;
    }
    else if(direction == 'L')
    {
        for(int i = 0; i < row; i++)
        {
            if(board[i][1] == enemyChar)
            {
                enemy++;
            }
        }
        if(enemy == 0)
            moveL++;
    }
}

```

```

// this function moves the enemies down, adjusts the enemy laser position,
// and checks each column to adjust everything
void Board::moveEnemiesD()
{
    int temp;

    for(int i = 1; i < col-1; i++)
    {
        if(checkCol(i))
        {
            temp = board[row-1][i];
            for(int j = row-1; j > 0; --j)
            {
                board[j][i] = board[j-1][i];
            }
            board[0][i] = temp;
        }
    }

    ++lasX1, ++lasX2, ++lasX3, ++lasX4, ++lasX5, ++lasX6, ++lasX7, ++lasX8, ++lasX9, ++lasX10;

    downState = false;
}

// this function checks the columns that enemies are in so they can move down,
// it accounts for player, laser, and shield characters so they dont get
// displaced
bool Board::checkCol(int col)
{
    bool state = false;
    int enemy = 0;

    for(int i = 0; i < row; i++)
    {
        if(board[i][col] == enemyChar)
        {
            enemy++;
        }
        else if(board[i][col] == plyrChar)
        {
            board[i][col] = emptyChar;
            board[i-1][col] = plyrChar;

            if(board[i-1][col-1] == plyrChar)
                board[i-1][col-1] = emptyChar;
            if(board[i][col-1] == plyrChar)
                board[i][col-1] = emptyChar;
            if(board[i-1][col+1] == plyrChar)
                board[i-1][col+1] = emptyChar;
            if(board[i][col+1] == plyrChar)
                board[i][col+1] = emptyChar;
        }
        else if((board[i][col] == shield1 || board[i][col] == shield2 ||
            board[i][col] == shield3 || board[i][col] == shield4 ||
            board[i][col] == shield5 || board[i][col] == shield6 ||
            board[i][col] == shield7 || board[i][col] == shield8 ||
            board[i][col] == shield9 || board[i][col] == shield10))
        {
            state = true;
        }
    }

    return state;
}

```

```

        if(board[i][col+1] == plyrChar)
            board[i][col+1] == emptyChar;
    }
    else if((board[i][col] == shield1 || board[i][col] == shield2 ||
        board[i][col] == shield3 || board[i][col] == shield4)
        && enemy > 0)
    {
        if(board[i-1][col] == enemyChar)
            board[i][col] = emptyChar;
        else
        {
            board[i-1][col] = board[i][col];
            board[i][col] = emptyChar;
        }
    }
}

if(enemy > 0)
    state = true;

return state;
}

// this function determines how quick the enemies move as a function of how many
// enemies are still on the board, the high the number the slower they move
// since it they remove a that many frames essentially
void Board::enemyTimer()
{
    if(numEnemies <= 10)
        enemySpeed = 10;
    else if(numEnemies <= 30)
        enemySpeed = 12;
    else if(numEnemies <= 50)
        enemySpeed = 15;
}

// this function search the initial columns the enemies are in to set the X Y
// position of each of the enemy lasers X Y positions
void Board::setEnemyLasers()
{
    bool boolVar;

    // column 4 through 13
    for(int i = 4; i < 14; i++)
    {
        boolVar = true;
        for(int j = row-1; j > 0; --j)
        {
            if(board[j][i] == enemyChar && boolVar)
            {
                boolVar = false;
                switch(i)
                {
                    case 4:
                        lasX1 = j;

```

```

boolVar = true;
for(int j = row-1; j > 0; --j)
{
    if(board[j][i] == enemyChar && boolVar)
    {
        boolVar = false;
        switch(i)
        {
            case 4:
                lasX1 = j;
                lasY1 = i;
                break;
            case 5:
                lasX2 = j;
                lasY2 = i;
                break;
            case 6:
                lasX3 = j;
                lasY3 = i;
                break;
            case 7:
                lasX4 = j;
                lasY4 = i;
                break;
            case 8:
                lasX5 = j;
                lasY5 = i;
                break;
            case 9:
                lasX6 = j;
                lasY6 = i;
                break;
            case 10:
                lasX7 = j;
                lasY7 = i;
                break;
            case 11:
                lasX8 = j;
                lasY8 = i;
                break;
            case 12:
                lasX9 = j;
                lasY9 = i;
                break;
            case 13:
                lasX10 = j;
                lasY10 = i;
                break;
        }
    }
}
}
}

```



```

// this function randomizes which of the enemies shoot a laser, makes sure they
// cant be the same enemy shooting the laser, used random numbers
void Board::enemyLas()
{
    int randNum, x, y;

    if(numEnemyLas == 0 || numEnemyLas == 1)
        ++numEnemyLas;

    if((numEnemyLas == 1 || numEnemyLas == 2) && (eLasSpeed == 20))
    {
        randNum = rand() % 10 + 1;        // in range 1 to 10

        switch(randNum)
        {
            case 1:
                x = lasX1;
                y = lasY1;
                break;
            case 2:
                x = lasX2;
                y = lasY2;
                break;
            case 3:
                x = lasX3;
                y = lasY3;
                break;
            case 4:
                x = lasX4;
                y = lasY4;
                break;
            case 5:
                x = lasX5;
                y = lasY5;
                break;
            case 6:
                x = lasX6;
                y = lasY6;
                break;
            case 7:
                x = lasX7;
                y = lasY7;
                break;
            case 8:
                x = lasX8;
                y = lasY8;
                break;
            case 9:
                x = lasX9;
                y = lasY9;
                break;
            case 10:
                x = lasX10;
                y = lasY10;
        }
    }
}

```



```

        if(!eLas1State)
        {
            eLas1X = x;
            eLas1Y = y;
            eLas1State = true;
        }

        if(!eLas2State)
        {
            eLas2X = x;
            eLas2Y = y;
            if((eLas2X == eLas1X) && (eLas2Y == eLas1Y))
                enemyLas();
            eLas2State = true;
        }

        eLasSpeed = 0;
    }

    if(numEnemyLas == 1 || numEnemyLas == 2)
    {
        if(eLas1State && eLas1X > 0)
            moveEnemyLas1(eLas1X, eLas1Y);

        if(eLas2State && eLas2X > 0)
            moveEnemyLas2(eLas2X, eLas2Y);
    }
}

// this function moves one of the enemy lasers, makes sure it check for each of
// the possible characters it could encounter and adjusts like the player laser
// function did
void Board::moveEnemyLas1(int x, int y)
{
    if(x+1 > row-1)
    {
        board[x][y] = emptyChar;
        --numEnemyLas;
        eLas1X = eLas1Y = 0;
        eLas1State = false;
    }
    else if(board[x+1][y] == emptyChar)
    {
        if(board[x][y] == enemyChar)
            board[x+1][y] = lasChar;
        else
        {
            board[x][y] = emptyChar;
            board[x+1][y] = lasChar;
        }

        ++eLas1X;
    }
}

```

```

else if(board[x+1][y] == shield1 || board[x+1][y] == shield2 ||
        board[x+1][y] == shield3 || board[x+1][y] == shield4)
{
    if(board[x+1][y] == shield1)
    {
        switch(board[x+1][y])
        {
            case '4':
                board[x+1][y] = shield1 = sHlth3;
                board[x][y] = emptyChar;
                break;
            case '3':
                board[x+1][y] = shield1 = sHlth2;
                board[x][y] = emptyChar;
                break;
            case '2':
                board[x+1][y] = shield1 = sHlth1;
                board[x][y] = emptyChar;
                break;
            case '1':
                board[x+1][y] = shield1 = emptyChar;
                board[x][y] = emptyChar;
                break;
        }
    }
    else if(board[x+1][y] == shield2)
    {
        switch(board[x+1][y])
        {
            case '4':
                board[x+1][y] = shield2 = sHlth3;
                board[x][y] = emptyChar;
                break;
            case '3':
                board[x+1][y] = shield2 = sHlth2;
                board[x][y] = emptyChar;
                break;
            case '2':
                board[x+1][y] = shield2 = sHlth1;
                board[x][y] = emptyChar;
                break;
            case '1':
                board[x+1][y] = shield2 = emptyChar;
                board[x][y] = emptyChar;
                break;
        }
    }
    else if(board[x+1][y] == shield3)
    {
        switch(board[x+1][y])
        {

```

```

else if(board[x+1][y] == shield3)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield3 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield3 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield3 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield3 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
else if(board[x+1][y] == shield4)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield4 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield4 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield4 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield4 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
--numEnemyLas;
eLas1State = false;
eLas1X = eLas1Y = 0;
}
else if(board[x+1][y] == enemyChar)
{
    board[x][y] = emptyChar;
    board[x+1][y] = emptyChar;
    --numEnemyLas;
    eLas1State = false;
    eLas1X = eLas1Y = 0;
}

```

```

else if(board[x+1][y] == enemyChar)
{
    board[x][y] = emptyChar;
    board[x+1][y] = emptyChar;
    --numEnemyLas;
    eLas1State = false;
    eLas1X = eLas1Y = 0;
}
else if(board[x+1][y] == lasChar)
{
    board[x+1][y] = emptyChar;
    board[x][y] = emptyChar;
    --numEnemyLas;
    eLas1State = false;
    eLas1X = eLas1Y = 0;
}
else if(board[x+1][y] == plyrChar)
{
    ldrBrd.lostLife();
    --numEnemyLas;
    eLas1State = false;
    eLas1X = eLas1Y = 0;
    loseLife();
}
else
{
    --numEnemyLas;
    eLas1State = false;
    eLas1X = eLas1Y = 0;
}
}

```

```
// this function moves one of the enemy lasers, makes sure it check for each of
// the possible characters it could encounter and adjusts like the player laser
// function did
```

```
void Board::moveEnemyLas2(int x, int y)
```

```
{
    if(x+1 > row-1)
    {
        board[x][y] = emptyChar;
        --numEnemyLas;
        eLas2State = false;
        eLas2X = eLas2Y = 0;
    }
    else if(board[x+1][y] == emptyChar)
    {
        if(board[x][y] == enemyChar)
            board[x+1][y] = lasChar;
        else
        {
            board[x][y] = emptyChar;
            board[x+1][y] = lasChar;
        }

        ++eLas2X;
    }
    else if(board[x+1][y] == shield1 || board[x+1][y] == shield2 ||
            board[x+1][y] == shield3 || board[x+1][y] == shield4)
    {
        if(board[x+1][y] == shield1)
        {
            switch(board[x+1][y])
            {
                case '4':
                    board[x+1][y] = shield1 = sHlth3;
                    board[x][y] = emptyChar;
                    break;
                case '3':
                    board[x+1][y] = shield1 = sHlth2;
                    board[x][y] = emptyChar;
                    break;
                case '2':
                    board[x+1][y] = shield1 = sHlth1;
                    board[x][y] = emptyChar;
                    break;
                case '1':
                    board[x+1][y] = shield1 = emptyChar;
                    board[x][y] = emptyChar;
                    break;
            }
        }
        else if(board[x+1][y] == shield2)
        {
            switch(board[x+1][y])
            {

```

```

else if(board[x+1][y] == shield2)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield2 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield2 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield2 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield2 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
else if(board[x+1][y] == shield3)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield3 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield3 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield3 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield3 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
else if(board[x+1][y] == shield4)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield4 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield4 = sHlth2;
            board[x][y] = emptyChar;

```

```

        board[x][y] = emptyChar;
        break;
    }
}
--numEnemyLas;
eLas2State = false;
eLas2X = eLas2Y = 0;
}
else if(board[x+1][y] == enemyChar)
{
    board[x][y] = emptyChar;
    board[x+1][y] = emptyChar;
    --numEnemyLas;
    eLas2State = false;
    eLas2X = eLas2Y = 0;
}
else if(board[x+1][y] == lasChar)
{
    board[x+1][y] = emptyChar;
    board[x][y] = emptyChar;
    --numEnemyLas;
    eLas2State = false;
    eLas2X = eLas2Y = 0;
}
else if(board[x+1][y] == plyrChar)
{
    ldrBrd.lostLife();
    --numEnemyLas;
    eLas2State = false;
    eLas2X = eLas2Y = 0;
    loseLife();
}
else
{
    --numEnemyLas;
    eLas2State = false;
    eLas2X = eLas2Y = 0;
}
}
}

```



```

// this function happens when the player loses a life by being hit by an enemy
// laser, it resets the conditions so the board can be recreated again
void Board::loseLife()
{
    Sleep(500);

    system("clear");

    cout << "
    _
    | |           | | | | ( ) / _ | | " << endl;
    | |           | | | | | | | | | | " << endl;
    | | / _ W _ | | | | | | | / _ W | | " << endl;
    | | _ | ( ) W _ W _ | | | | | | _ / | | " << endl;
    | | _ _ / _ _ / | _ _ W _ _ / | | _ | ( ) " << endl;

    cout << "\nPress [ENTER] to continue...";

    cin.ignore();

    system("clear");

    // reset some conditions
    plyrX = plyrY = 0;
    moveR = 3;
    moveL = 0;
    enemySpeed = 15;
    speedTime = 0;
    downState = false;
    netLR = 0;
    numEnemyLas = 0;
    lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
    lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;
    lasX = lasY = 0;
    lasStateP = false;
    numEnemyLas = 0;
    eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
    eLas1State = eLas2State = false;
    eLasSpeed = 0;

    if(ldrBrd.getLives() == 0)
        loseGame();
    else
    {
        enemyTimer(); // determine enemy speed again
        createBoard(); // recreate the board
    }
}

```



```
// this function double checks if the enemies are all gone so the player can
// either win the game or if the player has any lives left so they can lose
// the game
void Board::checkEnemies()
{
    int enemiesLeft = 0; // how many enemies are left

    for(int i = 0 ; i < row ; i++)
    {
        for(int j = 0 ; j < col ; j++)
        {
            if(board[i][j] == enemyChar)
            {
                enemiesLeft++;
            }
        }
    }

    if(enemiesLeft == 0)
    {
        winGame(); // you win
    }

    if(ldrBrd.lives <= 0 )
    {
        loseGame(); // you lose
    }
}

// this function is when the player defeats all the enemies and wins the game,
// it gives the player the option to play again and enter a highscore if the
// player got higher than the current highscore
void Board::winGame()
{
    Sleep(500);
    int choice = 0;

    system("clear");

    cin.clear();

    cout << " _ _ _ _ _ " << endl;
    cout << "| | | ( _ ) | | " << endl;
    cout << "| | | | _ _ _ _ _ | | " << endl;
    cout << "| | / \ / | | ' _ W / _ W ' _ | | | | " << endl;
    cout << "W / \ / | | | | | _ / | _ | | " << endl;
    cout << " W / W / | _ | | _ | _ W _ | | ( _ ) " << endl;

    // ldrBrd.setHiScore();
}
```

```

// ldrBrd.setHiScore();

cout << " \nPress 1 to PLAY AGAIN\nPress 2 to EXIT\n(ENTER YOUR CHOICE): ";
cin >> choice;

cin.ignore();

switch(choice)
{
    case 1: break;
    case 2: exit(0); break;
    default: cout << "\n"; winGame(); break;
}

// reset conditions in the constructor
ldrBrd.lives = 3;
ldrBrd.setScoreOne(0);
plyrX = plyrY = 0;
shield1 = shield2 = shield3 = shield4 = '4';
numEnemies = 50;
moveR = 3;
moveL = 0;
enemySpeed = 15;
speedTime = 0;
downState = false;
netLR = 0;
lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;
lasX = lasY = 0;
lasStateP = false;
numEnemyLas = 0;
eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
eLas1State = eLas2State = false;
eLasSpeed = 0;
gameOver = false;

system("clear");

startMenu();
createBoard();
}

```

```

// this function is when the player loses all their lives before defeating all
// of the enemies, gives the player the option to play again
void Board::loseGame()
{
    Sleep(500);
    int choice = 0;

    system("clear");

    cin.clear();

    cout << " _____ " << endl;
    cout << " |   _  \ \           |   _  | " << endl;
    cout << " | |  \ \ /  _ _ _ _ _ | | | | " << endl;
    cout << " | |  _ / ' ' ' ' _ \ \ / _ \ \ / _ \ \ / _ \ ' ' _ | " << endl;
    cout << " | |  \ \ \ ( | | | | | | _ / \ \ \ \ / \ \ \ / _ \ | " << endl;
    cout << " \ \ _ \ \ _ , _ | | | | _ \ \ _ \ \ \ / \ \ _ \ _ | " << endl;

    // ldrBrd.setHiScore();

    cout << " \nPress 1 to PLAY AGAIN\nPress 2 to EXIT\n[ENTER YOUR CHOICE]: ";
    cin >> choice;

    cin.ignore();

    switch(choice)
    {
        case 1: break;
        case 2: exit(0); break;
        default: cout << "\n"; winGame(); break;
    }

    // reset conditions in the constructor
    ldrBrd.lives = 3;
    ldrBrd.setScoreOne(0);
    plyrX = plyrY = 0;
    shield1 = shield2 = shield3 = shield4 = '4';
    numEnemies = 50;
    moveR = 3;
    moveL = 0;
    enemySpeed = 15;
    speedTime = 0;
    downState = false;
    netLR = 0;
    lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
    lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;
    lasX = lasY = 0;
    lasStateP = false;
    numEnemyLas = 0;
    eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
    eLas1State = eLas2State = false;

```

```

switch(choice)
{
    case 1: break;
    case 2: exit(0); break;
    default: cout << "\n"; winGame(); break;
}

// reset conditions in the constructor
ldrBrd.lives = 3;
ldrBrd.setScoreOne(0);
plyrX = plyrY = 0;
shield1 = shield2 = shield3 = shield4 = '4';
numEnemies = 50;
moveR = 3;
moveL = 0;
enemySpeed = 15;
speedTime = 0;
downState = false;
netLR = 0;
lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;
lasX = lasY = 0;
lasStateP = false;
numEnemyLas = 0;
eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
eLas1State = eLas2State = false;
eLasSpeed = 0;
gameOver = false;

system("clear");

startMenu();
createBoard();
}

// this function stops the enemies from moving down on the row before the shields
bool Board::enemyStop(){
    bool state = false;
    int enemy = 0;
    for (int i = 0; i < col-1; i++){
        if (board[10][i] == enemyChar){
            enemy++;
        }
    }
    if (enemy > 0){
        state = true;
    }
    return state;
}

```

```

// this is a debug function to make sure there are no extra player characters
// that pop up on the board, or lasers
void Board::debug()
{
    for(int i = 1; i < 13; i++)
    {
        for(int j = 1 ; j < col ; j++)
        {
            if(board[i][j] == plyrChar)
                board[i][j] = emptyChar;
            if(lasStateP && board[i][j] == lasChar)
            {
                if(eLas1X != i && eLas1Y != j)
                {
                    if(eLas2X != i && eLas2Y != j)
                    {
                        if(lasX != i && lasY != j)
                        {
                            board[i][j] = emptyChar;
                            lasX = lasY = 0;
                            lasStateP = false;
                        }
                    }
                }
            }
        }
    }
}

```

main.cpp

```

// -----
// This is our main which controls the game.
// -----
int main()
{
    // create board object, initialise bool variable
    Board board;
    bool notLost = true;

    // start menu
    board.startMenu();

    // create the game board
    board.createBoard();

    // play Space Invaders; display board, get input, move enemies
    while(notLost)
    {
        system("clear");           // clear screen to display board again

        board.debug();              // debug player

        if(board.getLaserStateP())  // if player laser is on the board
            board.movFlyrLas();

        // if the enemies are to move again; left, right, or down
        if(board.getSpeedTimer() == board.getEnemySpeed())
        {
            if(board.getDownState()){
                if (board.enemyStop()){
                    board.downStop();
                }else {
                    board.moveEnemiesD();
                }
            }
            board.moveEnemiesLR();
        }

        board.enemyLas();           // move enemy lasers

        board.debug();              // debug player

        board.ldrBrd.getLeaderboard(); // leaderboard top
        board.dispBrd();             // game board 2D array
        board.ldrBrd.getBtmBoard();  // leaderboard bottom

        board.input();              // player input

        board.speedTimer();          // increment speed timer
        board.eLasSpeedUp();         // increment enemy laser speed timer
        board.enemyTimer();          // increment enemy movement timer

        board.checkEnemies();        // checks number of enemies on board, lives
    }

    return 0;
}

```

# Code in text:

```
// header.h ////////////////////////////////////////

// -----
// This is the header file for the project with all the libraries.
// -----
#include <iostream>    // Input - Output Library
#include <iomanip>      // Various formatting
#include <cstdlib>     // srand to set the seed
#include <ctime>       // time for rand
#include <fstream>     // needed for file operations
#include <windows.h>   // for getasynkeystate function, Sleep();

using namespace std;

// record.h ////////////////////////////////////////

// -----
// This header file holds a structure that is used for the high score file.
// -----
const int NAME_SIZE = 11; // maximum size hi-score name can be (-1 for '\0')

struct Record{
    char name[NAME_SIZE];
    int score;
};

// leaderboard.h ////////////////////////////////////////

// -----
// This class holds all of the information for the leaderboard information that
// goes on the top and bottom of the game board. It is able to add points
// when the player hits an enemy with a laser and display a current high score
// from a file.
// -----
const int TOP5 = 5;

class Leaderboard
{
private:
    Record recorder;           //Holds hi-score information
    int size;                 //Size of all hi-score records from file.
    int index;                // variable to hold an index
    int scoreOne;             //Player One's score.
    int hiScore;              //High Score.
    int credit;               //Credits player has.
public:
```

```

Leaderboard();           //Default constructor.
void addScore(int);       //Add score for kills.
void setHiScore();       //Set Hi-Score.
int getScore(){ return scoreOne; } //Display Player One's score.
int getHiScore(){ return hiScore; } //Display Hi-Score.
void getLeaderboard();   //Display scores neatly.
void getBtmBoard();      //Display lives & credits
void displayLives();     //Determine how many lives to display.
int lives;               //The number of lives the player has.
bool hiScores();         // high scores function
void displayHiscores();  // display high scores function
void registration();     //Get name from user if hi-score reached.
void setScoreOne(int);   //For resetting the game
void lostLife() { --lives; } //Lose a life
int getLives() { return lives; } //Get current number of lives
};

Leaderboard::Leaderboard(){           //Default constructor.
    index=0;
    scoreOne=0;
    hiScore=0;
    lives=3;
    credit=0;
}

void Leaderboard::setScoreOne(int s) { scoreOne = s; } //lets the board class reset the score

void Leaderboard::addScore(int score){ //Accepts score for the kill and
    scoreOne+=score;                  //Adds score to scoreOne.
}

void Leaderboard::setHiScore(){
    if(hiScore<=scoreOne)
        hiScore=scoreOne;
    if(hiScores()){
        registration();
    }
}

void Leaderboard::getLeaderboard(){ //Display the top UI.
    cout<<"SCORE<1>"<<"\tHI-SCORE\n ";
    "<<setw(4)<<setfill('0')<<hiScore<<endl<<endl;   cout<<setw(4)<<setfill('0')<<scoreOne<<"\t\t"
}

void Leaderboard::getBtmBoard(){ //Display lives & credits below
    cout<<endl<<lives;           //game board.
    displayLives(); //Call separate function to display lives
    cout<<"\t\tCREDIT "<<credit<<endl;
}

```



```

void Leaderboard::displayLives(){
    switch(lives){
        //Determine how many lives to
        case 3: cout<<" PP";           //display based on lives left
            break;                       //in Space Invaders Theme.
        case 2: cout<<" P";
            break;
        case 1: cout<<" ";
            break;
    }
}

bool Leaderboard::hiScores(){
    fstream hiscoreFile;
    hiscoreFile.open("hiscores.dat", ios::in | ios::binary); //Check if file exists.
    if(hiscoreFile.fail()){
        hiscoreFile.open("hiscores.dat", ios::out|ios::binary); //create a new file.
        cout<<"NO HI-SCORES RECORDED YET."<<endl;
        cout<<"NEW HI-SCORE!"<<endl;
        hiscoreFile.close();
        return true;
    }
    else{
        long numBytes;
        hiscoreFile.seekg(0L, ios::end);
        numBytes=hiscoreFile.tellg();
        hiscoreFile.seekg(0L, ios::beg);
        hiscoreFile.close();
        displayHiscores();
        return true;
    }
}

void Leaderboard::displayHiscores(){
    fstream hiscoreFile;
    Record temp[size];
    hiscoreFile.open("hiscores.dat", ios::in|ios::binary);
    hiscoreFile.read(reinterpret_cast<char *>(&temp), sizeof(temp));
    for(int count=0;count<size;count++){
        cout<<"*****"<<endl;
        cout<<setw(12)<<temp[count].name<<setw(6)<<temp[count].score<<endl;
    }
    cout<<"*****"<<endl;
}

void Leaderboard::registration(){
    fstream hiscoresFile;
    char name[NAME_SIZE];
    size++;
    getLeaderboard();
    //reevaluateScores();
}

```

```

cout<<"\t\tHI-SCORER REGISTRATION\n";

cout<<"ENTER NAME (10 letters max): ";
cin.getline(recorder.name, NAME_SIZE);
recorder.score=hiScore;
hiscoresFile.open("hiscores.dat", ios::out | ios::app | ios::binary);
hiscoresFile.write(reinterpret_cast<char *>(&recorder),sizeof(recorder));
}

// board.h ////////////////////////////////////////

// -----
// This class holds everything that goes on the game board itself like the
// player, enemies, lasers, shields, and boundaries. The board is able to be
// printed based on how fast the computer running it can handle.
// -----
class Board
{
private:
    // board data
    int row;           // row size of 2D array
    int col;           // col size of 2D array
    char **board;      // dynamic 2D array for game board
    bool gameOver;     // bool to determine if the game is over
    char emptyChar;    // char for empty spaces on the game board
    char boundaryChar; // char for boundaries on the sides of the board

    // player data
    char plyrChar;     // char for the player on the board
    int plyrX, plyrY;  // x and y of the player in the 2D array

    // enemy data
    char enemyChar;    // char for enemy on the board
    char destroyedEnemyChar; // char for when the enemy is destroyed
    int enemyPoints;   // point value per enemy
    int numEnemies;    // total number of enemies on board
    int moveR;         // number of times the enemies need to move right
    int moveL;         // number of times the enemies need to move left
    int enemySpeed;    // how quickly the enemies move, higher = slower
    int speedTime;     // variable to control enemy speed, increments
    bool downState;    // bool if the enemies need to move down or not
    // X Y position for the two enemy lasers
    int lasX1, lasX2, lasX3, lasX4, lasX5, lasX6, lasX7, lasX8, lasX9, lasX10;
    int lasY1, lasY2, lasY3, lasY4, lasY5, lasY6, lasY7, lasY8, lasY9, lasY10;
    int netLR;         // the net enemy movement, used for enemy lasers
    int numEnemyLas;   // number of enemy lasers on board at once, max 2
    int eLas1X, eLas1Y; // enemy las 1's X Y position in 2D array
    int eLas2X, eLas2Y; // enemy las 2's X Y position in 2D array
    bool eLas1State, eLas2State; // bool for if las 1 or 2 is on the board
    int eLasSpeed;     // speed for the lasers

```

```

// shield data
char shield1;           // char for the first shield on the board
char shield2;           // char for the second shield on the board
char shield3;           // char for the third shield on the board
char shield4;           // char for the fourth shield on the board
char sHlth4;            // shield health of '4'
char sHlth3;            // shield health of '3'
char sHlth2;            // shield health of '2'
char sHlth1;            // shield health of '1'

// laserData
char lasChar;           // char for any laser in the 2D array
int lasX, lasY;         // x and y of the player's laser, vertically bound
bool lasStateP;         // state of laser, if its on board or not

// loseGame
int choice;             // variable to hold choice

public:
    Board();             // constructor
    ~Board();            // destructor

// create object for the Leaderboard class in the board class
Leaderboard ldrBrd;

// class functions
void startMenu();        // menu to introduce game
void createBoard();      // starting board conditions
void dispBrd();          // output the board itself
bool getGameState() { return gameOver; } // return if game is over or not

// player functions
void input();            // keyboard input of player
bool getLaserStateP() { return lasStateP; } // if player laser is on the board
void plyrLas();          // set X Y of laser from the player X Y
void movPlyrLas();       // move the player laser

// enemy functions
void moveEnemiesLR();    // move enemies Left or Right
void moveEnemiesD();     // move enemies Down
void speedTimer() { speedTime++; } // increment speed timer
int getSpeedTimer() { return speedTime; } // get speed timer for main
int getEnemySpeed() { return enemySpeed; } // get enemy speed for main
bool checkRow(int, char); // check if row has enemies in it
bool checkCol(int);       // check if column has enemies
void checkSide(char);     // check if sides have enemies
bool getDownState() { return downState; } // get down state for main
void setEnemyLasers();    // set up all the enemy lasers X Y
void enemyTimer();        // how fast enemies are based on how many are left

```

```

void enemyLas();           // randomize which enemies shoot a laser
void moveEnemyLas1(int, int); // move enemy laser 1
void moveEnemyLas2(int, int); // move enemy laser
void eLasSpeedUp() { eLasSpeed++; } // keeps track of enemy laser speed
void checkEnemies();       // double check how many enemies left
bool enemyStop();          // stop enemies on row before shields
void downStop() { downState = false; } // stop enemies from moving down

// game functions
void loseLife();           // lose a life
void winGame();           // you win the game
void loseGame();          // you lose the game

// debug function
void debug();             // housekeeping for the player icon with row movement
};

// constructor to set size of game board, initialize dynamic 2D array
Board::Board()
{
    // board variables
    row = 14;              // height of board
    col = 18;              // width of board

    // initialize dynamic 2D array
    board = new char*[row];
    for(int i=0; i<row; i++)
        board[i] = new char[col];

    // player variables
    plyrX = plyrY = 0;

    // character variables for the board
    emptyChar = '-';
    boundaryChar = '#';
    plyrChar = 'P';
    lasChar = '|';
    enemyChar = 'E';
    destroyedEnemyChar = 'x';

    // shield variables
    shield1 = shield2 = shield3 = shield4 = '4'; // starting health for shield
    sHlth4 = '4';
    sHlth3 = '3';
    sHlth2 = '2';
    sHlth1 = '1';

    // enemy variables
    numEnemies = 50;
    enemyPoints = 100;

```

```

moveR = 3;
moveL = 0;
enemySpeed = 15;
speedTime = 0;
downState = false;
netLR = 0;
lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;

// laser variables
lasX = lasY = 0;
lasStateP = false;
numEnemyLas = 0;
eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
eLas1State = eLas2State = false;
eLasSpeed = 0;

// game variables
gameOver = false;

// set the random number seed
srand(static_cast<unsigned int>(time(0)));
}

// destructor to delete dynamic memory from 2D array
Board::~Board()
{
    for(int i=0; i < row; i++)
        delete [] board[i];
    delete [] board;
}

// this function outputs a start menu and ascii graphic for the game
// it includes instructions and points along with the initial leaderboard
// and highscore
void Board::startMenu()
{
    ldrBrd.getLeaderboard();

    cout << "                                     \n";
    cout << " /_____|_/_/_/_/_/_/_|_|_|_/_/_/_/_/_/_\n";
    cout << " \|__ \|'_ \|/_/_/_/_ \| ||' \| \| \| '_ \|/_/_ \|'_ \|/_ \|_\n";
    cout << "   ) |_) |( _) |(_ _/_||| \| V / (_) |(_) |_/_ \|_\n";
    cout << " |___/| ._/ \|_, \|__ \|_| |_| \|/_/__, \|_, \|_ \|_| |_|/_ \|_\n";
    cout << "     |_\n\n";

    cout << "'A' to move LEFT\n";
    cout << "'D' to move RIGHT\n";
    cout << "'SPACEBAR' to shoot laser\n\n";

```

```

cout << " * SCORE ADVANCE TABLE *\n";
cout << "    " << enemyChar << " = " << enemyPoints << " POINTS\n\n";
cout << "Press [ENTER] to continue...";

cin.ignore();
}

// this functions sets up the initial conditions of the game board with:
// player, side barriers, enemies, blockades, empty space
void Board::createBoard()
{
    int e = numEnemies;           // to dynamically place in enemies

    for(int i=0; i < row; i++){
        for(int j=0; j < col; j++){
            if(i == 13 && j == 8)
            {
                board[i][j] = plyrChar;
                plyrX = i;
                plyrY = j;
                lasX = plyrX;
                lasY = plyrY;
            }
            else if(i == 11 && j == 4)
            {
                board[i][j] = shield1;
            }
            else if(i == 11 && j == 7)
            {
                board[i][j] = shield2;
            }
            else if(i == 11 && j == 10)
            {
                board[i][j] = shield3;
            }
            else if(i == 11 && j == 13)
            {
                board[i][j] = shield4;
            }
            else if (i > 0 && i < 6 && j > 3 && j < 14 && e > 0)
            {
                board[i][j] = enemyChar;
                --e;
            }
            else if(j==0 || j==col-1)
            {
                board[i][j] = boundaryChar;
            }
            else
            {

```

```

        board[i][j] = emptyChar;
    }
}
}
setEnemyLasers();           // initial enemy laser positions
}

// this function simply outputs the current state of the board, or the 2D array
// at the end it clears the screen just to display itself again
void Board::dispBrd()
{
    for(int i=0; i < row; i++){
        for(int j=0; j < col; j++){
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
}

// this function looks for input from the player to move Left and Right, and
// shoot lasers; it has boundary check and laser delay
void Board::input()
{
    if(board[plyrX][plyrY] != plyrChar)
        board[plyrX][plyrY] = plyrChar;

    if(GetAsyncKeyState(0x41) && (board[plyrX][plyrY-1] != '#')){ // 'a'
        board[plyrX][plyrY] = emptyChar;
        board[plyrX][plyrY-1] = plyrChar;
        plyrY--;
    }
    else if(GetAsyncKeyState(0x44) && (board[plyrX][plyrY+1] != '#')){ // 'd'
        board[plyrX][plyrY] = emptyChar;
        board[plyrX][plyrY+1] = plyrChar;
        plyrY++;
    }
    else if(GetAsyncKeyState(0x20) && lasStateP == false){ // 'SPACEBAR'
        plyrLas();
    }
}

// this function sets the X Y position for the player laser to the player's
// current position
void Board::plyrLas()
{
    lasStateP = true;

    if(board[plyrX][plyrY] == plyrChar)
    {
        lasX = plyrX;
    }
}

```

```

        lasY = plyrY;
    }
}

// this function moves the player laser, vertically bound
// it considers every type of character that could be above it
void Board::movPlyrLas()
{
    // if outside of the board, stop laser, set empty char
    if(lasX-1 < 0)
    {
        board[lasX][lasY] = emptyChar;
        lasX = lasY = 0;
        lasStateP = false;
    }
    // if empty, move laser up one, put empty space where it was
    else if(board[lasX-1][lasY] == emptyChar)
    {
        if(board[lasX][lasY] == plyrChar)
            board[lasX-1][lasY] = lasChar;
        else
        {
            board[lasX][lasY] = emptyChar;
            board[lasX-1][lasY] = lasChar;
        }
    }

    lasX -= 1;
}
// if a shield, decrement the enemy shield, keep track of each shield
// position X Y
else if(board[lasX-1][lasY] == shield1 || board[lasX-1][lasY] == shield2 ||
        board[lasX-1][lasY] == shield3 || board[lasX-1][lasY] == shield4)
{
    if(board[lasX-1][lasY] == shield1)
    {
        switch(board[lasX-1][lasY])
        {
            case '4':
                board[lasX-1][lasY] = shield1 = sHlth3;
                board[lasX][lasY] = emptyChar;
                break;
            case '3':
                board[lasX-1][lasY] = shield1 = sHlth2;
                board[lasX][lasY] = emptyChar;
                break;
            case '2':
                board[lasX-1][lasY] = shield1 = sHlth1;
                board[lasX][lasY] = emptyChar;
                break;
            case '1':

```



```

        board[lasX-1][lasY] = shield1 = emptyChar;
        board[lasX][lasY] = emptyChar;
        break;
    }
}
else if(board[lasX-1][lasY] == shield2)
{
    switch(board[lasX-1][lasY])
    {
        case '4':
            board[lasX-1][lasY] = shield2 = sHlth3;
            board[lasX][lasY] = emptyChar;
            break;
        case '3':
            board[lasX-1][lasY] = shield2 = sHlth2;
            board[lasX][lasY] = emptyChar;
            break;
        case '2':
            board[lasX-1][lasY] = shield2 = sHlth1;
            board[lasX][lasY] = emptyChar;
            break;
        case '1':
            board[lasX-1][lasY] = shield2 = emptyChar;
            board[lasX][lasY] = emptyChar;
            break;
    }
}
else if(board[lasX-1][lasY] == shield3)
{
    switch(board[lasX-1][lasY])
    {
        case '4':
            board[lasX-1][lasY] = shield3 = sHlth3;
            board[lasX][lasY] = emptyChar;
            break;
        case '3':
            board[lasX-1][lasY] = shield3 = sHlth2;
            board[lasX][lasY] = emptyChar;
            break;
        case '2':
            board[lasX-1][lasY] = shield3 = sHlth1;
            board[lasX][lasY] = emptyChar;
            break;
        case '1':
            board[lasX-1][lasY] = shield3 = emptyChar;
            board[lasX][lasY] = emptyChar;
            break;
    }
}
else if(board[lasX-1][lasY] == shield4)

```

```

{
    switch(board[lasX-1][lasY])
    {
        case '4':
            board[lasX-1][lasY] = shield4 = sHlth3;
            board[lasX][lasY] = emptyChar;
            break;
        case '3':
            board[lasX-1][lasY] = shield4 = sHlth2;
            board[lasX][lasY] = emptyChar;
            break;
        case '2':
            board[lasX-1][lasY] = shield4 = sHlth1;
            board[lasX][lasY] = emptyChar;
            break;
        case '1':
            board[lasX-1][lasY] = shield4 = emptyChar;
            board[lasX][lasY] = emptyChar;
            break;
    }
}
lasStateP = false;
lasX = lasY = 0;
}
// if its an enemy, replace las and enemy with empty char, add score, stop
// laser on board, decrement number of enemies on board, reset enemy
// laser position to enemy above it using net movement
else if(board[lasX-1][lasY] == enemyChar)
{
    board[lasX-1][lasY] = destroyedEnemyChar;
    board[lasX][lasY] = emptyChar;
    board[lasX-1][lasY] = emptyChar;

    ldrBrd.addScore(enemyPoints);
    lasX = lasY = 0;
    lasStateP = false;
    --numEnemies;

    switch(lasY - 3 + (-1*netLR))
    {
        case 1:
            --lasX1;
            break;
        case 2:
            --lasX2;
            break;
        case 3:
            --lasX3;
            break;
        case 4:

```

```

        --lasX4;
        break;
    case 5:
        --lasX5;
        break;
    case 6:
        --lasX6;
        break;
    case 7:
        --lasX7;
        break;
    case 8:
        --lasX8;
        break;
    case 9:
        --lasX9;
        break;
    case 10:
        --lasX10;
        break;
    }
}
// if its a laser they cancel out, both empty spots
else if(board[lasX-1][lasY] == lasChar)
{
    if(lasX-1 == eLas1X && lasY == eLas1Y)
    {
        board[lasX-1][lasY] = emptyChar;
        eLas1X = eLas1Y = 0;
        eLas1State = false;
    }
    else if(lasX-1 == eLas2X && lasY == eLas2Y)
    {
        board[lasX-1][lasY] = emptyChar;
        eLas2X = eLas2Y = 0;
        eLas2State = false;
    }
    board[lasX-1][lasY] = emptyChar;
    board[lasX][lasY] = emptyChar;
    lasX = lasY = 0;
    lasStateP = false;
}
// otherwise stop the laser
else
{
    lasX = lasY = 0;
    lasStateP = false;
}
}

```

```

// this function moves the group of enemies left or right one spot at a time
// using the timer, it check if each row even has an enemy otherwise it wont
// move the row, it also uses a temp variables to hold the end array spot
// so it can rotate the whole role, it also adjusts the enemy laser positions,
// it also check the side to see if the enemies can move left or right more
void Board::moveEnemiesLR()

```

```

{
    if(moveR > 0)
    {
        for(int i = 0; i < row-3; i++)
        {
            if(checkRow(i, 'R'))
            {
                int temp = board[i][col-2];
                for(int j = col-2; j > 0; --j)
                {
                    board[i][j] = board [i][j-1];
                }
                board [i][1] = temp;
            }
        }
    }
}

```

```

++lasY10, ++lasY1, ++lasY2, ++lasY3, ++lasY4, ++lasY5, ++lasY6, ++lasY7, ++lasY8, ++lasY9,

```

```

    if(moveR == 1)
        checkSide('R');

    if(moveR == 1)
    {
        moveR = 0;
        moveL = 5;
        downState = true;
    }
    else if(moveR > 1)
        moveR--;

    netLR++;
}
else if(moveL > 0)
{
    for(int i = 0; i < row-3; i++)
    {
        if(checkRow(i, 'L'))
        {
            int temp = board[i][1];
            for(int j = 1; j < col-2; j++)
            {
                board[i][j] = board [i][j+1];
            }
            board [i][col-2] = temp;
        }
    }
}

```

```

    }
}

--lasY1, --lasY2, --lasY3, --lasY4, --lasY5, --lasY6, --lasY7, --lasY8, --lasY9, --lasY10;

if(moveL == 1)
    checkSide('L');

if(moveL == 1)
{
    moveR = 5;
    moveL = 0;
    downState = true;
}
else if(moveL > 1)
    moveL--;

    netLR--;
}
speedTime = 0;
}

```

// this function check the row to see if there is an enemy or other thing in the  
// row so it can move it, it corrects if there is a laser character

```

bool Board::checkRow(int row, char direction)
{
    bool state = false;
    int enemy = 0;

    if(direction == 'R')
    {
        for(int i = 1; i < col-2; i++)
        {
            if(board[row][i] == enemyChar)
                enemy++;
            else if((board[row][i] == shield1 || board[row][i] == shield2 ||
                board[row][i] == shield3 || board[row][i] == shield4) &&
                enemy > 0)
            {
                if(board[row][i-1] == enemyChar)
                    board[row][i] = emptyChar;
                else
                {
                    board[row][i-1] = board[row][i];
                    board[row][i] = emptyChar;
                }
            }
        }
    }
    if(enemy > 0)
    {

```

```

for(int i = 1; i < col-2; i++)
{
    if(board[row][i] == lasChar)
    {
        if(board[row][i-1] == enemyChar)
        {
            board[row][i] = emptyChar;
            lasStateP = false;
        }
        else if(board[row][i-1] == emptyChar)
        {
            board[row][i-1] == lasChar;
            board[row][i] == enemyChar;
        }
    }
}
}
}
else if (direction == 'L')
{
    for(int i = col-2; i > 1; --i)
    {
        if(board[row][i] == enemyChar)
            enemy++;
        else if((board[row][i] == shield1 || board[row][i] == shield2 ||
            board[row][i] == shield3 || board[row][i] == shield4) &&
            enemy > 0)
        {
            if(board[row][i+1] == enemyChar)
                board[row][i] = emptyChar;
            else
            {
                board[row][i+1] = board[row][i];
                board[row][i] = emptyChar;
            }
            enemy++;
        }
    }
}
if(enemy > 0)
{
    for(int i = 1; i < col-2; i++)
    {
        if(board[row][i] == lasChar)
        {
            if(board[row][i+1] == enemyChar)
            {
                board[row][i] = emptyChar;
                lasStateP = false;
            }
            else if(board[row][i+1] == emptyChar)

```

```

        {
            board[row][i+1] == lasChar;
            board[row][i] == enemyChar;
        }
    }
}

if(enemy > 0)
    state = true;

return state;
}

// this board check the sides next to the boundaries to see if there are enemies
// so the moveLR function knows if it can move the enemies another spot left
// or right
void Board::checkSide(char direction)
{
    int enemy = 0;

    if(direction == 'R')
    {
        for(int i = 0; i < row; i++)
        {
            if(board[i][col-2] == enemyChar)
            {
                enemy++;
            }
        }
        if(enemy == 0)
            moveR++;
    }
    else if(direction == 'L')
    {
        for(int i = 0; i < row; i++)
        {
            if(board[i][1] == enemyChar)
            {
                enemy++;
            }
        }
        if(enemy == 0)
            moveL++;
    }
}

// this function moves the enemies down, adjusts the enemy laser position,
// and checks each column to adjust everything

```

```
void Board::moveEnemiesD()
```

```
{
    int temp;

    for(int i = 1; i < col-1; i++)
    {
        if(checkCol(i))
        {
            temp = board[row-1][i];
            for(int j = row-1; j > 0; --j)
            {
                board[j][i] = board[j-1][i];
            }
            board [0][i] = temp;
        }
    }
}
```

```
++lasX10, ++lasX1, ++lasX2, ++lasX3, ++lasX4, ++lasX5, ++lasX6, ++lasX7, ++lasX8, ++lasX9,
```

```
    downState = false;
}
```

```
// this function checks the columns that enemies are in so they can move down,
// it accounts for player, laser, and shield characters so they dont get
// displaced
```

```
bool Board::checkCol(int col)
```

```
{
    bool state = false;
    int enemy = 0;

    for(int i = 0; i < row; i++)
    {
        if(board[i][col] == enemyChar)
        {
            enemy++;
        }
        else if(board[i][col] == plyrChar)
        {
            board[i][col] = emptyChar;
            board[i-1][col] = plyrChar;

            if(board[i-1][col-1] == plyrChar)
                board[i-1][col-1] = emptyChar;
            if(board[i][col-1] == plyrChar)
                board[i][col-1] = emptyChar;
            if(board[i-1][col+1] == plyrChar)
                board[i-1][col+1] = emptyChar;
            if(board[i][col+1] == plyrChar)
                board[i][col+1] = emptyChar;
        }
    }
}
```



```

        else if((board[i][col] == shield1 || board[i][col] == shield2 ||
            board[i][col] == shield3 || board[i][col] == shield4)
            && enemy > 0)
        {
            if(board[i-1][col] == enemyChar)
                board[i][col] = emptyChar;
            else
            {
                board[i-1][col] = board[i][col];
                board[i][col] = emptyChar;
            }
        }
    }
}

if(enemy > 0)
    state = true;

return state;
}

// this function determines how quick the enemies move as a function of how many
// enemies are still on the board, the high the number the slower they move
// since it they remove a that many frames essentially
void Board::enemyTimer()
{
    if(numEnemies <= 10)
        enemySpeed = 10;
    else if(numEnemies <= 30)
        enemySpeed = 12;
    else if(numEnemies <= 50)
        enemySpeed = 15;
}

// this function search the initial columns the enemies are in to set the X Y
// position of each of the enemy lasers X Y positions
void Board::setEnemyLasers()
{
    bool boolVar;

    // column 4 through 13
    for(int i = 4; i < 14; i++)
    {
        boolVar = true;
        for(int j = row-1; j > 0; --j)
        {
            if(board[j][i] == enemyChar && boolVar)
            {
                boolVar = false;
                switch(i)
                {

```

```

        case 4:
            lasX1 = j;
            lasY1 = i;
            break;
        case 5:
            lasX2 = j;
            lasY2 = i;
            break;
        case 6:
            lasX3 = j;
            lasY3 = i;
            break;
        case 7:
            lasX4 = j;
            lasY4 = i;
            break;
        case 8:
            lasX5 = j;
            lasY5 = i;
            break;
        case 9:
            lasX6 = j;
            lasY6 = i;
            break;
        case 10:
            lasX7 = j;
            lasY7 = i;
            break;
        case 11:
            lasX8 = j;
            lasY8 = i;
            break;
        case 12:
            lasX9 = j;
            lasY9 = i;
            break;
        case 13:
            lasX10 = j;
            lasY10 = i;
            break;
    }
}
}
}
}
}

```

```

// this function randomizes which of the enemies shoot a laser, makes sure they
// cant be the same enemy shooting the laser, used random numbers
void Board::enemyLas()
{

```

```

int randNum, x, y;

if(numEnemyLas == 0 || numEnemyLas == 1)
    ++numEnemyLas;

if((numEnemyLas == 1 || numEnemyLas == 2) && ((eLasSpeed == 20)))
{
    randNum = rand() % 10 + 1;    // in range 1 to 10

    switch(randNum)
    {
        case 1:
            x = lasX1;
            y = lasY1;
            break;
        case 2:
            x = lasX2;
            y = lasY2;
            break;
        case 3:
            x = lasX3;
            y = lasY3;
            break;
        case 4:
            x = lasX4;
            y = lasY4;
            break;
        case 5:
            x = lasX5;
            y = lasY5;
            break;
        case 6:
            x = lasX6;
            y = lasY6;
            break;
        case 7:
            x = lasX7;
            y = lasY7;
            break;
        case 8:
            x = lasX8;
            y = lasY8;
            break;
        case 9:
            x = lasX9;
            y = lasY9;
            break;
        case 10:
            x = lasX10;
            y = lasY10;
    }
}

```

```

        break;
    }

    if(!eLas1State)
    {
        eLas1X = x;
        eLas1Y = y;
        eLas1State = true;
    }

    if(!eLas2State)
    {
        eLas2X = x;
        eLas2Y = y;
        if((eLas2X == eLas1X) && (eLas2Y == eLas1Y))
            enemyLas();
        eLas2State = true;
    }

    eLasSpeed = 0;
}

if(numEnemyLas == 1 || numEnemyLas == 2)
{
    if(eLas1State && eLas1X > 0)
        moveEnemyLas1(eLas1X, eLas1Y);

    if(eLas2State && eLas2X > 0)
        moveEnemyLas2(eLas2X, eLas2Y);
}
}

// this function moves one of the enemy lasers, makes sure it check for each of
// the possible characters it could encounter and adjusts like the player laser
// function did
void Board::moveEnemyLas1(int x, int y)
{
    if(x+1 > row-1)
    {
        board[x][y] = emptyChar;
        --numEnemyLas;
        eLas1X = eLas1Y = 0;
        eLas1State = false;
    }
    else if(board[x+1][y] == emptyChar)
    {
        if(board[x][y] == enemyChar)
            board[x+1][y] = lasChar;
        else
        {

```

```

        board[x][y] = emptyChar;
        board[x+1][y] = lasChar;
    }

    ++eLas1X;
}
else if(board[x+1][y] == shield1 || board[x+1][y] == shield2 ||
        board[x+1][y] == shield3 || board[x+1][y] == shield4)
{
    if(board[x+1][y] == shield1)
    {
        switch(board[x+1][y])
        {
            case '4':
                board[x+1][y] = shield1 = sHlth3;
                board[x][y] = emptyChar;
                break;
            case '3':
                board[x+1][y] = shield1 = sHlth2;
                board[x][y] = emptyChar;
                break;
            case '2':
                board[x+1][y] = shield1 = sHlth1;
                board[x][y] = emptyChar;
                break;
            case '1':
                board[x+1][y] = shield1 = emptyChar;
                board[x][y] = emptyChar;
                break;
        }
    }
}
else if(board[x+1][y] == shield2)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield2 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield2 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield2 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield2 = emptyChar;
            board[x][y] = emptyChar;
    }
}

```

```

        break;
    }
}
else if(board[x+1][y] == shield3)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield3 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield3 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield3 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield3 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
else if(board[x+1][y] == shield4)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield4 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield4 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield4 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield4 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
--numEnemyLas;
eLas1State = false;
eLas1X = eLas1Y = 0;

```

```

    }
    else if(board[x+1][y] == enemyChar)
    {
        board[x][y] = emptyChar;
        board[x+1][y] = emptyChar;
        --numEnemyLas;
        eLas1State = false;
        eLas1X = eLas1Y = 0;
    }
    else if(board[x+1][y] == lasChar)
    {
        board[x+1][y] = emptyChar;
        board[x][y] = emptyChar;
        --numEnemyLas;
        eLas1State = false;
        eLas1X = eLas1Y = 0;
    }
    else if(board[x+1][y] == plyrChar)
    {
        ldrBrd.lostLife();
        --numEnemyLas;
        eLas1State = false;
        eLas1X = eLas1Y = 0;
        loseLife();
    }
    else
    {
        --numEnemyLas;
        eLas1State = false;
        eLas1X = eLas1Y = 0;
    }
}

```

// this function moves one of the enemy lasers, makes sure it check for each of  
// the possible characters it could encounter and adjusts like the player laser  
// function did

```

void Board::moveEnemyLas2(int x, int y)
{
    if(x+1 > row-1)
    {
        board[x][y] = emptyChar;
        --numEnemyLas;
        eLas2State = false;
        eLas2X = eLas2Y = 0;
    }
    else if(board[x+1][y] == emptyChar)
    {
        if(board[x][y] == enemyChar)
            board[x+1][y] = lasChar;
        else

```

```

    {
        board[x][y] = emptyChar;
        board[x+1][y] = lasChar;
    }

    ++eLas2X;
}
else if(board[x+1][y] == shield1 || board[x+1][y] == shield2 ||
        board[x+1][y] == shield3 || board[x+1][y] == shield4)
{
    if(board[x+1][y] == shield1)
    {
        switch(board[x+1][y])
        {
            case '4':
                board[x+1][y] = shield1 = sHlth3;
                board[x][y] = emptyChar;
                break;
            case '3':
                board[x+1][y] = shield1 = sHlth2;
                board[x][y] = emptyChar;
                break;
            case '2':
                board[x+1][y] = shield1 = sHlth1;
                board[x][y] = emptyChar;
                break;
            case '1':
                board[x+1][y] = shield1 = emptyChar;
                board[x][y] = emptyChar;
                break;
        }
    }
    else if(board[x+1][y] == shield2)
    {
        switch(board[x+1][y])
        {
            case '4':
                board[x+1][y] = shield2 = sHlth3;
                board[x][y] = emptyChar;
                break;
            case '3':
                board[x+1][y] = shield2 = sHlth2;
                board[x][y] = emptyChar;
                break;
            case '2':
                board[x+1][y] = shield2 = sHlth1;
                board[x][y] = emptyChar;
                break;
            case '1':
                board[x+1][y] = shield2 = emptyChar;

```



```

        board[x][y] = emptyChar;
        break;
    }
}
else if(board[x+1][y] == shield3)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield3 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield3 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield3 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield3 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
else if(board[x+1][y] == shield4)
{
    switch(board[x+1][y])
    {
        case '4':
            board[x+1][y] = shield4 = sHlth3;
            board[x][y] = emptyChar;
            break;
        case '3':
            board[x+1][y] = shield4 = sHlth2;
            board[x][y] = emptyChar;
            break;
        case '2':
            board[x+1][y] = shield4 = sHlth1;
            board[x][y] = emptyChar;
            break;
        case '1':
            board[x+1][y] = shield4 = emptyChar;
            board[x][y] = emptyChar;
            break;
    }
}
--numEnemyLas;
eLas2State = false;

```

```

        eLas2X = eLas2Y = 0;
    }
    else if(board[x+1][y] == enemyChar)
    {
        board[x][y] = emptyChar;
        board[x+1][y] = emptyChar;
        --numEnemyLas;
        eLas2State = false;
        eLas2X = eLas2Y = 0;
    }
    else if(board[x+1][y] == lasChar)
    {
        board[x+1][y] = emptyChar;
        board[x][y] = emptyChar;
        --numEnemyLas;
        eLas2State = false;
        eLas2X = eLas2Y = 0;
    }
    else if(board[x+1][y] == plyrChar)
    {
        ldrBrd.lostLife();
        --numEnemyLas;
        eLas2State = false;
        eLas2X = eLas2Y = 0;
        loseLife();
    }
    else
    {
        --numEnemyLas;
        eLas2State = false;
        eLas2X = eLas2Y = 0;
    }
}

```

// this function happens when the player loses a life by being hit by an enemy  
// laser, it resets the conditions so the board can be recreated again

```
void Board::loseLife()
```

```

{
    Sleep(500);

    system("clear");

    cout << "
    cout << "| | | | ( ) / _ | | | " << endl;
    cout << "| | | | | | | | | | | | " << endl;
    cout << "| | / \ _ | | | | | / \ | | " << endl;
    cout << "| | ( ) \ \ \ | | | | | / | | " << endl;
    cout << "\ \ _ \ / \ \ _ \ / \ \ _ \ / \ \ | ( ) " << endl;

    cout << "\nPress [ENTER] to continue...";
}

```

```

cin.ignore();

system("clear");

// reset some conditions
plyrX = plyrY = 0;
moveR = 3;
moveL = 0;
enemySpeed = 15;
speedTime = 0;
downState = false;
netLR = 0;
numEnemyLas = 0;
lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;
lasX = lasY = 0;
lasStateP = false;
numEnemyLas = 0;
eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
eLas1State = eLas2State = false;
eLasSpeed = 0;

if(ldrBrd.getLives() == 0)
    loseGame();
else
{
    enemyTimer();           // determine enemy speed again
    createBoard();          // recreate the board
}
}

// this function double checks if the enemies are all gone so the player can
// either win the game or if the player has any lives left so they can lose
// the game
void Board::checkEnemies()
{
    int enemiesLeft = 0;           // how many enemies are left

    for(int i = 0 ; i < row ; i++)
    {
        for(int j = 0 ; j < col ; j++)
        {
            if(board[i][j] == enemyChar)
            {
                enemiesLeft++;
            }
        }
    }
}

```

```
if(enemiesLeft == 0)
{
    winGame();          // you win
}

if(ldrBrd.lives <= 0 )
{
    loseGame();         // you lose
}
}
```

// this function is when the player defeats all the enemies and wins the game,  
// it gives the player the option to play again and enter a highscore if the  
// player got higher than the current highscore

```
void Board::winGame()
{
    Sleep(500);
    int choice = 0;

    system("clear");

    cin.clear();

    cout << "                               " << endl;
    cout << "| |   | (|               |||" << endl;
    cout << "|| | | _ _ _ _ _ _ _ _ _ _ ||" << endl;
    cout << "| |\ \ / |' \ \ / \ \ / \ \|_|" << endl;
    cout << "\ \ \ / | | | | | | |_|_|" << endl;
    cout << "\ \ \ \|_|_|_|_|_|_\_|_ (|)" << endl;
```

// ldrBrd.setHiScore());

```
cout << " \nPress 1 to PLAY AGAIN\nPress 2 to EXIT\n[ENTER YOUR CHOICE]: ";
cin >> choice;
```

```
cin.ignore();

switch(choice)
{
    case 1: break;
    case 2: exit(0); break;
    default: cout << "\n"; winGame(); break;
}
```

// reset conditions in the constructor

```
ldrBrd.lives = 3;
ldrBrd.setScoreOne(0);
plyrX = plyrY = 0;
shield1 = shield2 = shield3 = shield4 = '4';
numEnemies = 50;
```

[illegible]

```

        case 2: exit(0); break;
        default: cout << "\n"; winGame(); break;
    }

    // reset conditions in the constructor
    ldrBrd.lives = 3;
    ldrBrd.setScoreOne(0);
    plyrX = plyrY = 0;
    shield1 = shield2 = shield3 = shield4 = '4';
    numEnemies = 50;
    moveR = 3;
    moveL = 0;
    enemySpeed = 15;
    speedTime = 0;
    downState = false;
    netLR = 0;
    lasX1 = lasX2 = lasX3 = lasX4 = lasX5 = lasX6 = lasX7 = lasX8 = lasX9 = lasX10 = 0;
    lasY1 = lasY2 = lasY3 = lasY4 = lasY5 = lasY6 = lasY7 = lasY8 = lasY9 = lasY10 = 0;
    lasX = lasY = 0;
    lasStateP = false;
    numEnemyLas = 0;
    eLas1X = eLas1Y = eLas2X = eLas2Y = 0;
    eLas1State = eLas2State = false;
    eLasSpeed = 0;
    gameOver = false;

    system("clear");

    startMenu();
    createBoard();
}

// this function stops the enemies from moving down on the row before the shields
bool Board::enemyStop(){
    bool state = false;
    int enemy = 0;
    for (int i = 0; i < col-1; i++){
        if (board[10][i] == enemyChar){
            enemy++;
        }
    }
    if (enemy > 0){
        state = true;
    }
    return state;
}

// this is a debug function to make sure there are no extra player characters
// that pop up on the board, or lasers
void Board::debug()

```

```

{
    for(int i = 1; i < 13; i++)
    {
        for(int j = 1 ; j < col ; j++)
        {
            if(board[i][j] == plyrChar)
                board[i][j] = emptyChar;
            if(lasStateP && board[i][j] == lasChar)
            {
                if(eLas1X != i && eLas1Y != j)
                {
                    if(eLas2X != i && eLas2Y != j)
                    {
                        if(lasX != i && lasY != j)
                        {
                            board[i][j] = emptyChar;
                            lasX = lasY = 0;
                            lasStateP = false;
                        }
                    }
                }
            }
        }
    }
}

```

// main.cpp //////////////////////////////////////

// -----  
// This is our main which controls the game.  
// -----

```

int main()
{
    // create board object, initialize bool variable
    Board board;
    bool notLost = true;

    // start menu
    board.startMenu();

    // create the game board
    board.createBoard();

    // play Space Invaders; display board, get input, move enemies
    while(notLost)
    {
        system("clear");           // clear screen to display board again

        board.debug();             // debug player
    }
}

```

```

    if(board.getLaserStateP())    // if player laser is on the board
        board.movPlyrLas();

    // if the enemies are to move again; left, right, or down
    if(board.getSpeedTimer() == board.getEnemySpeed())
    {
        if(board.getDownState()){
            if (board.enemyStop()){
                board.downStop();
            }else {
                board.moveEnemiesD();
            }
        }
        board.moveEnemiesLR();
    }

    board.enemyLas();           // move enemy lasers

    board.debug();              // debug player

    board.ldrBrd.getLeaderboard(); // leaderboard top
    board.dispBrd();             // game board 2D array
    board.ldrBrd.getBtmBoard();   // leaderboard bottom

    board.input();              // player input

    board.speedTimer();          // increment speed timer
    board.eLasSpeedUp();         // increment enemy laser speed timer
    board.enemyTimer();          // increment enemy movement timer

    board.checkEnemies();        // checks number of enemies on board, lives
}

return 0;
}

```