**Space Shooter**



Session: 2021 – 2024

**Submitted by:**

Umair Noor 2021-CS-207

**Supervised by:**

Mam Maida Shahid

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table Of Contents**

1. Description and Story Writing…………………………………….…Pg1
2. Game Characters……………………………….…………………….Pg1
3. Rules & Interactions.…………………………………………………Pg1
4. Goal…………………………….…………………………………….Pg1
5. Screenshots…………………………………………………………..Pg2-4
6. Data Structures………………………………………………………Pg5
7. Function Prototypes…………………………………………………Pg6
8. Full Code…………………………………………………………….Pg7-30

**Description:**

This game is a copy of the original space invaders game. In which a ship has to destroy all enemy ships with its lasers.

**Game Characters Description:**

There is a ship that will be controlled by the user. And there are enemy ships.

**Rules & Interactions:**

The user will have to fire from its ship by moving left and right and also avoid lasers shoot by enemy ships. The user only has 3 lives. If the enemy fire hits the ship its life will be decreased by one, when all lives are gone the game will be over.

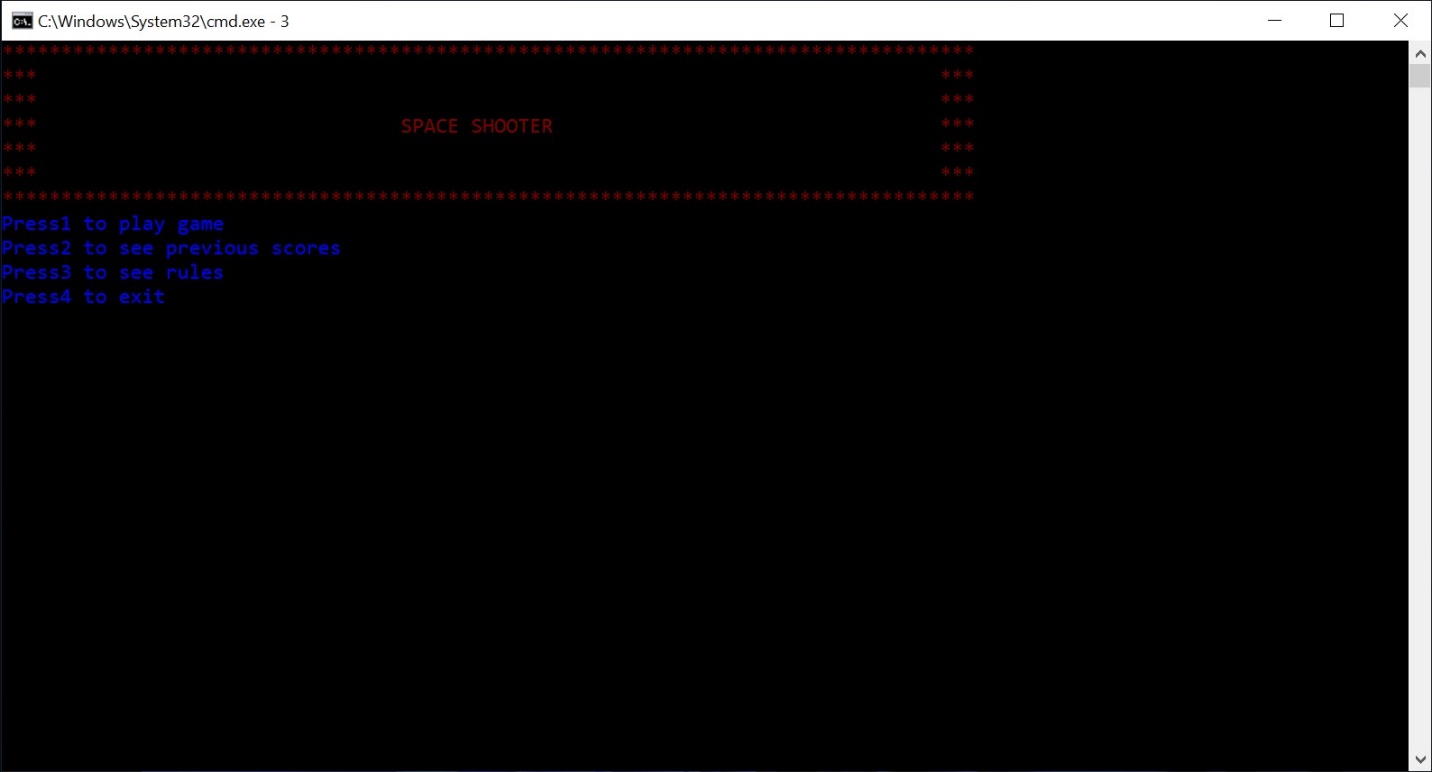
* Press left arrow key to move towards left
* Press right arrow key to move towards the right
* Press space bar to shoot

**Goal:**

Users have to destroy all of the enemy ships by shooting fire and avoid the fires coming from enemy ships.

**Screen Shots:**

Main Page



****Level1

Level2



Level3

****

Level4



**Data Structure:**

char maze[25][60];

int shipX = 22; // X-Y coordinates for user’s ship

int shipY = 30;

int Enemy1X = 2; // X-Y coordinates for enemy ship 1

int Enemy1Y = 8;

int Enemy2X = 5; // X-Y coordinates for enemy ship 2

int Enemy2Y = 43;

int Enemy3X = 8; // X-Y coordinates for enemy ship 3

int Enemy3Y = 29;

int shipLeft = 3; // Number of ships Remaining

int score = 0;

int lives = 10;

int Enemy1Movement = 0;

int Enemy2Movement = 0;

int Enemy3Movement = 0;

bool Enemy1Live = true;

bool Enemy2Live = true;

bool Enemy3Live = true;

bool NextLevel = false;

int level = 0;

int Level4KillCount = 0;

**Function Prototypes:**

void gotoxy(int x, int y);

void printMaze();

void DisplayCharacters();

void DisplayBarriers();

void Load();

void StoreScore();

void MoveShipLeft();

void MoveShipRight();

void Fire();

void MoveFire();

void CreateShip();

void DestroyShip();

void CreateEnemy(int x, int y);

void destroyEnemy(int x, int y);

void MoveEnemy1();

void MoveLevel4Enemy1();

void MoveEnemy2();

void MoveLevel4Enemy2();

void MoveEnemy3();

void MoveLevel4Enemy3();

void EnemyFire();

void MoveEnemyFire();

char menu();

void PrintData();

void resetData();

void DisplayScore();

void Rules();

void ScreenCls();

void LoosingScreen();

**Code:**

#include <iostream>

#include <fstream>

#include <ctime>

#include <windows.h>

#include <conio.h>

#include <cmath>

using namespace std;

// Data Structures

char maze[25][60];

int shipX = 22;

int shipY = 30;

int Enemy1X = 2; // X-Y coordinates for enemy ship 1

int Enemy1Y = 8;

int Enemy2X = 5; // X-Y coordinates for enemy ship 2

int Enemy2Y = 43;

int Enemy3X = 8; // X-Y coordinates for enemy ship 3

int Enemy3Y = 29;

int shipLeft = 3; // Number of ships Remaining

int score = 0;

int lives = 10;

int Enemy1Movement = 0;

int Enemy2Movement = 0;

int Enemy3Movement = 0;

bool Enemy1Live = true;

bool Enemy2Live = true;

bool Enemy3Live = true;

bool NextLevel = false;

int level = 0;

int Level4KillCount = 0;

// Function Prototypes

void gotoxy(int x, int y);

void printMaze();

void DisplayCharacters();

void DisplayBarriers();

void Load();

void StoreScore();

void MoveShipLeft();

void MoveShipRight();

void Fire();

void MoveFire();

void CreateShip();

void DestroyShip();

void CreateEnemy(int x, int y);

void destroyEnemy(int x, int y);

void MoveEnemy1();

void MoveLevel4Enemy1();

void MoveEnemy2();

void MoveLevel4Enemy2();

void MoveEnemy3();

void MoveLevel4Enemy3();

void EnemyFire();

void MoveEnemyFire();

char menu();

void PrintData();

void resetData();

void DisplayScore();

void Rules();

void ScreenCls();

void LoosingScreen();

main()

{

HANDLE console\_color;

console\_color = GetStdHandle(STD\_OUTPUT\_HANDLE);

while (true)

{

system("cls");

char option = menu();

if (option == '1')

{

score = 0;

lives = 10;

////////////////////////////////////////////////////////////// Level One

system("cls");

resetData();

level = 1;

Load();

SetConsoleTextAttribute(console\_color, 1);

printMaze();

SetConsoleTextAttribute(console\_color, 7);

DisplayCharacters();

bool gameRunning = true;

while (gameRunning)

{

Sleep(100);

PrintData();

if (GetAsyncKeyState(VK\_LEFT))

{

MoveShipLeft();

}

if (GetAsyncKeyState(VK\_RIGHT))

{

MoveShipRight();

}

if (GetAsyncKeyState(VK\_SPACE))

{

Fire();

}

if (GetAsyncKeyState(VK\_ESCAPE))

{

gameRunning = false;

}

MoveFire();

if (Enemy1Live)

{

MoveEnemy1();

}

if (Enemy2Live)

{

MoveEnemy2();

}

if (Enemy3Live)

{

MoveEnemy3();

}

if (shipLeft == 0)

{

NextLevel = true;

break;

}

}

if (NextLevel)

{

////////////////////////////////////////////////////////////// Level Two

system("cls");

resetData();

level = 2;

Load();

SetConsoleTextAttribute(console\_color, 4);

printMaze();

SetConsoleTextAttribute(console\_color, 7);

DisplayCharacters();

bool gameRunning = true;

while (gameRunning)

{

Sleep(100);

PrintData();

if (GetAsyncKeyState(VK\_LEFT))

{

MoveShipLeft();

}

if (GetAsyncKeyState(VK\_RIGHT))

{

MoveShipRight();

}

if (GetAsyncKeyState(VK\_SPACE))

{

Fire();

}

if (GetAsyncKeyState(VK\_ESCAPE))

{

gameRunning = false;

}

MoveFire();

EnemyFire();

MoveEnemyFire();

if (Enemy1Live)

{

MoveEnemy1();

}

if (Enemy2Live)

{

MoveEnemy2();

}

if (Enemy3Live)

{

MoveEnemy3();

}

if (shipLeft == 0)

{

NextLevel = true;

break;

}

if (lives == 0)

{

system("cls");

LoosingScreen();

break;

}

}

}

if (NextLevel)

{

////////////////////////////////////////////////////////////// Level Three

system("cls");

resetData();

level = 3;

Load();

SetConsoleTextAttribute(console\_color, 6);

printMaze();

DisplayBarriers();

SetConsoleTextAttribute(console\_color, 7);

DisplayCharacters();

bool gameRunning = true;

while (gameRunning)

{

Sleep(100);

PrintData();

if (GetAsyncKeyState(VK\_LEFT))

{

MoveShipLeft();

}

if (GetAsyncKeyState(VK\_RIGHT))

{

MoveShipRight();

}

if (GetAsyncKeyState(VK\_SPACE))

{

Fire();

}

if (GetAsyncKeyState(VK\_ESCAPE))

{

gameRunning = false;

}

MoveFire();

EnemyFire();

MoveEnemyFire();

if (Enemy1Live)

{

MoveEnemy1();

}

if (Enemy2Live)

{

MoveEnemy2();

}

if (Enemy3Live)

{

MoveEnemy3();

}

if (shipLeft == 0)

{

NextLevel = true;

break;

}

if (lives == 0)

{

system("cls");

LoosingScreen();

ScreenCls();

break;

}

}

}

if (NextLevel)

{

////////////////////////////////////////////////////////////// Level Two

system("cls");

resetData();

level = 4;

Load();

SetConsoleTextAttribute(console\_color, 4);

printMaze();

SetConsoleTextAttribute(console\_color, 7);

DisplayCharacters();

bool gameRunning = true;

while (gameRunning)

{

Sleep(100);

PrintData();

if (GetAsyncKeyState(VK\_LEFT))

{

MoveShipLeft();

}

if (GetAsyncKeyState(VK\_RIGHT))

{

MoveShipRight();

}

if (GetAsyncKeyState(VK\_SPACE))

{

Fire();

}

if (GetAsyncKeyState(VK\_ESCAPE))

{

gameRunning = false;

}

MoveFire();

// EnemyFire();

// MoveEnemyFire();

if (Enemy1Live)

{

MoveLevel4Enemy1();

}

else

{

destroyEnemy(Enemy1X, Enemy1Y);

Enemy1X = 3;

Enemy1Y = 8;

CreateEnemy(Enemy1X, Enemy1Y);

Enemy1Live = true;

shipLeft++;

}

if (Enemy2Live)

{

MoveLevel4Enemy2();

}

else

{

destroyEnemy(Enemy2X, Enemy2Y);

Enemy2X = 3;

Enemy2Y = 43;

CreateEnemy(Enemy2X, Enemy2Y);

Enemy2Live = true;

shipLeft++;

}

if (Enemy3Live)

{

MoveLevel4Enemy3();

}

else

{

destroyEnemy(Enemy3X, Enemy3Y);

Enemy3X = 3;

Enemy3Y = 29;

CreateEnemy(Enemy3X, Enemy3Y);

Enemy3Live = true;

shipLeft++;

}

if (score == 100)

{

break;

}

if (lives <= 0)

{

system("cls");

LoosingScreen();

break;

}

}

}

StoreScore();

}

if (option == '2')

{

DisplayScore();

ScreenCls();

}

if (option == '3')

{

Rules();

ScreenCls();

}

if (option == '4')

{

break;

}

}

}

void header()

{

HANDLE console\_color;

console\_color = GetStdHandle(STD\_OUTPUT\_HANDLE);

system("CLS");

SetConsoleTextAttribute(console\_color, 4);

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\* \*\*\*" << endl;

cout << "\*\*\* \*\*\*" << endl;

cout << "\*\*\* SPACE SHOOTER \*\*\*" << endl;

cout << "\*\*\* \*\*\*" << endl;

cout << "\*\*\* \*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

SetConsoleTextAttribute(console\_color, 7);

}

char menu()

{

header();

HANDLE console\_color;

console\_color = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(console\_color, 9);

cout << "Press1 to play game\n";

cout << "Press2 to see previous scores\n";

cout << "Press3 to see rules\n";

cout << "Press4 to exit\n";

char option;

cin >> option;

SetConsoleTextAttribute(console\_color, 7);

return option;

}

void resetData()

{

shipX = 22;

shipY = 30;

Enemy1X = 2;

Enemy1Y = 8;

Enemy2X = 5;

Enemy2Y = 43;

Enemy3X = 8;

Enemy3Y = 29;

shipLeft = 3;

Enemy1Movement = 0;

Enemy2Movement = 0;

Enemy3Movement = 0;

Enemy1Live = true;

Enemy2Live = true;

Enemy3Live = true;

NextLevel = false;

}

void gotoxy(int x, int y)

{

COORD cursorPosition;

cursorPosition.X = x;

cursorPosition.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cursorPosition);

}

void printMaze()

{

for (int row = 0; row < 25; row++)

{

for (int col = 0; col < 60; col++)

{

cout << maze[row][col];

}

cout << endl;

}

}

void StoreScore()

{

fstream storeScore;

storeScore.open("ScoreBoard.txt", ios::app);

storeScore << endl

<< score;

storeScore.close();

}

void DisplayScore()

{

header();

int count = 1;

fstream display;

display.open("ScoreBoard.txt", ios::in);

while (!display.eof())

{

display >> score;

if (score > 0)

{

cout << count << ".\t" << score << endl;

count++;

}

}

score = 0;

display.close();

}

void Rules()

{

header();

cout << "Rules\n\n";

cout << "Rule 1: " << endl;

cout << "You can only move Left and Right by using your Arrow Keys\n";

cout << "Rule 2: " << endl;

cout << "You can fire from your ship by pressing Space Bar\n";

cout << "Rule 3: " << endl;

cout << "You have to Dodge the fire coming from enemy ships\n";

cout << "Rule 4: " << endl;

cout << "Enemies won't get killed until you shoot their antinaas\n";

cout << "Rule 5: " << endl;

cout << "Each Enemy ship kills increase your score by 5\n";

cout << "Rule 6: " << endl;

cout << "You have to complete a level to continue to next level\n";

cout << "Level1: " << endl;

cout << "Shoot the enemies while they are just moving\n";

cout << "Level2: " << endl;

cout << "Enemies will also shoot\n";

cout << "Level3: " << endl;

cout << "There will be barriers in the play board\n";

}

void Load()

{

fstream loadmaze;

loadmaze.open("spaceshooterload.txt", ios::in);

string line;

int row = 0;

while (!loadmaze.eof())

{

getline(loadmaze, line);

for (int col = 0; col < 60; col++)

{

maze[row][col] = line[col];

}

row++;

}

}

void PrintData()

{

gotoxy(70, 2);

cout << "Lives: " << lives << " ";

gotoxy(70, 4);

cout << "Score: " << score << " ";

}

void LoosingScreen()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*\*\* \* \* \*\*\* \* \* \* \*\*\* \*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*" << endl;

cout << "\*\*\* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*" << endl;

cout << "\*\*\* \* \* \* \* \* \* \* \* \* \* \* \*\*\*" << endl;

cout << "\*\*\* \* \* \* \* \* \* \* \* \*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*" << endl;

cout << "\*\*\* \* \* \* \* \* \* \* \* \* \* \*\*\*" << endl;

cout << "\*\*\* \* \* \* \* \* \* \* \* \* \* \* \*\*\*" << endl;

cout << "\*\*\* \* \*\*\* \*\*\*\* \*\*\*\*\*\*\* \*\*\* \*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

char op;

cin >> op;

}

void DisplayCharacters()

{

CreateShip();

CreateEnemy(Enemy1X, Enemy1Y);

CreateEnemy(Enemy2X, Enemy2Y);

CreateEnemy(Enemy3X, Enemy3Y);

}

void CreateShip()

{

maze[shipX][shipY - 3] = '\_';

maze[shipX][shipY - 2] = '\_';

maze[shipX][shipY - 1] = '/';

maze[shipX][shipY] = '|';

maze[shipX][shipY + 1] = '\\';

maze[shipX][shipY + 2] = '\_';

maze[shipX][shipY + 3] = '\_';

maze[shipX + 1][shipY - 3] = ' ';

maze[shipX + 1][shipY - 2] = '/';

maze[shipX + 1][shipY - 1] = '\_';

maze[shipX + 1][shipY] = '|';

maze[shipX + 1][shipY + 1] = '\_';

maze[shipX + 1][shipY + 2] = '\\';

maze[shipX + 1][shipY + 3] = ' ';

gotoxy(shipY - 3, shipX);

cout << "\_\_/|\\\_\_ ";

gotoxy(shipY - 3, shipX + 1);

cout << " /\_|\_\\ ";

}

void DestroyShip()

{

maze[shipX][shipY - 3] = ' ';

maze[shipX][shipY - 2] = ' ';

maze[shipX][shipY - 1] = ' ';

maze[shipX][shipY] = ' ';

maze[shipX][shipY + 1] = ' ';

maze[shipX][shipY + 2] = ' ';

maze[shipX][shipY + 3] = ' ';

maze[shipX][shipY + 4] = ' ';

maze[shipX + 1][shipY - 3] = ' ';

maze[shipX + 1][shipY - 2] = ' ';

maze[shipX + 1][shipY - 1] = ' ';

maze[shipX + 1][shipY] = ' ';

maze[shipX + 1][shipY + 1] = ' ';

maze[shipX + 1][shipY + 2] = ' ';

maze[shipX + 1][shipY + 3] = ' ';

gotoxy(shipY - 3, shipX);

cout << " ";

gotoxy(shipY - 3, shipX + 1);

cout << " ";

}

void CreateEnemy(int x, int y)

{

maze[x - 1][y - 4] = ' ';

maze[x - 1][y - 3] = ' ';

maze[x - 1][y - 2] = '\_';

maze[x - 1][y - 1] = '\_';

maze[x - 1][y] = '\_';

maze[x - 1][y + 1] = '\_';

maze[x - 1][y + 2] = '\_';

maze[x - 1][y + 3] = ' ';

maze[x - 1][y + 4] = ' ';

maze[x][y - 4] = '<';

maze[x][y - 3] = '|';

maze[x][y - 2] = '\_';

maze[x][y - 1] = '\_';

maze[x][y] = '\_';

maze[x][y + 1] = '\_';

maze[x][y + 2] = '\_';

maze[x][y + 3] = '|';

maze[x][y + 4] = '>';

maze[x + 1][y - 4] = ' ';

maze[x + 1][y - 3] = ' ';

maze[x + 1][y - 2] = ' ';

maze[x + 1][y - 1] = '|';

maze[x + 1][y] = ' ';

maze[x + 1][y + 1] = '|';

maze[x + 1][y + 2] = ' ';

maze[x + 1][y + 3] = ' ';

maze[x + 1][y + 4] = ' ';

gotoxy(y - 4, x - 1);

cout << " \_\_\_\_\_ ";

gotoxy(y - 4, x);

cout << "<|\_\_\_\_\_|>";

gotoxy(y - 4, x + 1);

cout << " | | ";

}

void destroyEnemy(int x, int y)

{

maze[x - 1][y - 4] = ' ';

maze[x - 1][y - 3] = ' ';

maze[x - 1][y - 2] = ' ';

maze[x - 1][y - 1] = ' ';

maze[x - 1][y] = ' ';

maze[x - 1][y + 1] = ' ';

maze[x - 1][y + 2] = ' ';

maze[x - 1][y + 3] = ' ';

maze[x - 1][y + 4] = ' ';

maze[x][y - 4] = ' ';

maze[x][y - 3] = ' ';

maze[x][y - 2] = ' ';

maze[x][y - 1] = ' ';

maze[x][y] = ' ';

maze[x][y + 1] = ' ';

maze[x][y + 2] = ' ';

maze[x][y + 3] = ' ';

maze[x][y + 4] = ' ';

maze[x + 1][y - 4] = ' ';

maze[x + 1][y - 3] = ' ';

maze[x + 1][y - 2] = ' ';

maze[x + 1][y - 1] = ' ';

maze[x + 1][y] = ' ';

maze[x + 1][y + 1] = ' ';

maze[x + 1][y + 2] = ' ';

maze[x + 1][y + 3] = ' ';

maze[x + 1][y + 4] = ' ';

gotoxy(y - 4, x - 1);

cout << " ";

gotoxy(y - 4, x);

cout << " ";

gotoxy(y - 4, x + 1);

cout << " ";

}

void MoveShipLeft()

{

if (maze[shipX][shipY - 4] != '#')

{

DestroyShip();

shipY--;

// gotoxy(shipY - 3, shipX);

CreateShip();

}

}

void MoveShipRight()

{

if (maze[shipX][shipY + 5] != '#')

{

DestroyShip();

shipY++;

CreateShip();

}

}

void Fire()

{

gotoxy(shipY, shipX - 1);

cout << "^";

maze[shipX - 1][shipY] = '^';

}

void MoveFire()

{

for (int row = 0; row < 25; row++)

{

for (int col = 0; col < 60; col++)

{

if (maze[row][col] == '^')

{

gotoxy(col, row);

cout << " ";

maze[row][col] = ' ';

if (maze[row - 1][col] != '#')

{

if (maze[row - 1][col] == '|' || maze[row - 1][col] == '\_' || maze[row - 1][col] == '<' || maze[row - 1][col] == '>')

{

if ((Enemy1X == (row - 1) || Enemy1X == (row - 1) + 1 || Enemy1X == (row - 1) - 1) && (Enemy1Y - 1 == col || Enemy1Y + 1 == col))

{

destroyEnemy(Enemy1X, Enemy1Y);

shipLeft--;

Enemy1Live = false;

score = score + 5;

}

if ((Enemy2X == (row - 1) || Enemy2X == (row - 1) + 1 || Enemy2X == (row - 1) - 1) && (Enemy2Y - 1 == col || Enemy2Y + 1 == col))

{

destroyEnemy(Enemy2X, Enemy2Y);

shipLeft--;

Enemy2Live = false;

score = score + 5;

}

if ((Enemy3X == (row - 1) || Enemy3X == (row - 1) + 1 || Enemy3X == (row - 1) - 1) && (Enemy3Y - 1 == col || Enemy3Y + 1 == col))

{

destroyEnemy(Enemy3X, Enemy3Y);

shipLeft--;

Enemy3Live = false;

score = score + 5;

}

gotoxy(col, row - 1);

cout << " ";

maze[row - 1][col] = ' ';

}

else

{

gotoxy(col, row - 1);

cout << "^";

maze[row - 1][col] = '^';

}

}

}

}

}

}

void MoveEnemy1()

{

if (Enemy1Movement == 0)

{

if (maze[Enemy1X][Enemy1Y + 5] != '#')

{

destroyEnemy(Enemy1X, Enemy1Y);

Enemy1Y++;

CreateEnemy(Enemy1X, Enemy1Y);

}

else

{

Enemy1Movement = 1;

}

}

else if (Enemy1Movement == 1)

{

if (maze[Enemy1X][Enemy1Y - 5] != '#')

{

destroyEnemy(Enemy1X, Enemy1Y);

Enemy1Y--;

CreateEnemy(Enemy1X, Enemy1Y);

}

else

{

Enemy1Movement = 0;

}

}

}

void MoveEnemy2()

{

if (Enemy2Movement == 0)

{

if (maze[Enemy2X][Enemy2Y + 5] != '#')

{

destroyEnemy(Enemy2X, Enemy2Y);

Enemy2Y++;

CreateEnemy(Enemy2X, Enemy2Y);

}

else

{

Enemy2Movement = 1;

}

}

else if (Enemy2Movement == 1)

{

if (maze[Enemy2X][Enemy2Y - 5] != '#')

{

destroyEnemy(Enemy2X, Enemy2Y);

Enemy2Y--;

CreateEnemy(Enemy2X, Enemy2Y);

}

else

{

Enemy2Movement = 0;

}

}

}

void MoveEnemy3()

{

if (Enemy3Movement == 0)

{

if (maze[Enemy3X][Enemy3Y + 5] != '#')

{

destroyEnemy(Enemy3X, Enemy3Y);

Enemy3Y++;

CreateEnemy(Enemy3X, Enemy3Y);

}

else

{

Enemy3Movement = 1;

}

}

else if (Enemy3Movement == 1)

{

if (maze[Enemy3X][Enemy3Y - 5] != '#')

{

destroyEnemy(Enemy3X, Enemy3Y);

Enemy3Y--;

CreateEnemy(Enemy3X, Enemy3Y);

}

else

{

Enemy3Movement = 0;

}

}

}

int EnemyRandomFire()

{

// srand(time(0));

int value = 1 + (rand() % 3);

return value;

}

void EnemyFire()

{

int value = EnemyRandomFire();

if (value == 1)

{

if (Enemy1Live)

{

if (maze[Enemy1X + 2][Enemy1Y] != '#')

{

gotoxy(Enemy1Y, Enemy1X + 2);

cout << "o";

maze[Enemy1X + 2][Enemy1Y] = 'o';

}

}

}

if (value == 2)

{

if (Enemy2Live)

{

if (maze[Enemy2X + 2][Enemy2Y] != '#')

{

gotoxy(Enemy2Y, Enemy2X + 2);

cout << "o";

maze[Enemy2X + 2][Enemy2Y] = 'o';

}

}

}

if (value == 3)

{

if (Enemy3Live)

{

if (maze[Enemy3X + 2][Enemy3Y] != '#')

{

gotoxy(Enemy3Y, Enemy3X + 2);

cout << "o";

maze[Enemy3X + 2][Enemy3Y] = 'o';

}

}

}

}

void MoveEnemyFire()

{

for (int row = 25; row > 0; row--)

{

for (int col = 0; col < 60; col++)

{

if (maze[row][col] == 'o')

{

gotoxy(col, row);

cout << " ";

maze[row][col] = ' ';

if (maze[row + 1][col] != '#')

{

if (shipX == row + 1 && (col >= shipY - 3 && col <= shipY + 3))

{

DestroyShip();

shipX = 22;

shipY = 30;

CreateShip();

lives--;

}

else

{

gotoxy(col, row + 1);

cout << "o";

maze[row + 1][col] = 'o';

}

}

}

}

}

}

void DisplayBarriers()

{

gotoxy(3, 15);

cout << "###### ###### ######";

maze[15][3] = '#';

maze[15][4] = '#';

maze[15][5] = '#';

maze[15][6] = '#';

maze[15][7] = '#';

maze[15][8] = '#';

maze[15][16] = '#';

maze[15][17] = '#';

maze[15][18] = '#';

maze[15][19] = '#';

maze[15][20] = '#';

maze[15][21] = '#';

maze[15][31] = '#';

maze[15][32] = '#';

maze[15][33] = '#';

maze[15][34] = '#';

maze[15][35] = '#';

maze[15][36] = '#';

}

void MoveLevel4Enemy1()

{

if (maze[Enemy1X + 2][Enemy1Y] != '#')

{

if (shipX == Enemy1X + 2 && (Enemy1Y >= shipY - 3 && Enemy1Y <= shipY + 3))

{

DestroyShip();

shipX = 22;

shipY = 30;

CreateShip();

destroyEnemy(Enemy1X, Enemy1Y);

Enemy1X = 3;

Enemy1Y = 8;

CreateEnemy(Enemy1X, Enemy1Y);

lives--;

}

else

{

destroyEnemy(Enemy1X, Enemy1Y);

Enemy1X++;

CreateEnemy(Enemy1X, Enemy1Y);

}

}

else

{

destroyEnemy(Enemy1X, Enemy1Y);

Enemy1X = 3;

Enemy1Y = 8;

CreateEnemy(Enemy1X, Enemy1Y);

lives--;

}

}

void MoveLevel4Enemy2()

{

if (maze[Enemy2X + 2][Enemy2Y] != '#')

{

if (shipX == Enemy2X + 2 && (Enemy2Y >= shipY - 3 && Enemy2Y <= shipY + 3))

{

DestroyShip();

shipX = 22;

shipY = 30;

CreateShip();

destroyEnemy(Enemy2X, Enemy2Y);

Enemy2X = 3;

Enemy2Y = 43;

CreateEnemy(Enemy2X, Enemy2Y);

lives--;

}

else

{

destroyEnemy(Enemy2X, Enemy2Y);

Enemy2X++;

CreateEnemy(Enemy2X, Enemy2Y);

}

}

else

{

destroyEnemy(Enemy2X, Enemy2Y);

Enemy2X = 3;

Enemy2Y = 43;

CreateEnemy(Enemy2X, Enemy2Y);

lives--;

}

}

void MoveLevel4Enemy3()

{

if (maze[Enemy3X + 2][Enemy3Y] != '#')

{

if (shipX == Enemy3X + 2 && (Enemy3Y >= shipY - 3 && Enemy3Y <= shipY + 3))

{

DestroyShip();

shipX = 22;

shipY = 30;

CreateShip();

destroyEnemy(Enemy3X, Enemy3Y);

Enemy3X = 3;

Enemy3Y = 29;

CreateEnemy(Enemy3X, Enemy3Y);

lives--;

}

else

{

destroyEnemy(Enemy3X, Enemy3Y);

Enemy3X++;

CreateEnemy(Enemy3X, Enemy3Y);

}

}

else

{

destroyEnemy(Enemy3X, Enemy3Y);

Enemy3X = 3;

Enemy3Y = 29;

CreateEnemy(Enemy3X, Enemy3Y);

lives--;

}

}

void ScreenCls()

{

cout << "Enter any key to Continue\n";

getch();

system("cls");

}

**Student Reg. No. :**   **Student Name.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria are not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistent and according to given **guidelines**. The project **Poster** is professionally designed and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meets more than 80% of the criteria given. | Documentation meets more than 50% of the criteria. | When the documentation meets less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Short Description and Story Writing of Game** - **Game Characters** Description - **Rules** & Interactions - **Goal** of the Game **- Screenshot** of the Game - **Data Structures** Used in the Game - **Functions** Prototype - **Full Code** | | | | |
| Project Complexity  **Grade:** | The project has at least 1 Player and 3 enemies. Proper use of gotoxy() function. Health system, Firing System and lives decreasing system. In the case of the board game (Pong, Ludo, and 2048 is acceptable) | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Randomness  **Grade:** | Objects are produced randomly in the game. | meet more than 80% of the criteria given. | meet more than 50% of the criteria given. | Objects are appearing in the same pattern |
| Code Style  **Grade:** | All Code style criteria are followed | All code style criteria followed but some improvements required | A lot of improvements are required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation are synchronized. | Code and documentation does not synchronize at **some** places | Code and documentation do not synchronize in **many** places | Code and documentation **do not** synchronize. |
| Idea Novelty and Creativity  **Grade:** | The idea is unique of the game | The idea is merged by combining other different games | Same idea as a previous game | Could not implement the existing game idea. |
| Data Structure (2D Arrays)  **Grade:** | The data structure is sufficient for the project requirements | Data Structure is sufficient but requires improvement to meet project requirements. | The data structure is not sufficient and needs a lot of improvement | Data Structure is not properly identified and declared. |
| File Handling  **Grade:** | The game maze is loaded and the updated maze is stored in the file | The game maze is loaded and partial data is stored in the file. | The game maze is loaded, but the updated game configuration is not stored. | The project does not contain file handling |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places, it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- There is no global variable defined. Arrays and variables are passed as parameters to the functions. Functions exhibit a single responsibility principle. | | | | |
| Screen flickering  **Grade:** | There is no Screen flickering. | Maze is not flickering but the characters are flickering at great speed | Flickering is done a lot of places | The screen is flickering at all places |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | The presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | The student has a complete understanding of how the code is working and knows the concept. | The student has good understanding but in someplace he does not know the concepts | The student has very little understanding and lacks the major concepts. | The student does not have any level of understanding of the code. |

|  |  |
| --- | --- |
| **Checked by:** | Click or tap here to enter text. |