War Thunder

Session 2023-2027

Submitted by:

Huzaifa Arshad 2023-CS-86

Supervised By:

Dr. Muhammad Awais Hassan

Course:

CSC-102 Programming Fundamentals

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Content

➢ Overview

My game is a firing competition between a player and enemies. If player shoots enemies and score reaches 150 then player wins and if player health decreases to zero due to shooting of enemies, then player loses.

➢ Game Characters

There are following characters in my game.

- Player
- Enemy1
- Enemy2

✓ Player

My player will shoot bullet and due to shooting its score increases by 7. If enemy hits bullet to player then player health decreases by 8.

✓ Enemy1

My enemy1 will come moving to left and the up and then again left shooting bullets to player all time.

✓ Enemy2

My enemy2 will come moving to left and then move up and down and shoot bullets to the player.

➢ Power up

There is a pill at a certain location when the player collects it its health increases by 10.

➢ Rules and Regulations

1. Player is moved by pressing arrow keys.
2. When player fires bullet to enemy its score increases by 7.
3. When enemy fires bullet to player the player health decreases by 8.
4. Firing is done by using space Key.
5. If score reaches 150, player will win.
6. If Health reaches zero, Player will lose.

➢ Wireframes

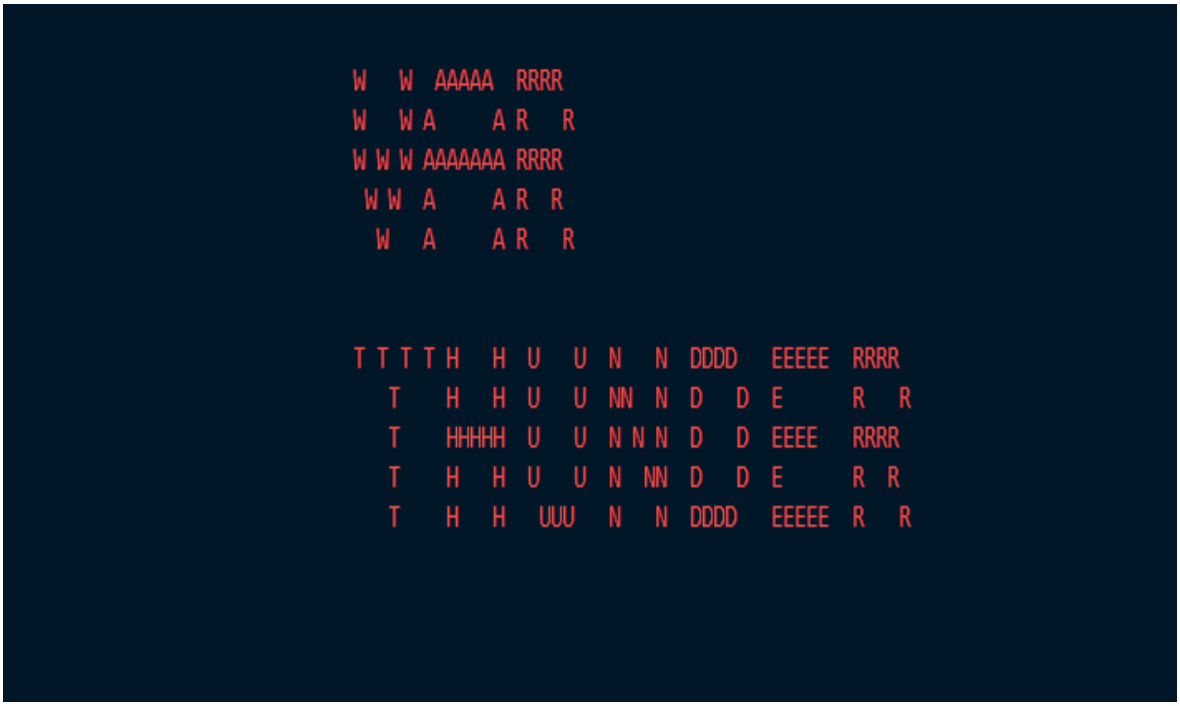

Figure 1 Header

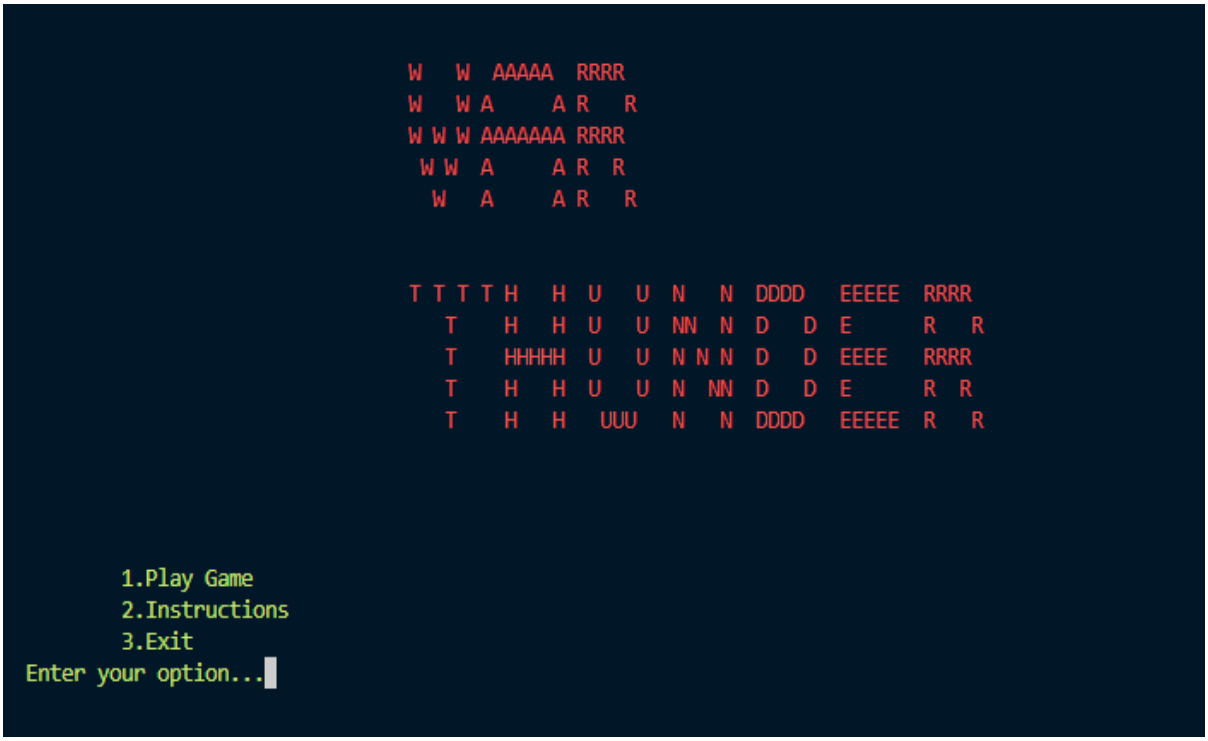

Figure 2 Menu

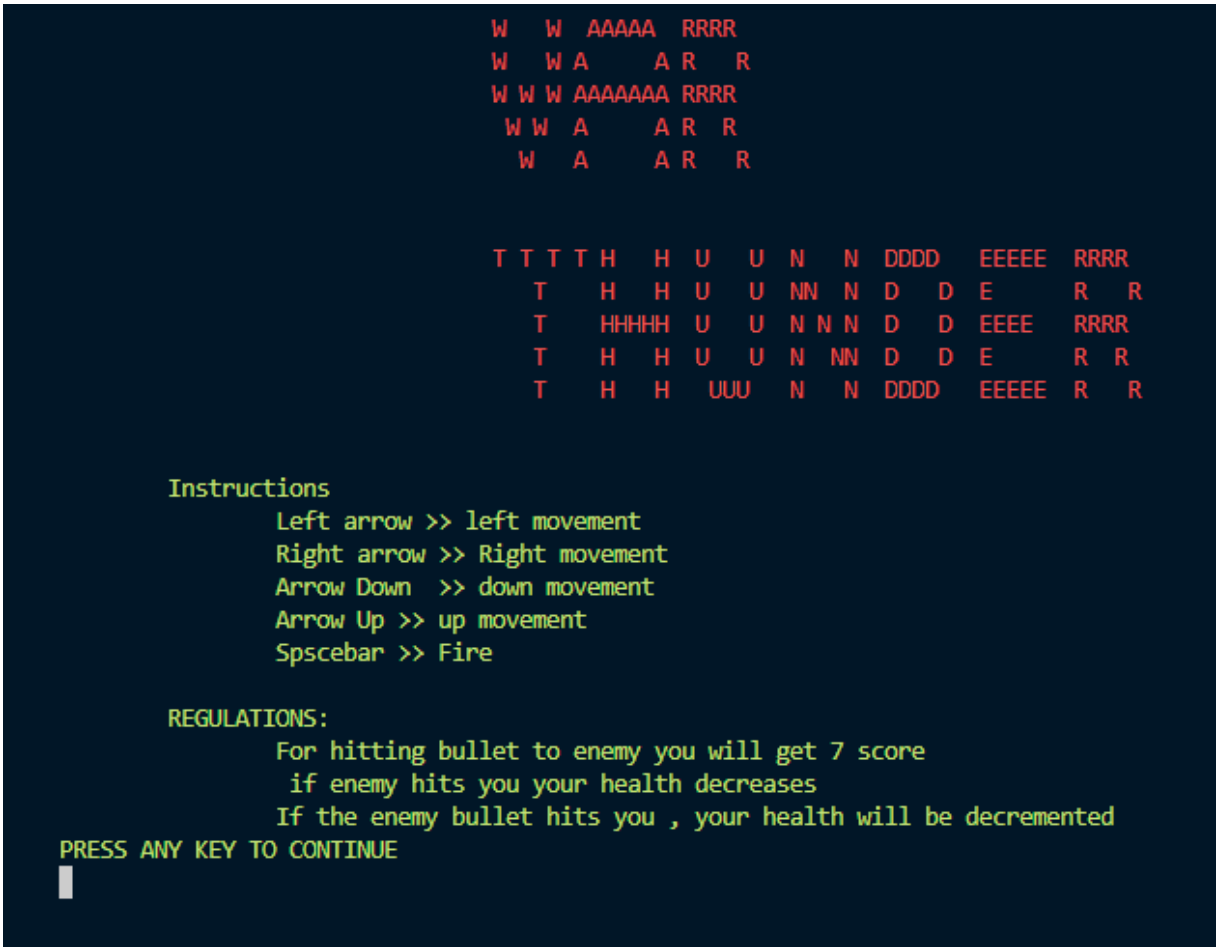

Figure 3 Instructions

Figure 4

Figure 5

Figure 6

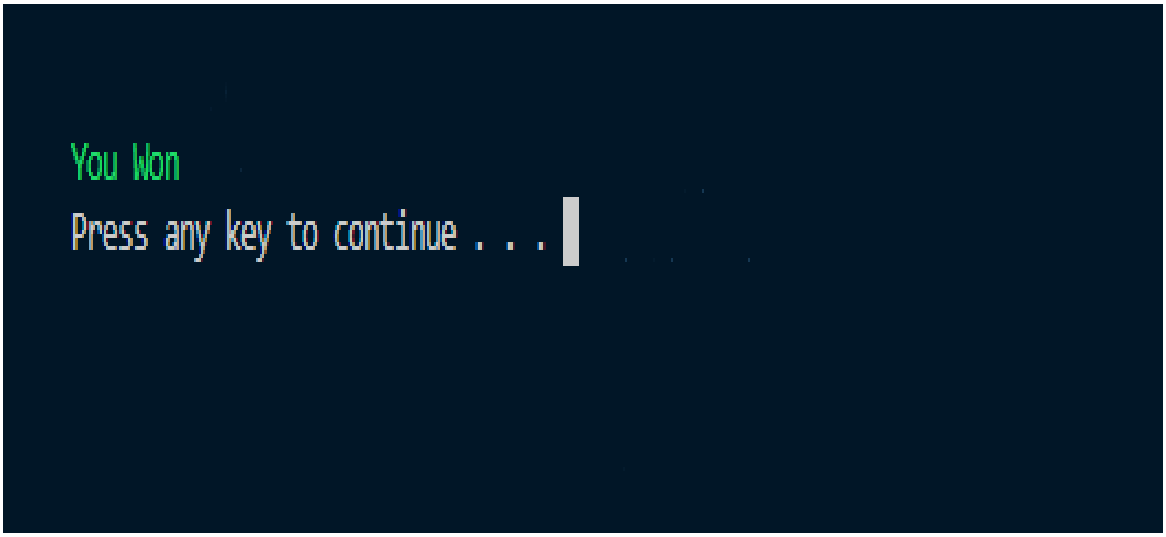

Figure 7 win

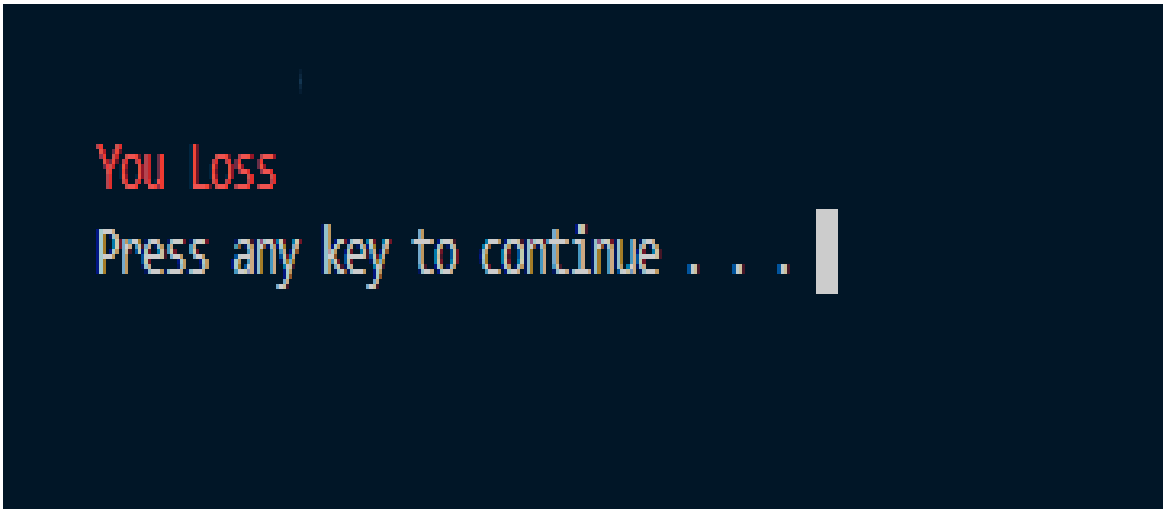

Figure 8 loss

➢ Data Structures

- **Variables**

```
1.      int px = 30;
2.      int py = 10;
3.      int e1x = 110;
4.      int e1y = 20;
5.      int e2x = 110;
6.      int e2y = 5
7.      int pillX = 50;
8.      int pillY = 10;
9.      int bx[1005];
10.     int by[1005];
11.     int Enemy1BulletX[800];
12.     int Enemy1BulletY[800];
13.     int Enemy1BulletCount = 0;
14.     int bulletCount = 0;
15.     bool flag1 = false;
16.     bool flag2 = false;
17.     int score = 0;
18.     int Health = 100;
19.     int count = 0;
20.     int enemy2BulletX[800];
21.     int enemy2BulletY[800];
22.     int enemy2BulletCount = 0;
23.     string direction = "left";
24.     string direction2 = "left";
25.     char previousLoc = ' ';
26.     string enemy1Active = "true";
27.     string enemy2Active = "true"; int aftertime = 0;
28.     int count1 = 0;
29.     int count2 = 0;
```

➢ Function Prototypes

```
1. void printHeader();
2. // BASIC FUNCTIONS//
3. void printMaze();
4. void setColor(int color);
5. void gotoxy(int x, int y);
6. char getCharATXY(short int x, short int y);
7. void addScore(int &score);
8. void minusScore(int &Health);
9. string Menu();
10.string validcheck(string option);
11.void inst();
12.void win();
13.void loss();
14.void showScore(int playerScore, int health);
15./*SCORING FUNCTIONS*/
16.bool checkScore(int score);
17.bool checkHealth(int health);
18.void generatePowerUp(int &pillX ,int &pillY );
19.void addPowerUp(int &px ,int &py,int &pillX ,int &pillY ,int
   &Health);
20.void removepowerUp(int &Health);
21.// player
22.void printPlayer(int &px ,int &py);
23.void erasePlayer(int &px, int &py);
24.void playerLeft(int &px ,int &py);
25.void playerRight(int &px ,int &py);
26.void playerUp(int &px ,int &py);
27.void playerDown(int &px ,int &py);
28.void printBullet(int x, int y);
29.void generateBullet(int &px ,int &py,int bx[],int by[],int
   &bulletCount);
30.void moveBullet(int bx[],int by[],int &bulletCount);
31.void removeBullet(int x,int bx[],int by[],int &bulletCount);
32.void eraseBullet(int x, int y);
33.// enemy1
34.void printenemy1(int &e1x, int &e1y);
35.void eraseenemy1(int &e1x, int &e1y);
36.void moveEnemy1Left(int &e1x , int &e1y,string &direction,char
   &previousLoc);
37.void moveEnemy1Right(int &e1x , int &e1y,string &direction,char
   &previousLoc);
```

```
38. void moveEnemy1Up(int &e1x , int &e1y,string &direction,char
    &previousLoc);
39. void Enemy1printBullet(int x, int y);
40. void enemy1BulletErase(int x, int y);
41. void generateEnemy1Bullet(int &e1x , int &e1y,int
    Enemy1BulletX[],int Enemy1BulletY[],int &Enemy1BulletCount);
42. void removeEnemy1Bullet(int x,int Enemy1BulletX[],int
    Enemy1BulletY[],int &Enemy1BulletCount);
43. void moveEnemy1Bullet(int Enemy1BulletX[],int Enemy1BulletY[],int
    &Enemy1BulletCount);
44. // enemy2
45. void printenemy2(int &e2x ,int &e2y);
46. void eraseenemy2(int &e2x ,int &e2y);
47. void moveEnemy2Left(int &e2x ,int &e2y,string &direction2,char
    &previousLoc);
48. void moveEnemy2Right(int &e2x ,int &e2y,string &direction2,char
    &previousLoc);
49. void moveEnemy2Down(int &e2x ,int &e2y,string &direction2,char
    &previousLoc);
50. void Enemy2printBullet(int x, int y);
51. void enemy2BulletErase(int x, int y);
52. void generateEnemy2Bullet(int &e2x ,int &e2y,int
    enemy2BulletX[],int enemy2BulletY[],int &enemy2BulletCount);
53. void removeEnemy2Bullet(int x,int enemy2BulletX[],int
    enemy2BulletY[],int &enemy2BulletCount);
54. void moveEnemy2Bullet(int enemy2BulletX[],int enemy2BulletY[],int
    &enemy2BulletCount);
55. /*COLLOSION CHECK*/
56. void collosion(int &e1x , int &e1y,int bx[],int by[],int
    &bulletCount,int &score,int &count,string &enemy1Active ,int
    &count1 );
57. void collosion2(int &e2x ,int &e2y,int bx[],int by[],int
    &bulletCount,int &score,int &count ,string &enemy2Active,int
    &count2 );
58. void enemyCollosion(int &px ,int &py,int Enemy1BulletX[],int
    Enemy1BulletY[],int &Enemy1BulletCount,int &Health);
59. void enemyCollosion2(int &px ,int &py,int &Health,int
    enemy2BulletX[],int enemy2BulletY[],int &enemy2BulletCount);
```

➢ Complete code of the Game

```
#include <iostream>
#include <windows.h>
#include <conio.h>
using namespace std;
// BASIC FUNCTIONS//
void printHeader(); //print the main logo
void printMaze();   //print the maze
void setColor(int color); //ste the color of text
void gotoxy(int x, int y); //
char getCharATXY(short int x, short int y);
void addScore(int &score);
void minusScore(int &Health);

string Menu();
string validcheck(string option);
void inst();
void win();
void loss();
void showScore(int playerScore, int health);
/*SCORING FUNCTIONS*/
bool checkScore(int score);
bool checkHealth(int health);
void generatePowerUp(int &pillX ,int &pillY );
void addPowerUp(int &px ,int &py,int &pillX ,int &pillY ,int &Health);
void removepowerUp(int &Health);

// player
void printPlayer(int &px ,int &py);
void erasePlayer(int &px, int &py);
void playerLeft(int &px ,int &py);
void playerRight(int &px ,int &py);
void playerUp(int &px ,int &py);
void playerDown(int &px ,int &py);
void printBullet(int x, int y);
void generateBullet(int &px ,int &py,int bx[],int by[],int &bulletCount);
void moveBullet(int bx[],int by[],int &bulletCount);
void removeBullet(int x,int bx[],int by[],int &bulletCount);
void eraseBullet(int x, int y);
// enemy1
void printenemy1(int &e1x, int &e1y);
```

```
void eraseenemy1(int &e1x, int &e1y);
void moveEnemy1Left(int &e1x , int &e1y,string &direction,char &previousLoc);
void moveEnemy1Right(int &e1x , int &e1y,string &direction,char &previousLoc);
void moveEnemy1Up(int &e1x , int &e1y,string &direction,char &previousLoc);
void Enemy1printBullet(int x, int y);
void enemy1BulletErase(int x, int y);
void generateEnemy1Bullet(int &e1x , int &e1y,int Enemy1BulletX[],int
Enemy1BulletY[],int &Enemy1BulletCount);
void removeEnemy1Bullet(int x,int Enemy1BulletX[],int Enemy1BulletY[],int
&Enemy1BulletCount);
void moveEnemy1Bullet(int Enemy1BulletX[],int Enemy1BulletY[],int
&Enemy1BulletCount);
// enemy2
void printenemy2(int &e2x ,int &e2y);
void eraseenemy2(int &e2x ,int &e2y);
void moveEnemy2Left(int &e2x ,int &e2y,string &direction2,char &previousLoc);
void moveEnemy2Right(int &e2x ,int &e2y,string &direction2,char &previousLoc);
void moveEnemy2Down(int &e2x ,int &e2y,string &direction2,char &previousLoc);
void Enemy2printBullet(int x, int y);
void enemy2BulletErase(int x, int y);
void generateEnemy2Bullet(int &e2x ,int &e2y,int enemy2BulletX[],int
enemy2BulletY[],int &enemy2BulletCount);
void removeEnemy2Bullet(int x,int enemy2BulletX[],int enemy2BulletY[],int
&enemy2BulletCount);
void moveEnemy2Bullet(int enemy2BulletX[],int enemy2BulletY[],int
&enemy2BulletCount);

/*COLLOSION CHECK*/
void collosion(int &e1x , int &e1y,int bx[],int by[],int &bulletCount,int
&score,int &count,string &enemy1Active ,int &count1 );
void collosion2(int &e2x ,int &e2y,int bx[],int by[],int &bulletCount,int
&score,int &count ,string &enemy2Active,int &count2 );
void enemyCollosion(int &px ,int &py,int Enemy1BulletX[],int Enemy1BulletY[],int
&Enemy1BulletCount,int &Health);
void enemyCollosion2(int &px ,int &py,int &Health,int enemy2BulletX[],int
enemy2BulletY[],int &enemy2BulletCount);

main()
{
    int px = 30;
    int py = 10;

    int e1x = 110;
    int e1y = 20;
```

```
int e2x = 110;
int e2y = 5;

int pillX = 50;
int pillY = 10;

int bx[800];
int by[800];

int Enemy1BulletX[800];
int Enemy1BulletY[800];
int Enemy1BulletCount = 0;

int bulletCount = 0;

bool flag1 = false;
bool flag2 = false;

int score = 0;
int Health = 100;

int count = 0;
int enemy2BulletX[1000];
int enemy2BulletY[1000];

int enemy2BulletCount = 0;
string direction = "left";
string direction2 = "left";
char previousLoc = ' ';
string enemy1Active = "true";
string enemy2Active = "true";
int aftertime = 0;
int count1 = 0;
int count2 = 0;

system("cls");
printHeader();
getch();
system("cls");
printHeader();
int option1 = 0;

while (option1 != 4)
```

```
    {
        system("cls");
        printHeader();
        string option1 = Menu();

        if (option1 == "1")
        {

            flag1 = false;
            flag2 = false;
            bool game = true;

            system("cls");
            printMaze();

            while (game == true)

            {
                showScore(score, Health);

                /*ENEMY1 MOVEMENTS*/
                if (enemy1Active == "true")
                {
                    if (direction == "left")
                    {
                        moveEnemy1Left(e1x,e1y,direction,previousLoc);
                        generateEnemy1Bullet(e1x,e1y, Enemy1BulletX,
Enemy1BulletY,Enemy1BulletCount);

                    }
                    moveEnemy1Bullet( Enemy1BulletX,
Enemy1BulletY,Enemy1BulletCount);

                    if (direction == "up")
                    {
                        moveEnemy1Up(e1x,e1y,direction,previousLoc);
                        generateEnemy1Bullet(e1x,e1y, Enemy1BulletX,
Enemy1BulletY,Enemy1BulletCount);

                    }
                    moveEnemy1Bullet( Enemy1BulletX,
Enemy1BulletY,Enemy1BulletCount);
                    if (direction == "right")
                    {
```

```
                        moveEnemy1Right(e1x,e1y,direction,previousLoc);
                        generateEnemy1Bullet(e1x,e1y, Enemy1BulletX,
Enemy1BulletY,Enemy1BulletCount);

                    }
                    moveEnemy1Bullet(Enemy1BulletX,
Enemy1BulletY,Enemy1BulletCount);
                }
                /*ENEMY 2 MOVEMENTS*/
                if (enemy2Active == "true")
                {
                    if (direction2 == "left")
                    {
                        moveEnemy2Left(e2x ,e2y,direction2,previousLoc);
                        generateEnemy2Bullet(e2x
,e2y,enemy2BulletX,enemy2BulletY,enemy2BulletCount);

                    }
                    moveEnemy2Bullet(enemy2BulletX,enemy2BulletY,enemy2BulletCoun
t);
                    if (direction2 == "down")
                    {
                        moveEnemy2Down(e2x ,e2y,direction2,previousLoc);
                        generateEnemy2Bullet(e2x
,e2y,enemy2BulletX,enemy2BulletY,enemy2BulletCount);

                    }
                    moveEnemy2Bullet(enemy2BulletX,enemy2BulletY,enemy2BulletCoun
t);
                    if (direction2 == "right")
                    {
                        moveEnemy2Right(e2x ,e2y,direction2,previousLoc);
                        generateEnemy2Bullet(e2x
,e2y,enemy2BulletX,enemy2BulletY,enemy2BulletCount);

                    }
                    moveEnemy2Bullet(enemy2BulletX,enemy2BulletY,enemy2BulletCoun
t);
                }

                if (aftertime == 50)
                {
                    generatePowerUp(pillX,pillY );
                    addPowerUp(px ,py, pillX, pillY,Health);
```

```
                    aftertime = 0;
                }
                aftertime++;
                /*PLAYER MOVEMENTS*/

                if (GetAsyncKeyState(VK_LEFT))
                {
                    playerLeft(px ,py);
                }

                if (GetAsyncKeyState(VK_RIGHT))
                {
                    playerRight(px ,py);
                }

                if (GetAsyncKeyState(VK_DOWN))
                {
                    playerDown(px ,py);
                }

                if (GetAsyncKeyState(VK_UP))
                {
                    playerUp(px ,py);
                }
                if (GetAsyncKeyState(VK_SPACE))
                {

                    generateBullet(px,py,bx,by,bulletCount);
                }

                moveBullet(bx,by,bulletCount);

                /*COLLOSION CHECKS*/
                collosion(e1x,e1y,bx,by,bulletCount,score, count ,enemy1Active,
count1);
                collosion2(e2x ,e2y,bx,by,bulletCount,score,count,enemy2Active
,count2 );
                enemyCollosion2(px
,py,Health,enemy2BulletX,enemy2BulletY,enemy2BulletCount);
                enemyCollosion(px
,py,Enemy1BulletX,Enemy1BulletY,Enemy1BulletCount,Health);

                Sleep(60);
```

```
                /*WIN AND LOSE CONDITIONS*/
                if (checkScore(score) == true && checkHealth(Health) != true)
                {
                    flag1 = true;
                }
                else if (checkScore(score) != true && checkHealth(Health) ==
true)
                {
                    flag2 = true;
                }
                if (flag1 == true)
                {
                    system("cls");
                    win();
                    getch();
                    break;

                }
                else if (flag2 == true)
                {
                    system("cls");
                    loss();
                    getch();
                    break;

                }
            }
        }
        if (option1 == "2")
        {
            system("cls");
            printHeader();
            inst();
            getch();
            break;
        }
        if (option1 == "3")
        {
            break;
        }
    }
}

void printHeader()
```

```
{

    setColor(4);
    cout << endl
         << endl;
    cout << "\t\t\t\tW   W  AAAAA  RRRR    " << endl;
    cout << "\t\t\t\tW   W A     A R   R   " << endl;
    cout << "\t\t\t\tW W W AAAAAAA RRRR    " << endl;
    cout << "\t\t\t\t W W  A     A R  R    " << endl;
    cout << "\t\t\t\t  W   A     A R   R   " << endl;
    cout << endl
         << endl;

    cout << "\t\t\t\tT T T T H   H  U   U  N   N  DDDD   EEEEE  RRRR   " << endl;
    cout << "\t\t\t\t   T    H   H  U   U  NN  N  D   D  E      R   R  " << endl;
    cout << "\t\t\t\t   T    HHHHH  U   U  N N N  D   D  EEEE   RRRR   " << endl;
    cout << "\t\t\t\t   T    H   H  U   U  N  NN  D   D  E      R  R   " << endl;
    cout << "\t\t\t\t   T    H   H   UUU   N   N  DDDD   EEEEE  R   R  " << endl;
    cout << endl
         << endl;
    setColor(6);
}
void printMaze()
{

    cout <<
"##############################################################################
########################################" << endl;
    cout <<
"#
                                      #" << endl;
    cout <<
"#
                                      #" << endl;
    cout <<
"#
                                      #" << endl;
    cout <<
"#
                                      #" << endl;
    cout <<
"#
                                      #" << endl;
```

```
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
    cout <<
"#
                                    #" << endl;
```

```
    cout <<
"#
                                        #" << endl;
    cout <<
"#
                                        #" << endl;
    cout <<
"#
                                        #" << endl;
    cout <<
"#
                                        #" << endl;
    cout <<
"#
                                        #" << endl;
    cout <<
"#
                                        #" << endl;
    cout <<
"##########################################################################
#####################################" << endl;
}
void setColor(int color)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hConsole, color);
}
void gotoxy(int x, int y)
{
    COORD coordinates;
    coordinates.X = x;
    coordinates.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinates);
}
char getCharATXY(short int x, short int y)
{

    CHAR_INFO ci;
    COORD xy = {0, 0};
    SMALL_RECT rect = {x, y, x, y};
    COORD coordBufsize;
    coordBufsize.X = 1;
    coordBufsize.Y = 1;
```

```
    return ReadConsoleOutput(GetStdHandle(STD_OUTPUT_HANDLE), &ci, coordBufsize,
xy, &rect) ? ci.Char.AsciiChar : ' ';
}

string Menu()
{
    string option = "0";
    cout << endl
         << endl;
    cout << "\t1.Play Game" << endl;
    cout << "\t2.Instructions" << endl;
    cout << "\t3.Exit" << endl;
    cout << "Enter your option...";
    // while(!(cin>>option))
    // {
    //     cout<<"Enter your option...";
    //     cin.clear();
    //     cin.ignore();
    // }
    cin >> option;
    cout << endl;
    option = validcheck(option);
    return option;
}
string validcheck(string option)
{
    while (true)
    {
        bool gun = true;    // 'gun' is a boolean variable used to track if the
current 'option' is valid.

        // Loop through each character in the 'option' string.
        for (int i = 0; i < option.length(); i++)
        {
            // Check if the character is not in the range '1' to '3'.
            if (option[i] < 49 || option[i] > 51)
            {
                gun = false;
                cout << "Invalid Option..." << endl;
                break;
            }
        }
        // If 'gun' is still true, all characters in 'option' are valid, so
return the 'option'.
```

```
        if (gun)
        {
            return option;
        }
        // If 'gun' is false, enter a new 'option' and continue the loop.
        cin >> option;
    }
}
// This function prints the player on the screen at the specified position (px,
py).
void printPlayer(int &px ,int &py)
{
    setColor(7);    // Set the console text color
    // char mmm = 219;  // Character used to represent the player
    char player[5][3] = {
        {'+', ' '   },
        {'+', '+'    },
        {'+', '+','+'},
        {'+', '+'    },
        {'+', ' '   },
    };
    // Loop through the player array and print it on the screen
    for (int i = 0; i < 5; i++)
    {
        gotoxy(px, py + i);
        for (int j = 0; j < 3; j++)
        {
            cout << player[i][j];
        }
        cout << endl;
    }
    setColor(7);  // Reset the console text color
}
// This function erases the player from the screen at the specified position (px,
py).
void erasePlayer(int &px, int &py)
{

    char erasePlayer[5][3] = {
        {' ', ' '   },
        {' ', ' '    },
        {' ', ' ', ' '},
        {' ', ' '    },
        {' ', ' '   },
```

```
    };
        // Loop through the erasePlayer array and print it on the screen to erase
the player
    for (int i = 0; i < 5; i++)
    {
        gotoxy(px, py + i);
        for (int j = 0; j < 3; j++)
        {
            cout << erasePlayer[i][j];
        }
        cout << endl;
    }
}
/*MOVING PLAYER FUNCTION*/
// move player up
void playerUp(int &px ,int &py)
{
    // Check the character at the next location above the player.
    char nextLocation = getCharATXY(px, py - 1);
    // If the next location is empty, erase the player, move it up, and print the
updated player.
    if ((nextLocation == ' '))
    {
       erasePlayer(px, py);
        py = py - 1; // moving player up
        printPlayer(px,py);
    }
}
// move player down
void playerDown(int &px ,int &py)
{
    // Check the character at the next location below the player.
    char nextLocation = getCharATXY(px, py + 6);
    // If the next location is empty or contains a dash ('-'), erase the player,
move it down, and print the updated player.
    if ((nextLocation == ' ') || (nextLocation == '-'))
    {
        erasePlayer(px, py);
        py = py + 1; // moving player down
        printPlayer(px, py);
    }
}
// move player left
void playerLeft(int &px ,int &py)
```

```
{
    // Check the character at the next location to the left of the player.
    char nextLocation = getCharATXY(px - 1, py);
    // If the next location is empty, erase the player, move it left, and print
the updated player.
    if (nextLocation == ' ')
    {
        erasePlayer(px, py);
        px = px - 1;
        printPlayer(px, py);
    }
}
// move player right
void playerRight(int &px ,int &py)
{
    // Check the character at the next location to the right of the player.
    char nextLocation = getCharATXY(px + 20, py);
    // If the next location is empty or contains a dash ('-'), erase the player,
move it right, and print the updated player.
    if (nextLocation == ' ' || nextLocation == '-')
    {
        erasePlayer(px, py);
        px = px + 1; // moving player right
        printPlayer(px, py);
    }
}
// Printing bullet
void printBullet(int x, int y)
{
    gotoxy(x, y);
    cout << "-";
}
// Erasing bullet
void eraseBullet(int x, int y)
{
    gotoxy(x, y);
    cout << " ";
}
// Generating bullets  at the player's position and updates the bullet array and
count.
void generateBullet(int &px ,int &py,int bx[],int by[],int &bulletCount)
{
    bx[bulletCount] = px + 3;
    by[bulletCount] = py + 2;
```

```
    gotoxy(px + 3, py + 2);
    cout << "-";
    bulletCount++;
}
// Removing bullets from array
void removeBullet(int x,int bx[],int by[],int &bulletCount)
{
    for (int i = 0; i < bulletCount - 1; i++)
    {
        bx[i] = bx[i + 1];
        by[i] = by[i + 1];
    }
    bulletCount--;
}
// Moving bullets on the screen and erase bullet if it hits.
void moveBullet(int bx[],int by[],int &bulletCount)
{
    for (int i = 0; i < bulletCount; i++)
    {
        char nextLocation = getCharATXY(bx[i] + 1, by[i] + 1);
        if (nextLocation != ' ')
        {
            eraseBullet(bx[i], by[i]);
            removeBullet(i,bx,by,bulletCount);
        }
        else
        {
            eraseBullet(bx[i], by[i]);
            bx[i] = bx[i] + 1;
            printBullet(bx[i], by[i]);
        }
    }
}

//  1 print enemy1
void printenemy1(int &e1x, int &e1y)
{
    setColor(6);   // Set the console text color
    char enemy1[3][3] = {{' ', ' ', '*'},
                         {'*', '*', '*'}, // initializating the array
                         {' ', ' ', '*'}};

    // Loop through the enemy1 array and print it on the screen
    for (int i = 0; i < 3; i++)
```

```
    {
        gotoxy(e1x, e1y + i);
        for (int j = 0; j < 3; j++)
        {
            cout << enemy1[i][j];
        }
        cout << endl;
    }
    setColor(5);
}
//  2 erasing enemy1
void eraseenemy1(int &e1x, int &e1y)
{
    char Enemy1Erase[3][3] = {{' ', ' ', ' '},
                              {' ', ' ', ' '},
                              {' ', ' ', ' '}};
    // Loop through the Enemy1Erase array
    for (int i = 0; i < 3; i++)
    {
        gotoxy(e1x, e1y + i);
        for (int j = 0; j < 3; j++)
        {
            cout << Enemy1Erase[i][j];
        }
        cout << endl;
    }
}
//  3  Moving left first
void moveEnemy1Left(int &e1x , int &e1y,string &direction,char &previousLoc)
{
    char nextLocation = getCharATXY(e1x - 5, e1y);

    if (nextLocation == '#')  // Change direction if a wall ('|') is encountered
    {
        direction = "right";
    }
    else if (e1x == 80)// Change direction if the enemy reaches it
    {
        direction = "up";
    }
    else if (nextLocation == ' ' || nextLocation == '-')
    {
        eraseenemy1(e1x, e1y);
        e1x = e1x - 1;
```

```
        previousLoc = nextLocation;
        printenemy1(e1x, e1y);
    }
}
//  4 Moving enemy1 to left again
void moveEnemy1Right(int &e1x , int &e1y,string &direction,char &previousLoc)
{
    char nextLocation = getCharATXY(e1x - 5, e1y);
    if (nextLocation == '#')// Change direction if a wall ('|') is encountered
    {
        direction = "up";
    }
    else if (nextLocation == ' ' || nextLocation == '-')
    {
        eraseenemy1(e1x, e1y);
        e1x = e1x - 1;
        previousLoc = nextLocation;
        printenemy1(e1x, e1y);
    }
}

//  5 enemy1 up movement
void moveEnemy1Up(int &e1x , int &e1y,string &direction,char &previousLoc)
{
    char nextLocation = getCharATXY(e1x - 5, e1y + 3);

    if (e1y == 5) // Change direction if the enemy reaches it
    {
        direction = "right";
    }

    else if (nextLocation == ' ' || nextLocation == '-')
    {
        eraseenemy1(e1x, e1y);
        e1y = e1y - 1;
        previousLoc = nextLocation;
        printenemy1(e1x, e1y);
    }
}
//  6 printing bullet
void Enemy1printBullet(int x, int y)
{
    gotoxy(x, y + 1);
    cout << "-" << endl;
```

```
}
// // 7 erasing bullet
void enemy1BulletErase(int x, int y)
{
    gotoxy(x, y + 1);
    cout << " " << endl;
}
// 8 generate bullet for enemy1
void generateEnemy1Bullet(int &e1x , int &e1y,int Enemy1BulletX[],int
Enemy1BulletY[],int &Enemy1BulletCount)
{
    Enemy1BulletX[Enemy1BulletCount] = e1x - 1;
    Enemy1BulletY[Enemy1BulletCount] = e1y;

    Enemy1BulletCount++;
}
// 9 removing bullet of enemy1
void removeEnemy1Bullet(int x,int Enemy1BulletX[],int Enemy1BulletY[],int
&Enemy1BulletCount)
{
    // Loop through the array starting from the given index 'x'
    for (int i = x; i < Enemy1BulletCount - 1; i++)
    {
        // Shift the elements to the left to fill the gap created by removing the
bullet at index 'x'
        Enemy1BulletX[i] = Enemy1BulletX[i + 1];
        Enemy1BulletY[i] = Enemy1BulletY[i + 1];
    }
        // Decrease the bullet count since one bullet has been removed
    Enemy1BulletCount--;
}
// 10 Moving the enemy1 bullet if it hits then it will erase bullet
void moveEnemy1Bullet(int Enemy1BulletX[],int Enemy1BulletY[],int
&Enemy1BulletCount)
{
    for (int i = 0; i < Enemy1BulletCount; i++)
    {
        // Check the character at the position where the bullet is about to move
        char nextLocation = getCharATXY(Enemy1BulletX[i] - 1, Enemy1BulletY[i]);
        if (nextLocation != ' ') // if it hits then erase bullet
        {
            enemy1BulletErase(Enemy1BulletX[i], Enemy1BulletY[i]);
            removeEnemy1Bullet(i,Enemy1BulletX,Enemy1BulletY,Enemy1BulletCount);
        }
```

```
        else if (nextLocation == ' ') // if its space then it first erase bullet
then move and print the bullet
        {
            enemy1BulletErase(Enemy1BulletX[i], Enemy1BulletY[i]);
            Enemy1BulletX[i] = Enemy1BulletX[i] - 1;
            Enemy1printBullet(Enemy1BulletX[i], Enemy1BulletY[i]);
        }
    }
}

// Enemy2Functions //

// 1 print enemy2
void printenemy2(int &e2x ,int &e2y)
{
    setColor(6);  // Set the console text color
    char enemy2[3][3] = {{' ', ' ', '*'},
                         {'*', '*', '*'}, // initializating the array
                         {' ', ' ', '*'}};
    // Loop through the enemy2 array and print it on the screen
    for (int i = 0; i < 3; i++)
    {
        gotoxy(e2x, e2y + i);
        for (int j = 0; j < 3; j++)
        {
            cout << enemy2[i][j];
        }
        cout << endl;
    }
    setColor(7);
}
// 2 erase enemy2
void eraseenemy2(int &e2x ,int &e2y)
{
    setColor(5);
    char Eraseenemy2[3][3] = {{' ', ' ', ' '},
                              {' ', ' ', ' '}, // initializating the array
                              {' ', ' ', ' '}};
    // Loop through the Enemy2Erase array
    for (int i = 0; i < 3; i++)
    {
        gotoxy(e2x, e2y + i);
        for (int j = 0; j < 3; j++)
        {
```

```
            cout << Eraseenemy2[i][j];
        }
        cout << endl;
    }
    setColor(9);
}
// 3 enemy2 movement Left
void moveEnemy2Left(int &e2x ,int &e2y,string &direction2,char &previousLoc)
{
    char nextLocation = getCharATXY(e2x - 5, e2y);
    if (nextLocation == '|') // Change direction if a wall ('|') is encountered
    {
        direction2 = "right";
    }
    else if (e2x == 100)
    {
        direction2 = "down";
    }
    else if (nextLocation == ' ' || nextLocation == '-')
    {
        eraseenemy2(e2x, e2y);
        e2x = e2x - 1;
        previousLoc = nextLocation;
        printenemy2(e2x, e2y);
    }
}
// 4 enemy2 movement Right
void moveEnemy2Right(int &e2x ,int &e2y,string &direction2,char &previousLoc)
{
    char nextLocation = getCharATXY(e2x - 5, e2y - 4);
    if (nextLocation == '#')// Change direction if a wall ('#') is encountered
    {
        direction2 = "down";
    }
    else if (nextLocation == ' ' || nextLocation == '-')
    {
        eraseenemy2(e2x, e2y);
        e2y = e2y - 1;
        previousLoc = nextLocation;
        printenemy2(e2x, e2y);
    }
}
// 5 enemy2 movement down
void moveEnemy2Down(int &e2x ,int &e2y,string &direction2,char &previousLoc)
```

```
{
    char nextLocation = getCharATXY(e2x - 5, e2y + 3);
    if (e2y == 18)// Change direction if the enemy reaches it
    {
        direction2 = "right";
    }
    else if (nextLocation == ' ' || nextLocation == '-')
    {
        eraseenemy2(e2x, e2y);
        e2y = e2y + 1;
        previousLoc = nextLocation;
        printenemy2(e2x, e2y);
    }
}
// 6 print enemy2 bullet
void Enemy2printBullet(int x, int y)
{
    gotoxy(x, y + 1);
    cout << "." << endl;
}
// 7 erasing bullet
void enemy2BulletErase(int x, int y)
{
    gotoxy(x, y + 1);
    cout << " " << endl;
}
//  8 generating bullets for enemy2 from array and updates the bullet count.
void generateEnemy2Bullet(int &e2x ,int &e2y,int enemy2BulletX[],int
enemy2BulletY[],int &enemy2BulletCount)
{
    enemy2BulletX[enemy2BulletCount] = e2x - 1;
    enemy2BulletY[enemy2BulletCount] = e2y;

    enemy2BulletCount++;
}
//  9 removing from arrays
void removeEnemy2Bullet(int x,int enemy2BulletX[],int enemy2BulletY[],int
&enemy2BulletCount)
{
    // Loop through the array starting from the given index 'x'
    for (int i = x; i < enemy2BulletCount - 1; i++)
    {
        // Shift the elements to the left to fill the gap created by removing the
bullet at index 'x'
```

```
        enemy2BulletX[i] = enemy2BulletX[i + 1];
        enemy2BulletY[i] = enemy2BulletY[i + 1];
    }
    enemy2BulletCount--;
}
// 10 Moving bullets for enemy2 ,erases bullet if it hits any object
void moveEnemy2Bullet(int enemy2BulletX[],int enemy2BulletY[],int
&enemy2BulletCount)
{
    for (int i = 0; i < enemy2BulletCount; i++)
    {
        char nextLocation = getCharATXY(enemy2BulletX[i] - 1, enemy2BulletY[i]);
        if (nextLocation != ' ') // if it hits then erase bullet
        {
            enemy2BulletErase(enemy2BulletX[i], enemy2BulletY[i]);
            removeEnemy2Bullet(i,enemy2BulletX, enemy2BulletY,enemy2BulletCount);
        }
        else
        {
            enemy2BulletErase(enemy2BulletX[i], enemy2BulletY[i]);
            enemy2BulletX[i] = enemy2BulletX[i] - 1;
            Enemy2printBullet(enemy2BulletX[i], enemy2BulletY[i]);
        }
    }
}
/*Collision Checks*/

// checking player bullet collision with enemy1
void collosion(int &e1x , int &e1y,int bx[],int by[],int &bulletCount,int
&score,int &count,string &enemy1Active,int &count1)
{
    // loop through each player bullet
    for (int i = 0; i < bulletCount; i++)
    {
        // Check if the bullet position coincides with the first enemy position
        if ((bx[i] + 1 == e1x || bx[i] + 1 == e1x + 2 || bx[i] + 3 == e1x) &&
(by[i] == e1y || by[i] == e1y + 1 || by[i] == e1y + 2 || by[i] == e1y + 3))
        {
            if (enemy1Active == "true")
            {
                addScore(score);
                eraseBullet(bx[i], by[i]);   //erase bullet if it hit enemy
                removeBullet(i,bx,by,bulletCount);
            }
```

```
            // Increment the count of hits on the first enemy
            count1 = count1 + 1;
            // If the first enemy has been hit 10 times, deactivate it and update
the count
            if (count1 == 15)
            {
                enemy1Active = "false";
                eraseenemy1(e1x, e1y);

            }
        }
    }
}
// checking player bullet collision with enemy2
void collosion2(int &e2x ,int &e2y,int bx[],int by[],int &bulletCount,int
&score,int &count,string &enemy2Active ,int &count2 )
{
    // loop through each player bullet
    for (int i = 0; i < bulletCount; i++)
    {
        // Check if the bullet position coincides with the second enemy position
        if ((bx[i] + 1 == e2x || bx[i] + 1 == e2x + 2 || bx[i] + 3 == e2x) &&
(by[i] == e2y || by[i] == e2y + 1 || by[i] == e2y + 2 || by[i] == e2y + 3))
        {
            if (enemy2Active == "true")
            {
                addScore(score);
                eraseBullet(bx[i], by[i]);// Erase the bullet that hit the enemy
and remove it from the array
                removeBullet(i,bx,by,bulletCount);
                count2++;
            }
            // If the second enemy has been hit 20 times, deactivate it and
update the count
            if (count2 == 18)
            {
                enemy2Active = "false";
                eraseenemy2(e2x, e2y);
                eraseBullet(bx[i], by[i]);
            }
        }
    }
}
// collison check for enemy1 bullets with player
```

```
void enemyCollosion(int &px ,int &py,int Enemy1BulletX[],int Enemy1BulletY[],int
&Enemy1BulletCount,int &Health)
{
    // loop through each bullet fired by the first enemy
    for (int i = 0; i < Enemy1BulletCount; i++)
    {
        // Check if the bullet position coincides with the player position
        if ((Enemy1BulletX[i] + 1 == px + 5 || Enemy1BulletX[i] + 1 == px + 8 ||
Enemy1BulletX[i] + 1 == px + 9) &&
            (Enemy1BulletY[i] + 1 == py + 3 || Enemy1BulletY[i] + 1 == py + 2 ||
Enemy1BulletY[i] + 1 == py + 1 || Enemy1BulletY[i] + 1 == py || Enemy1BulletY[i]
+ 1 == py - 1 || Enemy1BulletY[i] + 1 == py - 2 || Enemy1BulletY[i] + 1 == py -
3))
        {
            minusScore(Health); // Decrease player health
            enemy1BulletErase(Enemy1BulletX[i], Enemy1BulletY[i]);// Erase the
bullet that hit the player and remove it from the array
            removeEnemy1Bullet(i,Enemy1BulletX,Enemy1BulletY,Enemy1BulletCount);
        }
    }
}
// collison check for enemy2 bullets with player
void enemyCollosion2(int &px ,int &py,int &Health,int enemy2BulletX[],int
enemy2BulletY[],int &enemy2BulletCount)
{
    // loop through each bullet fired by the second enemy
    for (int i = 0; i < enemy2BulletCount; i++)
    {
        // Check if the bullet position coincides with the player position
        if ((enemy2BulletX[i] + 1 == px + 5 || enemy2BulletX[i] + 1 == px + 8 ||
enemy2BulletX[i] + 1 == px + 9) &&
            (enemy2BulletY[i] + 1 == py + 3 || enemy2BulletY[i] + 1 == py + 2 ||
enemy2BulletY[i] + 1 == py + 1 || enemy2BulletY[i] + 1 == py || enemy2BulletY[i]
+ 1 == py - 1 || enemy2BulletY[i] + 1 == py - 2 || enemy2BulletY[i] + 1 == py -
3))
        {
            minusScore(Health);// Decrease player health
            // Erase the bullet that hit the player and remove it from the array
            enemy2BulletErase(enemy2BulletX[i], enemy2BulletY[i]);
            removeEnemy2Bullet(i,enemy2BulletX,enemy2BulletY,enemy2BulletCount);
        }
    }
}
```

```
// check scores and health
bool checkScore(int score)
{
    if (score >= 150)
    {
        return true;
    }
    else
    {
        return false;
    }
}
bool checkHealth(int health)
{
    if (health <= 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
/*POWER UP FUNCTIONS*/
void addPowerUp(int &px ,int &py,int &pillX ,int &pillY ,int &Health)
{

    // Check if the player position coincides with pills
    if (px == pillX || px + 1 == pillX || px + 3 == pillX || px + 5 == pillX ||
px + 7 == pillX || px + 9 == pillX || px + 11 == pillX || px + 13 == pillX || px
+ 16 == pillX)
    {
        // Increase player health by 10
        Health = Health + 10;
    }
}
void generatePowerUp(int &pillX ,int &pillY )
{

    char heart = 'O';
    gotoxy(pillX ,pillY);
    cout << heart;  // print thr heart
}
void removepowerUp(int &Health)
```

```
{
    cout << " ";
}
// Scoring functions
void addScore(int &score )
{
    score = score + 7;  // increasing score
}
void minusScore(int &Health)
{
    Health = Health - 8; // decreasing score
}
void showScore(int playerScore, int health)
{
    gotoxy(1, 27);
    cout << "score: " << playerScore << " ";
    gotoxy(1, 28);
    cout << "Health: " << health << " ";
}
void win()
{
    setColor(2);
    cout << endl
         << endl;
    cout << "You Won" << endl;
    setColor(7);
}
void loss()
{
    setColor(4);
    cout << endl
         << endl;
    cout << "You Loss" << endl;
    setColor(7);
}
void inst()
{

    cout <<"\tInstructions"<<endl;
    cout << "\t\tLeft arrow >> left movement" << endl;
    cout << "\t\tRight arrow >> Right movement" << endl;
    cout << "\t\tArrow Down  >> down movement" << endl;
    cout << "\t\tArrow Up >> up movement" << endl;
    cout << "\t\tSpscebar >> Fire" << endl;
```

```
    cout <<endl;
    cout << "\tREGULATIONS:" << endl;
    cout << "\t\tFor hitting bullet to enemy you will get 7 score" << endl;
    cout << "\t\t if enemy hits you your health decreases"<<endl;
    cout << "\t\tIf the enemy bullet hits you , your health will be decremented"
<< endl;
    cout << "PRESS ANY KEY TO CONTINUE" << endl;
}
```