SURVIVAL RUN

Session: 2022 – 2026

Submitted by:

Afeera Fatima 2022-CS-151

Supervised by:

Dr. Awais Hassan

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Contents

1. Description and Story Writing of Game

In this exciting prison escape game, the player takes on the role of a prisoner determined to escape from his jail cell. The only way to do so is by collecting keys scattered throughout the prison.

As the player collects more keys, he can navigate to new areas of the prison that were previously inaccessible. However, he must also avoid the police, who are moving randomly throughout the game and will chase him down when he gets too close.

To make things more challenging, the police are armed and will fire on the player if they get too close. Luckily, the player has a gun that can give the police an electric shock, temporarily stunning them and giving the player a chance to escape. But using the gun takes a toll on the player's health, so he must use it wisely.

With multiple levels, unpredictable police movements, and new areas to explore, this game is sure to keep players engaged and challenged.

2. Game Characters Description

2.1 Player

There is one human player in the game.

2.1.1 Jeff:

He is the main character of the game who is running for his life. He can attack police with electric shock but that will only stop them for few seconds and again they will attack. So he can only save his life by escaping from the prison.

2.2 Enemies

2.2.1 Commando:

The guard is always on high alert and will not hesitate to use his weapon if he detects player’s presence. He is always supervising the player and preventing him from escaping.

2.2.2 Guards:

The guards in the game are known for their unpredictable movement patterns, which **make** it difficult for the **player character \ to trick** them. The guards are also skilled at coordinating with each other to trap Jeff and prevent his escape**.**

2.2.3 Jailer:

He is known for his patrolling in the game. He will only attack player if he collides with him, but will give chase if Jeff attempts to escape.

3. Game Objects Description

3.1 Keys:

T**hese** are used to open doors that provide access to new areas, where he can find additional keys to aid in his escape**.**

These are closed doors of the prison that player cannot cross without keys.

4. Rules and interactions

In the game, the player takes on the role of a prisoner who is attempting to escape from jail. The player must navigate the prison while avoiding guards who are patrolling the area. The guards move randomly but will give chase if they detect the player's presence, making it difficult for him to collect the keys he needs.

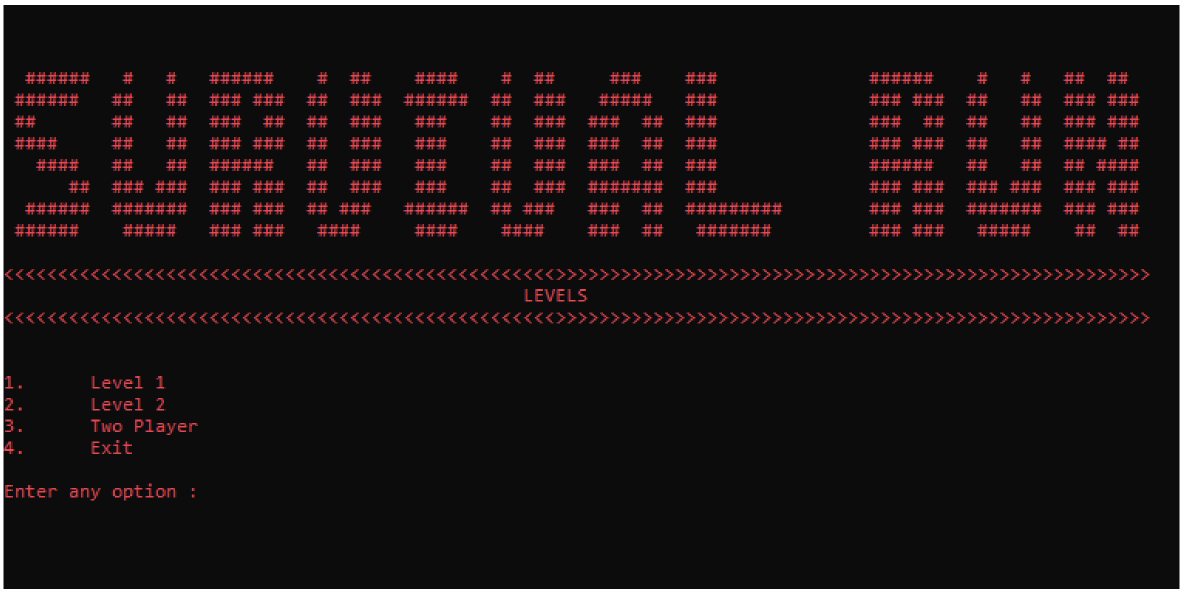
To temporarily avoid the enemies, the player has access to a stun gun that can shock them in four different directions. This will stop them for a few seconds, allowing the player to make a quick escape. However, the player's health will decrease if he is hit by a guard. The keys are used for access to different areas. Keys and the stun increase the player's score.

The game ends when the player reaches the escape door with keys or if he dies.

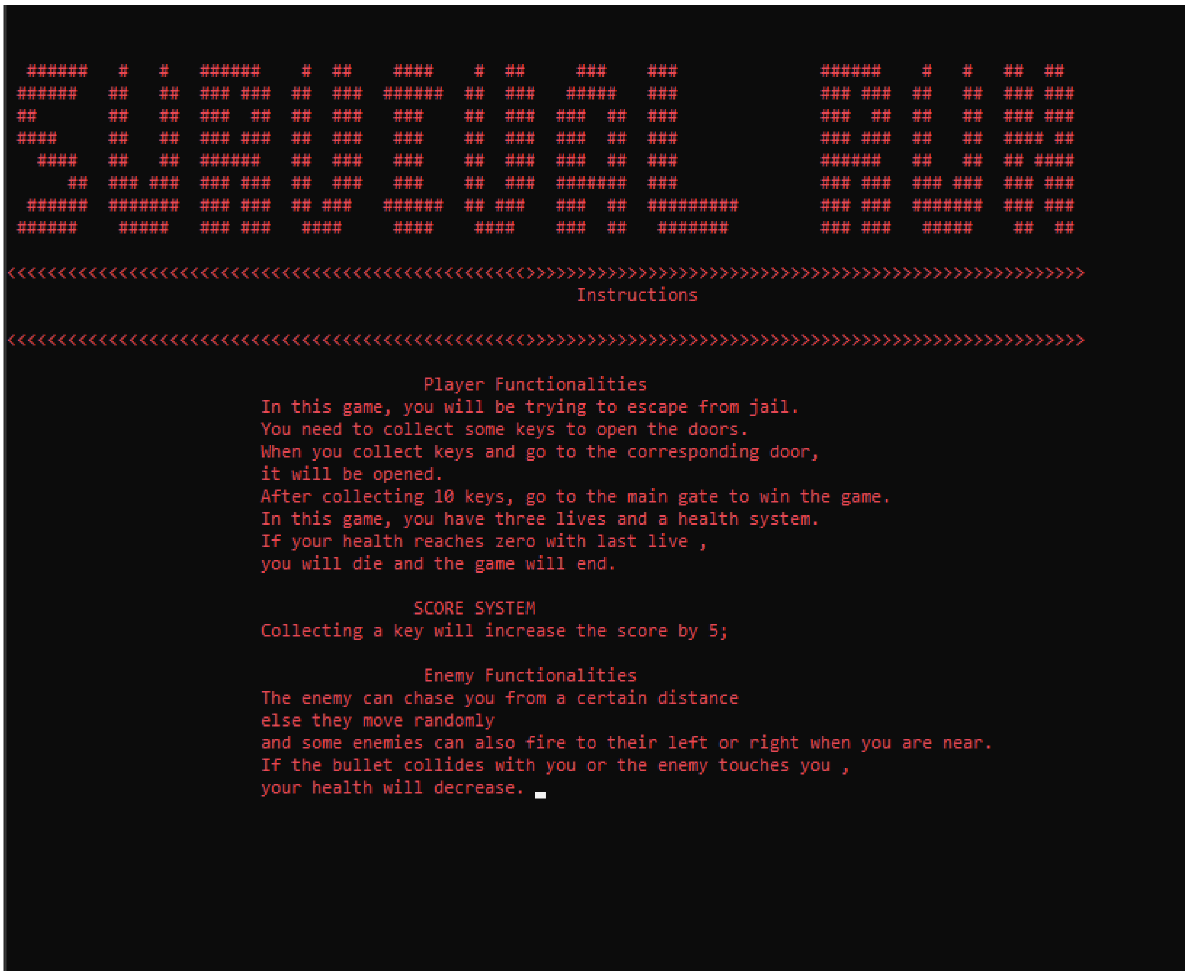
5. Goal of the Game

The goal of the game is to collect keys scattered throughout the prison that will ultimately unlock the escape door for player's escape.

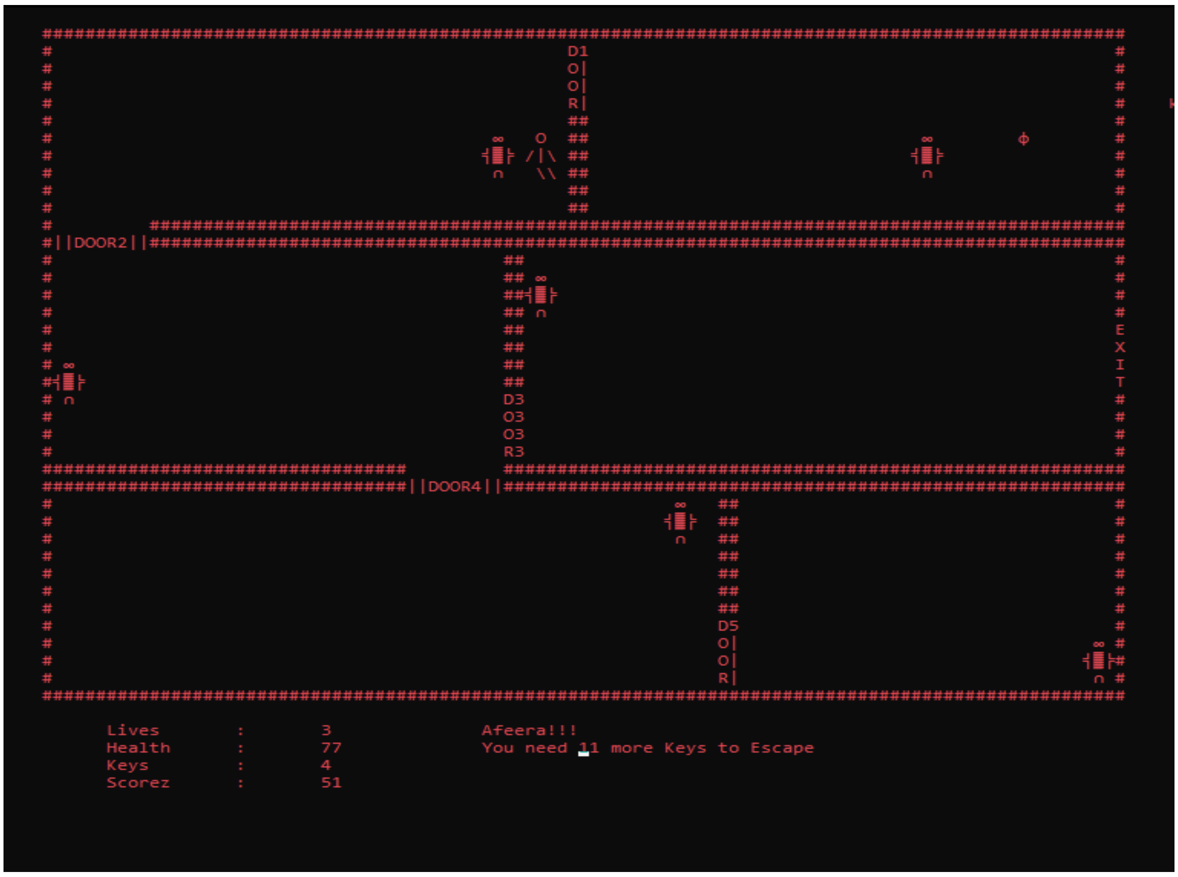
6. Wireframes of the Game



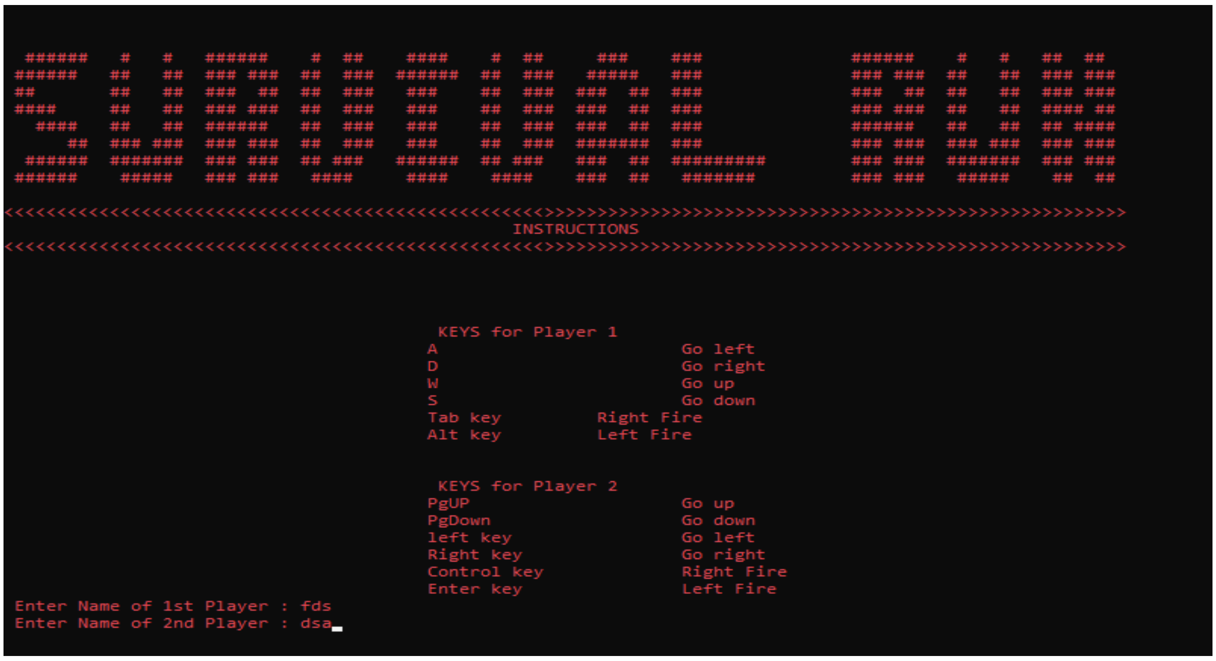
```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                 LEVELS
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>


1.      Level 1
2.      Level 2
3.      Two Player
4.      Exit

Enter any option :
```



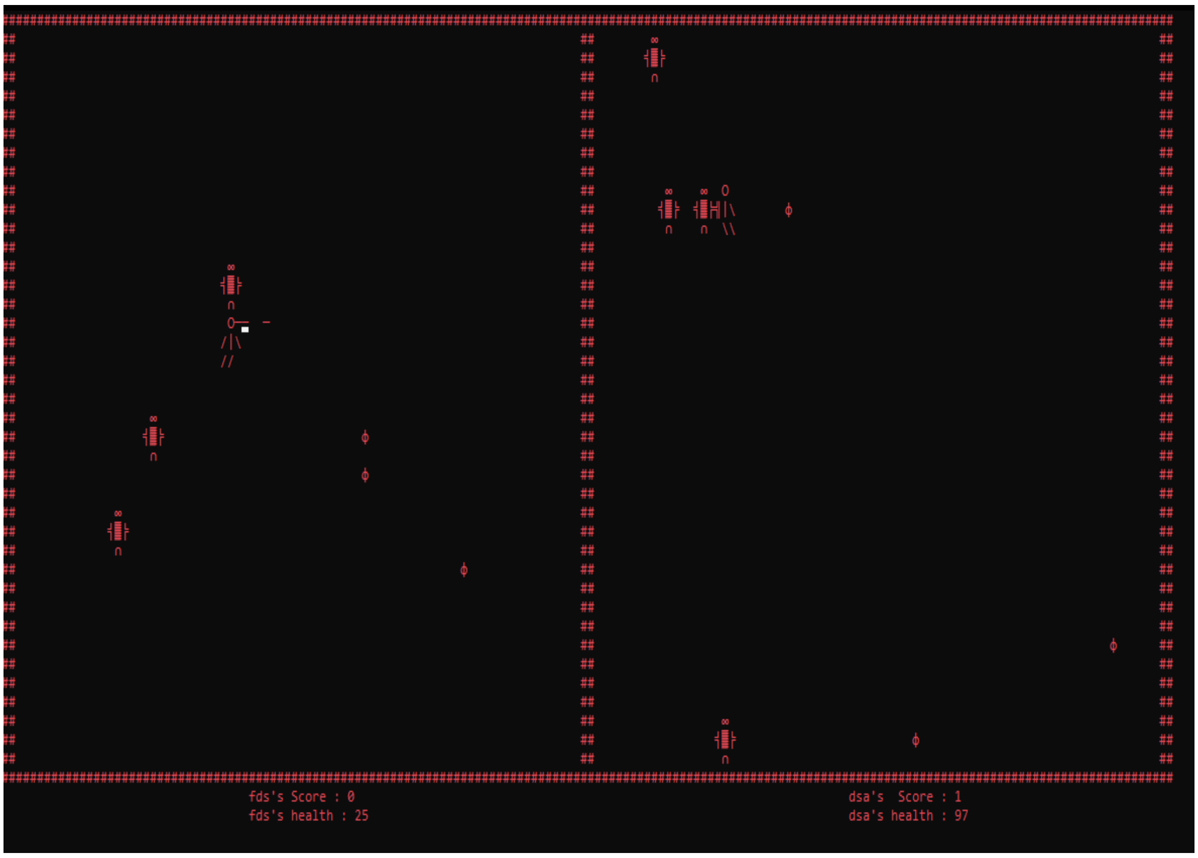
```
 ######   #   #   ######    #  ##    ####     #  ##     ###    ###             ######    #   #   ##  ##
######   ##   ##  ### ###  ##  ###  ######   ##  ###   #####   ###             ### ###  ##   ##  ### ###
##       ##   ##  ###  ##  ##  ###   ###     ##  ###  ###  ##  ###             ###  ##  ##   ##  ### ###
####     ##   ##  ### ###  ##  ###   ###     ##  ###  ###  ##  ###             ### ###  ##   ##  #### ##
  ####   ##   ##  ######   ##  ###   ###     ##  ###  ###  ##  ###             ######   ##   ##  ## ####
     ##  ### ###  ### ###  ##  ###   ###     ##  ###  #######  ###             ### ###  ### ###  ### ###
 ######  #######  ### ###  ## ###   ######   ## ###   ###  ##  #########       ### ###  #######  ### ###
######    #####   ### ###   ####     ####     ####    ###  ##   #######        ### ###   #####    ##   ##

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                        Instructions

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

                                         Player Functionalities
                         In this game, you will be trying to escape from jail.
                         You need to collect some keys to open the doors.
                         When you collect keys and go to the corresponding door,
                         it will be opened.
                         After collecting 10 keys, go to the main gate to win the game.
                         In this game, you have three lives and a health system.
                         If your health reaches zero with last live ,
                         you will die and the game will end.

                                        SCORE SYSTEM
                         Collecting a key will increase the score by 5;

                                         Enemy Functionalities
                         The enemy can chase you from a certain distance
                         else they move randomly
                         and some enemies can also fire to their left or right when you are near.
                         If the bullet collides with you or the enemy touches you ,
                         your health will decrease.
```



```
      Lives        :        3              Afeera!!!
      Health       :        77             You need 11 more Keys to Escape
      Keys         :        4
      Scorez       :        51
```



```
                                      INSTRUCTIONS


                         KEYS for Player 1
                        A                           Go left
                        D                           Go right
                        W                           Go up
                        S                           Go down
                        Tab key         Right Fire
                        Alt key         Left Fire


                         KEYS for Player 2
                        PgUP                        Go up
                        PgDown                      Go down
                        left key                    Go left
                        Right key                   Go right
                        Control key                 Right Fire
                        Enter key                   Left Fire
Enter Name of 1st Player : fds
Enter Name of 2nd Player : dsa
```



```
fds's Score : 0
fds's health : 25
```

```
dsa's  Score : 1
dsa's health : 97
```

7. Data Structure(2D Arrays)

```
string maze[37][88];
string player[3][3] = {
        {" ", "O", " "},
        {"/", "|", "\\"},
        {"/", "/", ""}};
char enemy[3][3] = {
        {' ', char(236), ' '},
        {char(181), char(178), char(198)},
        {' ', char(239), ' '}};
int EbulletR[100][2];
int EbulletL[100][2];
int PbulletU[100][2];
int PbulletD[100][2];
int PbulletR[100][2];
int PbulletL[100][2];
```

8. Function Prototypes

```
/*Virtual Keys*/
void leftkey(int jeffCoordinates[2], int &Keys, int &score);
void rightkey(int jeffCoordinates[2], int &Keys, int &score);
void upkey(int jeffCoordinates[2], int &Keys, int &score);
void downkey(int jeffCoordinates[2], int &Keys, int &score);
/*Erase and print jeff(player)*/
void eraseplayers(int Coordinates[2]); // erase both enemy and player
void printJeffLeft(int jeffCoordinates[2]);
void printJeffRight(int jeffCoordinates[2]);
void printJeffRightbulletfire(int jeffCoordinates[2]); // animations
void printJeffLeftbulletfire(int jeffCoordinates[2]);
void printJeffUpbulletfire(int jeffCoordinates[2]);
void printJeffDownbulletfire(int jeffCoordinates[2]);

void chaseEnemy(int enemyCoordinates[2], int jeffCoordinates[2], int &Health);
void random(int enemyCoordinates[2], int index);
bool CheckChase(int enemyCoordinates[2], int jeffCoordinates[2]);
bool EnemyCollisionwithBullet(int enemyCoordinates[2], int &Score);
// enemies
void printEnemy(int enemyCoordinates[2]);
void printenemyunderweb(int enemyCoordinates[2]); // prints enemy when bullet collides
char getCharAtxy(short int x, short int y);
void gotoxy(int x, int y);
// enemy bullet
void smartFire(int enemyCoordinates[2], int jeffCoordinates[2]);
void printBulletE(int x, int y);
void generateBulletRightE(int enemyCoordinates[2]);
void generateBulletLeftE(int enemyCoordinates[2]);
void moveBulletRightE(); // Enemy Bullet
void moveBulletLeftE();
void removeBulletFromArrayER(int index);
void removeBulletFromArrayEL(int index);
// Player Bullets
void generateBulletRight(int jeffCoordinates[2]); //  Generate bullet to right side of player
void generateBulletLeft(int jeffCoordinates[2]);  //  Generate bullet to left side of player
void generateBulletDown(int jeffCoordinates[2]);  //  Generate bullet to down side of player
void generateBulletUp(int jeffCoordinates[2]);    //  Generate bullet to up side of player

void removeBulletFromArrayR(int index); // removes bullet from array right to enemy
void removeBulletFromArrayL(int index); // removes bullet from array left to enemy
void removeBulletFromArrayD(int index); // removes bullet from array down to enemy
void removeBulletFromArrayU(int index); // removes bullet from array Up to enemy

void moveBulletRight(); // move player bullet to right side
void moveBulletLeft();  // move player bullet to left side
void moveBulletUp();    // move player bullet to up side
void moveBulletDown();  // move player bullet to down side

void printBullet(int x, int y); // print player bullet on desired location
void eraseBullet(int x, int y); // remove bullet from desired location

// all cout data
void loading(); // display
void header();  // top name Survival Run
string levelsMenu();
void printBox(int x, int y, int gotox, int gotoy);
void maze1();
void maze2();
void maze3(); // two player
void Level2();
void Level1();
void level2cout(int Keys, string name);
void keys2(int &keycount, int Keys);
void opendoors2(int jeffCoordinates[], int Keys);
void twoPlayer();
void ScoreBoard(int keys, int health, int lives, int score);
void playerHealth(int jeffCoordinates[2], int &Health);
void Instructions3();
void Instructions2();
void Instructions1();
bool nameCheck(string word);
// filehandling
string getfield(string line, int field);
bool nameExists(string name, string players[][7], int playercount);
int userindex(string name, string players[][7], int playerscount);
void removePlayerfromArray(int index, int &playercount, string players[][7]);
void storePlayerDataLevel1(string players[][7], int playercount);
void loadPlayerDataLevel1(string players[][7], int &playercount);
void storePlayerDataLevel2(string players[][7], int playercount);
void loadPlayerDataLevel2(string players[][7], int &playercount);
```

9. Complete Code

```
#include <iostream>
#include <windows.h>
#include <conio.h>
#include <fstream>
using namespace std;
HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);
char key = 237;
string random_Direction[6] = {"left", "right", "up", "left", "down", "right"};
/*Enemy Bullets*/
int EbulletR[100][2];
int ErightCount[100];
int E_bulletCountR = 0; // total enemys bullets to right side
int EbulletL[100][2];
int EleftCount[100];
int E_bulletCountL = 0; // total  enemy's bulletS to left side

/*Player Bullets*/
int PbulletU[100][2];
int PUpCount[100];      // to calculate total count each bullet is moved
int P_bulletCountU = 0; // total Player's bullets to Up

int PbulletD[100][2];
int PDownCount[100];    // total count of 1 bullet
int P_bulletCountD = 0; // total Player's bullets

int PbulletR[100][2];
int PrightCount[100];
int P_bulletCountR = 0; // total Player's bullets to right side

int PbulletL[100][2];
int PleftCount[100];
int P_bulletCountL = 0; // total  Player's bulletS to left side
/*Virtual Keys*/
void leftkey(int jeffCoordinates[2], int &Keys, int &score);
void rightkey(int jeffCoordinates[2], int &Keys, int &score);
void upkey(int jeffCoordinates[2], int &Keys, int &score);
void downkey(int jeffCoordinates[2], int &Keys, int &score);
/*Erase and print jeff(player)*/
void eraseplayers(int Coordinates[2]); // erase both enemy and player
void printJeffLeft(int jeffCoordinates[2]);
void printJeffRight(int jeffCoordinates[2]);
void printJeffRightbulletfire(int jeffCoordinates[2]); // animations
void printJeffLeftbulletfire(int jeffCoordinates[2]);
void printJeffUpbulletfire(int jeffCoordinates[2]);
void printJeffDownbulletfire(int jeffCoordinates[2]);
void chaseEnemy(int enemyCoordinates[2], int jeffCoordinates[2], int &Health);
void random(int enemyCoordinates[2], int index);
bool CheckChase(int enemyCoordinates[2], int jeffCoordinates[2]);
bool EnemyCollisionwithBullet(int enemyCoordinates[2], int &Score);
// enemies
void printEnemy(int enemyCoordinates[2]);
void printenemyunderweb(int enemyCoordinates[2]); // prints enemy when bullet collides
char getCharAtxy(short int x, short int y);
void gotoxy(int x, int y);
// enemy bullet
void smartFire(int enemyCoordinates[2], int jeffCoordinates[2]);
void printBulletE(int x, int y);
void generateBulletRightE(int enemyCoordinates[2]);
void generateBulletLeftE(int enemyCoordinates[2]);
void moveBulletRightE(); // Enemy Bullet
void moveBulletLeftE();
void removeBulletFromArrayER(int index);
void removeBulletFromArrayEL(int index);
// Player Bullets
void generateBulletRight(int jeffCoordinates[2]); //  Generate bullet to right side of player
void generateBulletLeft(int jeffCoordinates[2]);  //  Generate bullet to left side of player
void generateBulletDown(int jeffCoordinates[2]);  //  Generate bullet to down side of player
void generateBulletUp(int jeffCoordinates[2]);    //  Generate bullet to up side of player

void removeBulletFromArrayR(int index); // removes bullet from array right to enemy
void removeBulletFromArrayL(int index); // removes bullet from array left to enemy
void removeBulletFromArrayD(int index); // removes bullet from array down to enemy
void removeBulletFromArrayU(int index); // removes bullet from array Up to enemy

void moveBulletRight(); // move player bullet to right side
void moveBulletLeft();  // move player bullet to left side
void moveBulletUp();    // move player bullet to up side
void moveBulletDown();  // move player bullet to down side
void printBullet(int x, int y); // print player bullet on desired location
void eraseBullet(int x, int y); // remove bullet from desired location

// all cout data
void loading(); // display
void header();  // top name Survival Run
string levelsMenu();
void printBox(int x, int y, int gotox, int gotoy);
void maze1();
void maze2();
void maze3(); // two player
void Level2();
void Level1();
void level2cout(int Keys, string name);
void keys2(int &keycount, int Keys);
void opendoors2(int jeffCoordinates[], int Keys);
void twoPlayer();
void ScoreBoard(int keys, int health, int lives, int score);
void playerHealth(int jeffCoordinates[2], int &Health);
void Instructions3();
void Instructions2();
void Instructions1();
bool nameCheck(string word);
// filehandling
string getfield(string line, int field);
bool nameExists(string name, string players[][7], int playercount);
int userindex(string name, string players[][7], int playerscount);
void removePlayerfromArray(int index, int &playercount, string players[][7]);
void storePlayerDataLevel1(string players[][7], int playercount);
void loadPlayerDataLevel1(string players[][7], int &playercount);
void storePlayerDataLevel2(string players[][7], int playercount);
void loadPlayerDataLevel2(string players[][7], int &playercount);

main()
```

```
{
   system("cls");
   // // Instructions2();

   loading();
   while (true)
   {
      header();
      string option = levelsMenu();

      if (option == "1")
      {
         Instructions1();
         Level1();
      }
      else if (option == "2")
      {
         Instructions2();
         Level2();
      }
      else if (option == "3")
      {
         Instructions3();
         twoPlayer();
      }
      else if (option == "4")
      {
         break;
      }
   }
}
void opendoors2(int jeffCoordinates[], int Keys)
{
   if (Keys >= 2 && (jeffCoordinates[0] > 75 && jeffCoordinates[0] < 85) &&
(jeffCoordinates[1] > 1 && jeffCoordinates[1] < 7))
```

```
  {
     for (int i = 0; i < 4; i++)
     {
        gotoxy(79, 2 + i);
        cout << "  ";
     }
  }
  if (Keys >= 5 && (jeffCoordinates[0] > 31 && jeffCoordinates[0] < 36) && (jeffCoordinates[1] > 9 && jeffCoordinates[1] < 20))
  {
     gotoxy(31, 13);
     cout << "        ";
  }
  if (Keys >= 7 && (jeffCoordinates[0] > 68 && jeffCoordinates[0] < 82) && (jeffCoordinates[1] > 22 && jeffCoordinates[1] < 27))
  {
     for (int i = 0; i < 4; i++)
     {
        gotoxy(73, 22 + i);
        cout << "  ";
     }
  }
  if (Keys >= 10 && (jeffCoordinates[0] > 64 && jeffCoordinates[0] < 70) && (jeffCoordinates[1] > 22 && jeffCoordinates[1] < 32))
  {
     gotoxy(64, 27);
     cout << "        ";
  }
  if (Keys >= 13 && (jeffCoordinates[0] > 89 && jeffCoordinates[0] < 97) && (jeffCoordinates[1] > 35 && jeffCoordinates[1] < 39))
  {
     for (int i = 0; i < 4; i++)
     {
        gotoxy(93, 35 + i);
        cout << "  ";
        }
    }
}
void keys2(int &keycount, int Keys) // print keys
{
    gotoxy(135, 5);
    cout << "keycount" << keycount << endl;
    if (Keys < 2)
    {
        int x = (rand() % 50) + 30;
        int y = rand() % 12 + 1;

        char location = getCharAtxy(x, y);
        if (location == ' ' && keycount < 2)
        {
            gotoxy(x, y);
            cout << key;
            keycount++;
        }
    }
    else if (Keys >= 2 && Keys < 5)
    {
        cout << "gfhjd";
        int x = rand() % 103 + 30;
        int y = rand() % 12 + 1;
        if (x > 55)
        {
            char location = getCharAtxy(x, y);
            if (location == ' ' && keycount < 5)
            {
                cout << "gfhjd";
                gotoxy(x, y);
                cout << key;
                keycount++;
            }
        }
    }
    else if (Keys >= 5 && Keys < 7)
    {
        int x = (rand() % 25) + 30;
        int y = rand() % 25;
        if (y > 13)
        {
            char location = getCharAtxy(x, y);
            if (location == ' ' && keycount < 7)
            {
                gotoxy(x, y);
                cout << key;
                keycount++;
            }
        }
    }
    else if (Keys >= 7 && Keys < 10)
    {
        int x = (rand() % 103) + 30;
        int y = rand() % 25;
        if (y > 13 && x > 75)
        {
            char location = getCharAtxy(x, y);
            if (location == ' ' && keycount < 10)
            {
                gotoxy(x, y);
                cout << key;
                keycount++;
            }
        }
    }
    else if (Keys >= 10 && Keys <= 13)
    {
        int x = (rand() % 103) + 30;
        int y = rand() % 38;
        char location = getCharAtxy(x, y);
        if (y > 25 && x < 93)
        {

            if (location == ' ' && keycount < 15)
            {
                gotoxy(x, y);
                cout << key;
                keycount++;
            }
        }
    }
    else if (Keys > 13 && Keys <= 15)
    {
        int x = (rand() % 103) + 30;
        int y = rand() % 38;
        char location = getCharAtxy(x, y);
        if (y > 25 && x > 93)
        {

            if (location == ' ' && keycount < 15)
            {
                gotoxy(x, y);
                cout << key;
                keycount++;
            }
        }
    }
}
void maze1()
{
    string maze[37][88];
    fstream myfile;
    string line;
    int row = 0;
    myfile.open("maze1.txt", ios::in);
    while (getline(myfile, line))
    {
        for (int col = 0; col < 88; col++)
        {
            maze[row][col] = line[col];
        }
        row++;
    }
    myfile.close();
    for (int i = 0; i < 37; i++)
    {
        gotoxy(30, 2 + i);
        for (int j = 0; j < 88; j++)
        {
            cout << maze[i][j];
        }
        cout << endl;
    }
}
void maze2()
{
    string maze[39][102];
    fstream myfile;
    string line;
    int row = 0;
    myfile.open("maze2.txt", ios::in);
    while (getline(myfile, line))
    {
        for (int col = 0; col < 102; col++)
        {
            maze[row][col] = line[col];
        }
        row++;
    }
    myfile.close();
    for (int i = 0; i < 39; i++)
    {
        gotoxy(30, 1 + i);
        for (int j = 0; j < 102; j++)
        {
            cout << maze[i][j];
        }
        cout << endl;
    }
}
string getfield(string line, int field) // line read by load function and field is word separated by
comma's
{
    int commacount = 1;
    string item = "";
    for (int i = 0; i < line.length(); i++)
    {
        if (line[i] == ',')
        {
            commacount++;
        }
        else if (commacount == field)
        {
            item = item + line[i];
        }
    }
    return item;
}
void twoPlayer()
```

```
{
    int jeff1Coordinates[2] = {31, 15}; // left side player
    int jeff2Coordinates[2] = {100, 8};
    int enemy1Coordinates[2] = {30, 7}; // left side enemy
    int enemy2Coordinates[2] = {90, 8};
    int enemy3Coordinates[2] = {30, 15};
    int enemy4Coordinates[2] = {90, 15};
    int enemy5Coordinates[2] = {15, 15};
    int enemy6Coordinates[2] = {110, 15};
    int enemy7Coordinates[2] = {90, 15};
    int enemy8Coordinates[2] = {20, 15};
    string name1;
    string name2;
    int health2 = 100; // left side player
    int health1 = 100;
    int score2 = 0; // increment when bullet collides with enemy
    int score1 = 0;
    int keys2 = 0; // incerement when player picks food(key ascii)
    int keys1 = 0;
    int counter = 0;
    int enemy6counter = 50; // for enemy movement
    int enemy5counter = 50;
    int enemy4counter = 50; // for enemy movement
    int enemy3counter = 50;
    int enemy2counter = 50; // for enemy movement
    int enemy1counter = 50;
a:
    string name;
    cout << " Enter Name of 1st Player : ";
    cin >> name;
    bool validname1 = nameCheck(name);
    if (validname1)
    {
        name1 = name;
```

```
b:
    cout << " Enter Name of 2nd Player : ";
    cin >> name;
    bool validname2 = nameCheck(name);
    if (validname2)
    {
        name2 = name;
    }
    else
    {
        goto b;
    }
}
else
{
    goto a;
}
system("cls");
maze3();

while (health1 > -1 && health2 > -1)
{
    Sleep(1);
    counter++;
    enemy1counter++; // counter becomees zero when bullet collides with enemy
    enemy2counter++;
    enemy3counter++; // counter becomees zero when bullet collides with enemy
    enemy4counter++;
    moveBulletRightE(); // bullet movement for enemy
    moveBulletLeftE();
    moveBulletLeft(); // bullet for player
    moveBulletRight();
    /* player 2(right side)*/
    if (GetAsyncKeyState(VK_LEFT))
```

```
{
   leftkey(jeff2Coordinates, keys2, score2);
}
if (GetAsyncKeyState(VK_RIGHT))
{
   rightkey(jeff2Coordinates, keys2, score2);
}
if (GetAsyncKeyState(VK_UP))
{
   upkey(jeff2Coordinates, keys2, score2);
}
if (GetAsyncKeyState(VK_DOWN))
{
   downkey(jeff2Coordinates, keys2, score2);
}
if (GetAsyncKeyState(VK_CONTROL)) // player2 bullets
{
   printJeffRightbulletfire(jeff2Coordinates);
   generateBulletRight(jeff2Coordinates);
}
if (GetAsyncKeyState(VK_RETURN)) // Enter key
{
   printJeffLeftbulletfire(jeff2Coordinates);
   generateBulletLeft(jeff2Coordinates);
}
/*Player 1 (left Side) Movements*/
if (GetAsyncKeyState(0x44)) // D_key for right size movement
{
   rightkey(jeff1Coordinates, keys1, score1);
}
if (GetAsyncKeyState(0x41)) // A_key for left side movement
{
   leftkey(jeff1Coordinates, keys1, score1);
}
```

```
if (GetAsyncKeyState(0x57)) // W_key for up movement
{
   upkey(jeff1Coordinates, keys1, score1);
}
if (GetAsyncKeyState(0x53)) // S_key for down movement
{
   downkey(jeff1Coordinates, keys1, score1);
}
if (GetAsyncKeyState(VK_TAB)) // player 1 bullets
{
   printJeffRightbulletfire(jeff1Coordinates);
   generateBulletRight(jeff1Coordinates);
}
if (GetAsyncKeyState(VK_MENU)) // Alt key
{
   printJeffLeftbulletfire(jeff1Coordinates);
   generateBulletLeft(jeff1Coordinates);
}
if (counter == 10)
{
   // for random food(keys) in game
   int x = rand() % 160 + 1;
   int y = rand() % 39 + 1;
   char location = getCharAtxy(x, y);
   if (location == ' ')
   {
      gotoxy(x, y);
      cout << key;
   }
   counter = 0;
}
bool bulletcollision = EnemyCollisionwithBullet(enemy1Coordinates, score1); // checks
bullet collission with enemy
if (bulletcollision)
{
   enemy1counter = 0; // to stop enemy
}
bool bulletcollision2 = EnemyCollisionwithBullet(enemy2Coordinates, score2);
if (bulletcollision2) // checks bullet collission with enemy
{
   enemy2counter = 0; // to stop enemy
}
bool bulletcollision3 = EnemyCollisionwithBullet(enemy3Coordinates, score1); // checks
bullet collission with enemy
if (bulletcollision)
{
   enemy3counter = 0; // to stop enemy
}
bool bulletcollision4 = EnemyCollisionwithBullet(enemy4Coordinates, score2);
if (bulletcollision4) // checks bullet collission with enemy
{
   enemy4counter = 0; // to stop enemy
}
if (counter % 5 == 0)
{
   if (enemy1counter > 50)
   {
      chaseEnemy(enemy1Coordinates, jeff1Coordinates, health1); // enemy chasing
player
      smartFire(enemy1Coordinates, jeff1Coordinates);      // enemy fire when player is
near
   }
   if (enemy2counter > 50)
   {
      chaseEnemy(enemy2Coordinates, jeff2Coordinates, health2);
      smartFire(enemy2Coordinates, jeff2Coordinates); // enemy fire when player is near
   }
}
if (counter % 2 == 0)
{
   if (enemy3counter > 50)
   {
      bool chase3 = CheckChase(enemy3Coordinates, jeff1Coordinates);
      if (chase3)
      {
          chaseEnemy(enemy3Coordinates, jeff1Coordinates, health1);
          smartFire(enemy3Coordinates, jeff1Coordinates); // enemy fire when player is near
      }
      else
      {
          random(enemy3Coordinates, 0);
      }
   }
   // Enemy 4
   if (enemy4counter > 50)
   {
      bool chase4 = CheckChase(enemy4Coordinates, jeff2Coordinates);
      if (chase4)
      {
          chaseEnemy(enemy4Coordinates, jeff2Coordinates, health2);
          smartFire(enemy4Coordinates, jeff2Coordinates); // enemy fire when player is near
      }
      else
      {
          random(enemy4Coordinates, 1);
      }
   }
   random(enemy5Coordinates, 2);
   random(enemy6Coordinates, 3);
   random(enemy7Coordinates, 4);
   random(enemy8Coordinates, 5);
}
if (keys1 == 20 || keys2 == 20)
```

```
        {
            break;
        }
        playerHealth(jeff1Coordinates, health1); // health decrement when enemy bullet collides with player
        playerHealth(jeff2Coordinates, health2);
        gotoxy(35, 41);
        cout << name1 << "'s Score : " << score1;
        gotoxy(120, 41);
        cout << name2 << "'s  Score : " << score2;
        gotoxy(35, 42);
        cout << name1 << "'s health : " << health1 << " ";
        gotoxy(120, 42);
        cout << name2 << "'s health : " << health2 << " ";
        if ((score1 >= 1000) || (score2 >= 1000))
        {
            break;
        }
    }
    if ((score1 > score2))
    {
        printBox(30, 9, 70, 19);
        gotoxy(75, 22);
        cout << name1 << "!!!" << endl;
        gotoxy(75, 23);
        cout << "You Won ;(";
        gotoxy(75, 24);
        cout << "Your Score\t:\t" << score1;
        getch();
    }
    else
    {
        printBox(30, 9, 70, 19);
        gotoxy(75, 22);
        cout << name2 << "!!!" << endl;
        gotoxy(75, 23);
        cout << "You Won ;(";
        gotoxy(75, 24);
        cout << "Your Score\t:\t" << score2;

        getch();
        // Sleep(10000);
    }
}
void loading()
{
    for (int i = 0; i < 40; i++)
    {
        cout << char(219);
        Sleep(100);
    }
}
string levelsMenu()
{
    string opt;
    cout << "1.\tLevel 1" << endl;
    cout << "2.\tLevel 2" << endl;
    cout << "3.\tTwo Player" << endl;
    cout << "4.\tExit" << endl;
    cout << "\nEnter any option : ";
    cin >> opt;
    return opt;
}
void Instructions3()
{
    header();

    cout << "\n\n\t\t\t \t\t KEYS for Player 1\n";
    cout << "\t\t\t \t\tA\t\t\tGo left" << endl;
    cout << "\t\t\t \t\tD\t\t\tGo right" << endl;
    cout << "\t\t\t \t\tW\t\t\tGo up" << endl;
    cout << "\t\t\t \t\tS\t\t\tGo down" << endl;
    cout << "\t\t\t \t\tTab key\t\tRight Fire" << endl;
    cout << "\t\t\t \t\tAlt key\t\tLeft Fire" << endl;

    cout << "\n\n\t\t\t \t\t KEYS for Player 2\n";
    cout << "\t\t\t \t\tPgUP\t\t\tGo up" << endl;
    cout << "\t\t\t \t\tPgDown\t\t\tGo down" << endl;
    cout << "\t\t\t \t\tleft key\t\tGo left" << endl;
    cout << "\t\t\t \t\tRight key\t\tGo right" << endl;
    cout << "\t\t\t \t\tControl key\t\tRight Fire" << endl;
    cout << "\t\t\t \t\tEnter key\t\tLeft Fire" << endl;
    Sleep(5000);
}
void Instructions2()
{

    cout << "\t\t\t you will die and the game will end.\n";
    cout << "\n\t\t\t \t\t ENEMY FUNCTIONALITIES\n";
    cout << "\t\t\t The enemy can chase you from a certain distance else they move randomly\n";
    cout << "\t\t\t and can also fire bullets to the left or right when you are near.\n";
    cout << "\t\t\t If the bullet collides with you or the enemy touches you ,\n ";
    cout << "\t\t\t your health will decrease. ";
    cout << "\n\n\t\t\t\t\tSCORE SYSTEM\n";
    cout << "\t\t\t Collecting a key will increase the score by 5;";
    cout << "\n\t\t\t Bullet collision with enemy increases score by 1";
    getch();

    header();
    cout                                                                    <<
"\n<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
    cout << "\t\t\t\t\t\tInstructions\n";
    cout << "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
    cout << "\n\n\t\t\t \t\t\t KEYS\n";
    cout << "\t\t\t \t\tPgUP\t\t\tGo up" << endl;
    cout << "\t\t\t \t\tPgDown\t\t\tGo down" << endl;
    cout << "\t\t\t \t\tleft key\t\tGo left" << endl;
    cout << "\t\t\t \t\tRight key\t\tGo right" << endl;
    cout << "\t\t\t \t\tA\t\t\tLeft Fire" << endl;
    cout << "\t\t\t \t\tD\t\t\tRight Fire" << endl;
    cout << "\t\t\t \t\tW\t\t\tUp Fire" << endl;
    cout << "\t\t\t \t\tS\t\t\tDown Fire" << endl;
    cout << "\t\t\t \t\tESC\t\t\tPause Game" << endl;
    Sleep(5000);
    // cout << "6.\tEsc\t\t\tExit game" << endl;
}
void Instructions1()
{
    header();
    cout << "\n<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
    cout << "\t\t\t\t\t\t\tInstructions\n";
    cout << "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";

    cout << "\n\t\t\t \t\t Player Functionalities\n";
    cout << "\t\t\t In this game, you will be trying to escape from jail.\n";
    cout << "\t\t\t You need to collect some keys to open the doors.\n";
    cout << "\t\t\t When you collect keys and go to the corresponding door,\n";
    cout << "\t\t\t it will be opened.\n";
    cout << "\t\t\t After collecting 10 keys, go to the main gate to win the game.";
    cout << "\n\t\t\t In this game, you have three lives and a health system.\n";
    cout << "\t\t\t If your health reaches zero with last live ,\n";
    cout << "\t\t\t you will die and the game will end.";
    cout << "\n\n\t\t\t\t\tSCORE SYSTEM\n";
    cout << "\t\t\t Collecting a key will increase the score by 5;";
    cout << "\n\n\t\t\t \t\t Enemy Functionalities\n";
    cout << "\t\t\t The enemy can chase you from a certain distance\n\t\t\t else they move
randomly\n";
    cout << "\t\t\t and some enemies can also fire to their left or right when you are near.\n";
    cout << "\t\t\t If the bullet collides with you or the enemy touches you ,\n ";
    cout << "\t\t\t your health will decrease. ";
    getch();
    header();
    cout                                                                             <<
"\n<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
    cout << "\t\t\t\t\t\t\tInstructions\n";
    cout                                                                             <<
"<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
    cout << "\n\n\t\t\t \t\t\t\t KEYS\n";
    cout << "\t\t\t \t\tPgUP\t\t\tGo up" << endl;
    cout << "\t\t\t \t\tPgDown\t\t\tGo down" << endl;
    cout << "\t\t\t \t\tleft key\t\tGo left" << endl;
    cout << "\t\t\t \t\tRight key\t\tGo right" << endl;
    cout << "\t\t\t \t\tESC\t\t\tPause Game" << endl;
    Sleep(5000);


}
void keys2()
{
    cout << "\t\t\t\t\t\t\tKeys" << endl
         << endl;
    cout << "1.\tUP\t\t\tGo up" << endl;
    cout << "2.\tDown\t\t\tGo down" << endl;
    cout << "3.\tLeft\t\t\tGo left" << endl;
    cout << "4.\tRight\t\t\tGo right" << endl;
    cout << "5.\tA\t\t\tLeft Fire" << endl;
    cout << "5.\tD\t\t\tRight Fire" << endl;
    cout << "5.\tW\t\t\tUp Fire" << endl;
    cout << "5.\tS\t\t\tDown Fire" << endl;
    cout << "6.\tEsc\t\t\tExit game" << endl;
    cout << "\nPress any key to Start Game...";
    getch();
}

void level2cout(int Keys, string name)
{
    if (Keys < 15)
    {
        gotoxy(70, 41);
        cout << " " << name << "!!!";
        gotoxy(70, 42);
        cout << " You need " << 15 - Keys << " more Keys to Escape";
        if (Keys == 2)
        {
            gotoxy(70, 43);
            cout << " Congratulations, you've successfully located";
            gotoxy(70, 44);

            cout << " the keys needed to open door 1";
        }
        if (Keys == 5)
        {
            gotoxy(70, 43);
            cout << " Congratulations, you've successfully located";
            gotoxy(70, 44);

            cout << " the keys needed to open door 2";
        }
        if (Keys == 7)
        {
            gotoxy(70, 43);
            cout << " Congratulations, you've successfully located";
            gotoxy(70, 44);

            cout << " the keys needed to open door 3";
        }
        if (Keys == 10)
        {
            gotoxy(70, 43);
            cout << " Congratulations, you've successfully located";
            gotoxy(70, 44);

            cout << " the keys needed to open door 4";
        }
        if (Keys == 13)
        {
            gotoxy(70, 43);
            cout << " Congratulations, you've successfully located";
            gotoxy(70, 44);

            cout << " the keys needed to open door 5";
        }
    }
    else
    {
        gotoxy(70, 41);
        cout << " Great job!!!"
             << "                                        ";
        gotoxy(70, 42);
        cout << " You Have found the keys you needed."
             << "                    ";
        gotoxy(70, 43);
        cout << " Reach Exit Door ASAP!                    ";
        gotoxy(70, 43);
        cout << "                                        ";
    }
}
void Level2()
{
    string players[100][7]; // to store the name of all players
    int playercount = 0;    // player count
    int playerIndexinArray = -11111;
    loadPlayerDataLevel2(players, playercount);

    int enemy1counter = 50;
    int enemy2counter = 50;
    int enemy3counter = 50;
    int enemy4counter = 50;
    int enemy5counter = 50;
    int enemy6counter = 50;

    int lives = 3;
    int Keys = 0;
    int timer = 0;
    int health = 100;
    int score = 0;
    int keycount = 0;

    int jeffCoordinates[2] = {31, 2};
    int enemy1Coordinates[2] = {50, 8};
    int enemy2Coordinates[2] = {90, 7};
    int enemy3Coordinates[2] = {40, 14};
    int enemy4Coordinates[2] = {80, 17};
    int enemy5Coordinates[2] = {90, 30};
    int enemy6Coordinates[2] = {110, 35};
    string name;
a:
    cout << " Enter Your Name : ";
    cin >> name;
    bool validname = nameCheck(name);
    if (!validname)
    {
        goto a;
    }
    else
    {
        bool userexists = nameExists(name, players, playercount);
        if (userexists)
        {
            playerIndexinArray = userindex(name, players, playercount); // to fetch data  of existing
player from file from desiredline

            cout << players[playerIndexinArray][1];
            getch();
            Keys = stoi(players[playerIndexinArray][1]);
            lives = stoi(players[playerIndexinArray][2]);
            health = stoi(players[playerIndexinArray][3]);
            score = stoi(players[playerIndexinArray][4]);
            jeffCoordinates[0] = stoi(players[playerIndexinArray][5]);
            jeffCoordinates[1] = stoi(players[playerIndexinArray][6]);
            keycount = Keys - 1;
        }
        else
        {
            playerIndexinArray = playercount;
            playercount++;
            cout << "\n No User Exists. ";
            cout << "\n Thus,added new Player.";
            getch();
        }

        system("cls");
        // Mazeprint3();
        maze2();
        while ((health > -1 && lives > 0))
        {

            level2cout(Keys, name);
            if (lives > 1)
            {
                if (health == 0)
                {
                    lives = lives - 1;
                    health = 100;
                    eraseplayers(jeffCoordinates);
                    jeffCoordinates[0] = 31;
                    jeffCoordinates[1] = 2;
                    printJeffRight(jeffCoordinates);
                    Sleep(100);
                }
            }
            enemy1counter++; // counter becomees zero when bullet collides with enemy
            enemy2counter++;
            enemy3counter++;
            enemy4counter++;
            enemy5counter++;
            enemy6counter++;
            Sleep(1);
            playerHealth(jeffCoordinates, health);
            ScoreBoard(Keys, health, lives, score);
            timer++;
            if (GetAsyncKeyState(VK_ESCAPE))
            {
                break;
            }
            if (GetAsyncKeyState(VK_LEFT))
            {
                leftkey(jeffCoordinates, Keys, score);
```

```
}
if (GetAsyncKeyState(VK_RIGHT))
{
   rightkey(jeffCoordinates, Keys, score);
}
if (GetAsyncKeyState(VK_UP))
{
   upkey(jeffCoordinates, Keys, score);
}
if (GetAsyncKeyState(VK_DOWN))
{
   downkey(jeffCoordinates, Keys, score);
}
if (GetAsyncKeyState(0x44)) // D_key
{
   printJeffRightbulletfire(jeffCoordinates);
   generateBulletRight(jeffCoordinates);
}
if (GetAsyncKeyState(0x41)) // A_key
{
   printJeffLeftbulletfire(jeffCoordinates);
   generateBulletLeft(jeffCoordinates);
}
if (GetAsyncKeyState(0x57)) // W_key
{
   printJeffUpbulletfire(jeffCoordinates);
   generateBulletUp(jeffCoordinates);
}
if (GetAsyncKeyState(0x53)) // S_key
{
   printJeffDownbulletfire(jeffCoordinates);
   generateBulletDown(jeffCoordinates);
}
if (timer == 10)
```

```
        {
            keys2(keycount, Keys); // to print keys in game;
            timer = 0;
        }

        bool bulletcollision = EnemyCollisionwithBullet(enemy1Coordinates, score); // checks
bullet collission with enemy
        if (bulletcollision)
        {
            enemy1counter = 0; // to stop enemy
        }
        bool bulletcollision2 = EnemyCollisionwithBullet(enemy2Coordinates, score);
        if (bulletcollision2)
        {
            enemy2counter = 0;
        }
        bool bulletcollision3 = EnemyCollisionwithBullet(enemy3Coordinates, score);
        if (bulletcollision3)
        {
            enemy3counter = 0;
        }
        bool bulletcollision4 = EnemyCollisionwithBullet(enemy4Coordinates, score);
        if (bulletcollision4)
        {
            enemy4counter = 0;
        }
        bool bulletcollision5 = EnemyCollisionwithBullet(enemy5Coordinates, score);
        if (bulletcollision5)
        {
            enemy5counter = 0;
        }
        bool bulletcollision6 = EnemyCollisionwithBullet(enemy6Coordinates, score);
        if (bulletcollision6)
        {
            enemy6counter = 0;
```

```
}

if (timer % 5 == 0)
{
   // Enemy 1
   if (enemy1counter > 50) // if bullet collides with enemy then counter equals to zero
   {
      bool chase = CheckChase(enemy1Coordinates, jeffCoordinates); // chase
condition
      if (chase)
      {
         chaseEnemy(enemy1Coordinates, jeffCoordinates, health);
         smartFire(enemy1Coordinates, jeffCoordinates); // enemy fire when player is
near
      }
      else
      {
         random(enemy1Coordinates, 0); // random enemy
      }
   }
   // Enemy 2
   if (enemy2counter > 50)
   {
      bool chase2 = CheckChase(enemy2Coordinates, jeffCoordinates);
      if (chase2)
      {
         chaseEnemy(enemy2Coordinates, jeffCoordinates, health);
         smartFire(enemy2Coordinates, jeffCoordinates); // enemy fire when player is
near
      }
      else
      {
         random(enemy2Coordinates, 1);
      }
   }
```

```
// Enemy 3
if (enemy3counter > 50)
{
    bool chase3 = CheckChase(enemy3Coordinates, jeffCoordinates);
    if (chase3)
    {
        chaseEnemy(enemy3Coordinates, jeffCoordinates, health);
        smartFire(enemy3Coordinates, jeffCoordinates); // enemy fire when player is
near
    }
    else
    {
        random(enemy3Coordinates, 2);
    }
}
// Enemy 4
if (enemy4counter > 50)
{
    bool chase4 = CheckChase(enemy4Coordinates, jeffCoordinates);
    if (chase4)
    {
        chaseEnemy(enemy4Coordinates, jeffCoordinates, health);
        smartFire(enemy4Coordinates, jeffCoordinates); // enemy fire when player is
near
    }
    else
    {
        random(enemy4Coordinates, 3);
    }
}
// Enemy 5
if (enemy5counter > 50)
{
    bool chase5 = CheckChase(enemy5Coordinates, jeffCoordinates);
    if (chase5)
```

```
            {
                chaseEnemy(enemy5Coordinates, jeffCoordinates, health);
                smartFire(enemy5Coordinates, jeffCoordinates); // enemy fire when player is
near
            }
            else
            {
                random(enemy5Coordinates, 4);
            }
        }
        // enemy 6
        if (enemy6counter > 50)
        {
            bool chase4 = CheckChase(enemy6Coordinates, jeffCoordinates);
            if (chase4)
            {
                chaseEnemy(enemy6Coordinates, jeffCoordinates, health);
                smartFire(enemy6Coordinates, jeffCoordinates); // enemy fire when player is
near
            }
            else
            {
                random(enemy6Coordinates, 5);
            }
        }
    }
    if (timer % 2 == 0)
    {
        moveBulletRightE(); // bullet movement for enemy
        moveBulletLeftE();
        moveBulletLeft(); // bullet for player
        moveBulletRight();
        moveBulletUp();
        moveBulletDown();
    }
```

```
            /*Door Opens*/
            opendoors2(jeffCoordinates, Keys);
            if (Keys >= 15 && (jeffCoordinates[0] > 126 && jeffCoordinates[0] < 135) && (jeffCoordinates[1] > 18 && jeffCoordinates[1] < 22))
            {
                // last Gate to escape
                for (int i = 0; i < 4; i++)
                {
                    gotoxy(130, 18 + i);
                    cout << " ";
                }
                break;
            }
        }

        getch();
        players[playerIndexinArray][0] = name;
        players[playerIndexinArray][1] = to_string(Keys);
        players[playerIndexinArray][2] = to_string(lives);
        players[playerIndexinArray][3] = to_string(health);
        players[playerIndexinArray][4] = to_string(score);
        players[playerIndexinArray][5] = to_string(jeffCoordinates[0]);
        players[playerIndexinArray][6] = to_string(jeffCoordinates[1]);

        getch();
        if (Keys == 15 && (jeffCoordinates[0] > 126 && jeffCoordinates[0] < 135) && (jeffCoordinates[1] > 18 && jeffCoordinates[1] < 22))
        {

            printBox(30, 9, 65, 15);
            gotoxy(75, 18);
            cout << name << "!!!" << endl;
            gotoxy(72, 19);
            cout << "You won the game.";
            gotoxy(67, 20);
            cout << "Your Score\t:\t" << score;
            Sleep(3000);
            removePlayerfromArray(playerIndexinArray, playercount, players);
        }
        else if (health <= 0)
        {
            printBox(30, 9, 65, 15);
            gotoxy(75, 18);
            cout << name << "!!!" << endl;
            gotoxy(71, 19);
            cout << "You LOST ;(";
            gotoxy(67, 20);
            cout << "Your Score\t:\t" << score;
            Sleep(3000);
            removePlayerfromArray(playerIndexinArray, playercount, players);
        }
        else
        {
            printBox(30, 12, 63, 15);
            gotoxy(73, 18);
            cout << " Whoops!!!";
            gotoxy(73, 19);
            cout << name << "!!!";
            gotoxy(66, 20);
            cout << " You Paused your game..";
            gotoxy(66, 21);
            cout << " Your game data is stored, ";
            gotoxy(66, 22);
            cout << " so you can pick up where ";
            gotoxy(65, 23);
            cout << "you left off when you return.";
            Sleep(3000);
        }
        storePlayerDataLevel2(players, playercount);
        }
}
void Level1()
{
    string players[100][7]; // to store the name of all players
    int playercount = 0;    // player count
    int playerIndexinArray = -11111;
    loadPlayerDataLevel1(players, playercount);
    int Keys = 0;
    int timer = 0;
    int lives = 3;
    int health = 100;
    int score = 0;
    int keycount = 0; // for random keys limit

    int jeffCoordinates[2] = {31, 4};
    int enemy1Coordinates[2] = {59, 5};
    int enemy2Coordinates[2] = {32, 16};
    int enemy3Coordinates[2] = {60, 30};
    int enemy4Coordinates[2] = {49, 27};

    string name;
a:
    cout << "\n\n\n Enter Your Name : ";
    cin >> name;
    bool validname = nameCheck(name);
    if (!validname)
    {
        goto a;
    }
    else
    {
        bool userexists = nameExists(name, players, playercount);
        if (userexists)
```

```
{
    playerIndexinArray = userindex(name, players, playercount); // to fetch data of existing player from file from desiredline
    Keys = stoi(players[playerIndexinArray][1]);
    lives = stoi(players[playerIndexinArray][2]);
    health = stoi(players[playerIndexinArray][3]);
    score = stoi(players[playerIndexinArray][4]);
    jeffCoordinates[0] = stoi(players[playerIndexinArray][5]);
    jeffCoordinates[1] = stoi(players[playerIndexinArray][6]);
    keycount = Keys - 1;
}
else
{
    playerIndexinArray = playercount;
    playercount++;
    cout << "\n New Player Added.";
    cout << "\nPress Any key to start you game...";
    getch();
}
system("cls");
maze1();

// random key generate

while (health > -1)
{
    if (Keys < 10)
    {
        gotoxy(70, 41);
        cout << name << "!!!";
        gotoxy(70, 42);
        cout << "You need " << 10 - Keys << " more Keys to Escape";
    }
    else
    {
        gotoxy(70, 41);
        cout << " Great job!!!"
             << "                    ";
        gotoxy(70, 42);
        cout << " You Have found the keys you needed.";
        gotoxy(70, 43);
        cout << " Reach Escape Door ASAP!";
    }
    if (health == 0)
    {
        if (lives > 1)
        {
            Sleep(10);
            lives = lives - 1;
            health = 100;
            eraseplayers(jeffCoordinates);
            jeffCoordinates[0] = 32;
            jeffCoordinates[1] = 4;
            printJeffRight(jeffCoordinates);
            Sleep(100);
        }
    }
    Sleep(10);
    ScoreBoard(Keys, health, lives, score);
    timer++;
    if (Keys < 3)
    {
        int x = rand() % 88;
        int y = rand() % 9;

        char location = getCharAtxy(x + 30, y + 2);
        if (location == ' ' && keycount < 2)
        {
            gotoxy(x + 30, y + 2);
            cout << key;
            keycount++;
        }
    }
    if (Keys < 6 && Keys >= 2)
    {
        int x = rand() % 88;
        int y = rand() % 9;
        char location = getCharAtxy(x + 30, y + 10 + 2);
        if (location == ' ' && keycount < 5)
        {
            gotoxy(x + 30, y + 10 + 2);
            cout << key;
            keycount++;
        }
    }
    if (Keys < 10 && Keys >= 5)
    {
        int x = rand() % 88;
        int y = rand() % 13;
        char location = getCharAtxy(x + 25, y + 25);
        if (location == ' ' && keycount < 10)
        {
            gotoxy(x + 30, y + 25);
            cout << key;
            keycount++;
        }
    }
    if (GetAsyncKeyState(VK_ESCAPE))
    {
        break;
    }
    if (GetAsyncKeyState(VK_LEFT))
    {
        leftkey(jeffCoordinates, Keys, score);
    }
    if (GetAsyncKeyState(VK_RIGHT))
    {
        rightkey(jeffCoordinates, Keys, score);
    }
    if (GetAsyncKeyState(VK_UP))
    {
        upkey(jeffCoordinates, Keys, score);
    }
    if (GetAsyncKeyState(VK_DOWN))
    {
        downkey(jeffCoordinates, Keys, score);
    }
    if (timer == 5)
    {
        // Enemy 1
        bool chase = CheckChase(enemy1Coordinates, jeffCoordinates);
        if (chase)
        {
            chaseEnemy(enemy1Coordinates, jeffCoordinates, health);
        }
        else
        {
            random(enemy1Coordinates, 0);
        }
        // Enemy 2
        bool chase2 = CheckChase(enemy2Coordinates, jeffCoordinates);
        if (chase2)
        {
            chaseEnemy(enemy2Coordinates, jeffCoordinates, health);
        }
        else
        {
```

```
        random(enemy2Coordinates, 1);
    }
    // Enemy 3
    bool chase3 = CheckChase(enemy3Coordinates, jeffCoordinates);
    if (chase3)
    {
        chaseEnemy(enemy3Coordinates, jeffCoordinates, health);
    }
    else
    {
        random(enemy3Coordinates, 2);
    }
    // Enemy 4
    bool chase4 = CheckChase(enemy4Coordinates, jeffCoordinates);
    if (chase4)
    {
        chaseEnemy(enemy4Coordinates, jeffCoordinates, health);
    }
    else
    {
        random(enemy4Coordinates, 3);
    }
    timer = 0;
}
smartFire(enemy1Coordinates, jeffCoordinates); // enemy fires when player is near
moveBulletRightE();                          // bullet movement for enemy
moveBulletLeftE();
if (Keys >= 2)
{
    gotoxy(31, 13);
    cout << "        ";
}
if (Keys >= 5)
{
```

```
            gotoxy(63, 24);
            cout << "        ";
        }
        if (Keys >= 10 && (jeffCoordinates[0] > 110 && jeffCoordinates[0] < 115) && (jeffCoordinates[1] > 2 && jeffCoordinates[1] < 5))
        {
            for (int i = 0; i < 6; i++)
            {
                gotoxy(115, 3 + i);
                cout << "  ";
                Sleep;
            }
            break;
        }

        playerHealth(jeffCoordinates, health);
    }
    players[playerIndexinArray][0] = name;
    players[playerIndexinArray][1] = to_string(Keys);
    players[playerIndexinArray][2] = to_string(lives);
    players[playerIndexinArray][3] = to_string(health);
    players[playerIndexinArray][4] = to_string(score);
    players[playerIndexinArray][5] = to_string(jeffCoordinates[0]);
    players[playerIndexinArray][6] = to_string(jeffCoordinates[1]);

    if (Keys >= 10)
    {

        printBox(30, 9, 65, 15);
        gotoxy(75, 18);
        cout << name << "!!!" << endl;
        gotoxy(72, 19);
        cout << "You won the game.";
        gotoxy(67, 20);
        cout << "Your Score\t:\t" << score;
        // Sleep(1000);
        getch();
        removePlayerfromArray(playerIndexinArray, playercount, players);
    }
    else if (health <= 0)
    {
        printBox(30, 9, 65, 15);
        gotoxy(75, 18);
        cout << name << "!!!" << endl;
        gotoxy(71, 19);
        cout << "You LOST ;(";
        gotoxy(67, 20);
        cout << "Your Score\t:\t" << score;
        getch();
        removePlayerfromArray(playerIndexinArray, playercount, players);
        // Sleep(1000);
    }
    else
    {
        printBox(30, 13, 63, 15);
        gotoxy(73, 18);
        cout << " Whoops!!!";
        gotoxy(73, 19);
        cout << name << "!!!";
        gotoxy(64, 20);
        cout << " You Paused your game..";
        gotoxy(64, 21);
        cout << " Your game data is stored, ";
        gotoxy(64, 22);
        cout << " so you can pick up where ";
        gotoxy(64, 23);
        cout << "you left off when you return.";
        getch();
    }
```

```
        storePlayerDataLevel1(players, playercount);
    }
}
void leftkey(int jeffCoordinates[2], int &Keys, int &score)
{
    char nextlocation = getCharAtxy(jeffCoordinates[0] - 1, jeffCoordinates[1]);
    char nextlocation1 = getCharAtxy(jeffCoordinates[0] - 1, jeffCoordinates[1] + 1);
    char nextlocation2 = getCharAtxy(jeffCoordinates[0] - 1, jeffCoordinates[1] + 2);
    if (nextlocation == key || nextlocation1 == key || nextlocation2 == key)
    {
        Keys++;
        score = score + 5;
        eraseplayers(jeffCoordinates);
        jeffCoordinates[0] = jeffCoordinates[0] - 1;
        printJeffLeft(jeffCoordinates);
    }
    else if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
    {
        eraseplayers(jeffCoordinates);
        jeffCoordinates[0] = jeffCoordinates[0] - 1;
        printJeffLeft(jeffCoordinates);
    }
}
void rightkey(int jeffCoordinates[2], int &Keys, int &score)
{
    char nextlocation = getCharAtxy(jeffCoordinates[0] + 3, jeffCoordinates[1]);
    char nextlocation1 = getCharAtxy(jeffCoordinates[0] + 3, jeffCoordinates[1] + 1);
    char nextlocation2 = getCharAtxy(jeffCoordinates[0] + 3, jeffCoordinates[1] + 2);
    if (nextlocation == key || nextlocation1 == key || nextlocation2 == key)
    {
        Keys++;
        score = score + 5;
        eraseplayers(jeffCoordinates);
        jeffCoordinates[0] = jeffCoordinates[0] + 1;
        printJeffLeft(jeffCoordinates);
    }
    else if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
    {

        eraseplayers(jeffCoordinates);
        jeffCoordinates[0] = jeffCoordinates[0] + 1;
        printJeffRight(jeffCoordinates);
    }
}
void upkey(int jeffCoordinates[2], int &Keys, int &score)
{
    char nextlocation = getCharAtxy(jeffCoordinates[0], jeffCoordinates[1] - 1);
    char nextlocation1 = getCharAtxy(jeffCoordinates[0] + 1, jeffCoordinates[1] - 1);
    char nextlocation2 = getCharAtxy(jeffCoordinates[0] + 2, jeffCoordinates[1] - 1);
    if (nextlocation == key || nextlocation1 == key || nextlocation2 == key)
    {
        Keys++;
        score = score + 5;
        eraseplayers(jeffCoordinates);
        jeffCoordinates[1] = jeffCoordinates[1] - 1;
        printJeffLeft(jeffCoordinates);
    }
    else if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
    {
        eraseplayers(jeffCoordinates);
        jeffCoordinates[1] = jeffCoordinates[1] - 1;
        printJeffRight(jeffCoordinates);
    }
}
void downkey(int jeffCoordinates[2], int &Keys, int &score)
{
    char nextlocation = getCharAtxy(jeffCoordinates[0], jeffCoordinates[1] + 3);
    char nextlocation1 = getCharAtxy(jeffCoordinates[0] + 1, jeffCoordinates[1] + 3);
    char nextlocation2 = getCharAtxy(jeffCoordinates[0] + 2, jeffCoordinates[1] + 3);
    if (nextlocation == key || nextlocation1 == key || nextlocation2 == key)
    {
        Keys++;
        score = score + 5;
        eraseplayers(jeffCoordinates);
        jeffCoordinates[1] = jeffCoordinates[1] + 1;
        printJeffLeft(jeffCoordinates);
    }
    else if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
    {
        eraseplayers(jeffCoordinates);
        jeffCoordinates[1] = jeffCoordinates[1] + 1;
        printJeffRight(jeffCoordinates);
    }
}
void printJeffLeft(int jeffCoordinates[2])
{
    string arr[3][3] = {
        {" ", "O", " "},
        {"/", "|", "\\"},
        {" ", "\\", "\\"}};
    for (int i = 0; i < 3; i++)
    {
        gotoxy(jeffCoordinates[0], jeffCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j];
        }
        cout << endl;
    }
}
void printJeffRight(int jeffCoordinates[2])
{
    string player[3][3] = {
        {" ", "O", " "},
        {"/", "|", "\\"},
        {"/", "/", ""}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(jeffCoordinates[0], jeffCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j];
        }
        cout << endl;
    }
}
void printJeffRightbulletfire(int jeffCoordinates[2])
{

    char arr[3][3] = {
        {' ', 'O', ' '},
        {'/', '|', char(61)},
        {'/', '/', ' '}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(jeffCoordinates[0], jeffCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j];
        }
        cout << endl;
    }
}
```

```
void printJeffUpbulletfire(int jeffCoordinates[2])
{

    char arr[3][3] = {
        {' ', char(208), ' '},
        {'\\', 'O', '/'},
        {char(193), char(193), char(193)}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(jeffCoordinates[0], jeffCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j];
        }
        cout << endl;
    }
}
void printJeffDownbulletfire(int jeffCoordinates[2])
{

    char arr[3][3] = {
        {char(194), char(194), char(194)},
        {'/', 'O', '\\'},
        {' ', char(210), ' '}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(jeffCoordinates[0], jeffCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j];
        }
        cout << endl;
    }
}
void printJeffLeftbulletfire(int jeffCoordinates[2])
{

    char arr[3][3] = {
        {' ', 'O', ' '},
        {char(185), '|', '\\'},
        {' ', '\\', '\\'}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(jeffCoordinates[0], jeffCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j];
        }
        cout << endl;
    }
}
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 : ' ';
}
void gotoxy(int x, int y)
{
    COORD coordinates;
    coordinates.X = x;
    coordinates.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinates);
}
}

void eraseplayers(int Coordinates[2])
{
    for (int i = 0; i < 3; i++)
    {
        gotoxy(Coordinates[0], Coordinates[1] + i);
        cout << "   ";
    }
}
void printEnemy(int enemyCoordinates[2])
{

    char enemy[3][3] = {
        {' ', char(236), ' '},
        {char(181), char(178), char(198)},
        {' ', char(239), ' '}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(enemyCoordinates[0], enemyCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << enemy[i][j];
        }
        cout << endl;
    }
}
void printenemyunderweb(int enemyCoordinates[2])
{
    char enemy[3][3] = {
        {' ', char(236), ' '},
        {'{', char(178), '}'},
        {' ', char(239), ' '}};

    for (int i = 0; i < 3; i++)
    {
        gotoxy(enemyCoordinates[0], enemyCoordinates[1] + i);
        for (int j = 0; j < 3; j++)
        {
            cout << enemy[i][j];
        }
        cout << endl;
    }
}
bool CheckChase(int enemyCoordinates[2], int jeffCoordinates[2])
{
    int diff_x = jeffCoordinates[0] - enemyCoordinates[0];
    int diff_y = jeffCoordinates[1] - enemyCoordinates[1];
    char location = getCharAtxy(enemyCoordinates[0], enemyCoordinates[1]);
    if ((diff_x > -20 && diff_x < 20) && (diff_y > -7 && diff_y < 7))
    {
        return true;
    }
    return false;
}
void chaseEnemy(int enemyCoordinates[2], int jeffCoordinates[2], int &Health)
{
    int locationX = enemyCoordinates[0] - jeffCoordinates[0]; // difference for chase
    int locationY = enemyCoordinates[1] - jeffCoordinates[1];

    if (locationX > 0) // left
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0] - 1, enemyCoordinates[1]);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] - 1, enemyCoordinates[1] + 1);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] - 1, enemyCoordinates[1] + 2);
        if (nextlocation == 'O' || nextlocation == '\\' || nextlocation == '/' || nextlocation1 == 'O' || nextlocation1 == '\\' || nextlocation1 == '/' || nextlocation2 == 'O' || nextlocation2 == '\\' || nextlocation2 == '/')
        {
            Health--;
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[0] = enemyCoordinates[0] - 1;
            printEnemy(enemyCoordinates);
        }
    }
    if (locationX < 0) // right
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0] + 3, enemyCoordinates[1]);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] + 3, enemyCoordinates[1] + 1);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] + 3, enemyCoordinates[1] + 2);
        if (nextlocation == 'O' || nextlocation == '\\' || nextlocation == '/' || nextlocation1 == 'O' || nextlocation1 == '\\' || nextlocation1 == '/' || nextlocation2 == 'O' || nextlocation2 == '\\' || nextlocation2 == '/')
        {
            Health--;
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[0] = enemyCoordinates[0] + 1;
            printEnemy(enemyCoordinates);
        }
    }
    if (locationY < 0) // down
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0], enemyCoordinates[1] + 3);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] + 1, enemyCoordinates[1] + 3);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] + 2, enemyCoordinates[1] + 3);
        if (nextlocation == 'O' || nextlocation == '\\' || nextlocation == '/' || nextlocation1 == 'O' ||
nextlocation1 == '\\' || nextlocation1 == '/' || nextlocation2 == 'O' || nextlocation2 == '\\' ||
nextlocation2 == '/')
        {
            Health--;
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[1] = enemyCoordinates[1] + 1;
            printEnemy(enemyCoordinates);
        }
    }
    if (locationY > 0) // up
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0], enemyCoordinates[1] - 1);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] + 1, enemyCoordinates[1] - 1);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] + 2, enemyCoordinates[1] - 1);
        if (nextlocation == 'O' || nextlocation == '\\' || nextlocation == '/' || nextlocation1 == 'O' ||
nextlocation1 == '\\' || nextlocation1 == '/' || nextlocation2 == 'O' || nextlocation2 == '\\' ||
nextlocation2 == '/')
        {
            Health--;
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[1] = enemyCoordinates[1] - 1;
            printEnemy(enemyCoordinates);
        }
    }
}
void random(int enemyCoordinates[2], int index)
```

```
{
    if (random_Direction[index] == "right")
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0] + 3, enemyCoordinates[1]);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] + 3, enemyCoordinates[1] + 1);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] + 3, enemyCoordinates[1] + 2);
        if (nextlocation != ' ' || nextlocation1 != ' ' || nextlocation2 != ' ')
        {
            int count = rand() % 3 + 1;
            if (count == 0)
            {
                random_Direction[index] = "left";
            }
            if (count == 1)
            {
                random_Direction[index] = "up";
            }
            if (count == 2)
            {
                random_Direction[index] = "down";
            }
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[0] = enemyCoordinates[0] + 1;
            printEnemy(enemyCoordinates);
        }
    }
    if (random_Direction[index] == "left")
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0] - 1, enemyCoordinates[1]);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] - 1, enemyCoordinates[1] + 1);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] - 1, enemyCoordinates[1] + 2);
        if (nextlocation != ' ' || nextlocation1 != ' ' || nextlocation2 != ' ')
        {
            int count = rand() % 3 + 1;
            if (count == 0)
            {
                random_Direction[index] = "right";
            }
            if (count == 1)
            {
                random_Direction[index] = "up";
            }
            if (count == 2)
            {
                random_Direction[index] = "down";
            }
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[0] = enemyCoordinates[0] - 1;
            printEnemy(enemyCoordinates);
        }
    }
    if (random_Direction[index] == "up")
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0], enemyCoordinates[1] - 1);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] + 1, enemyCoordinates[1] - 1);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] + 2, enemyCoordinates[1] - 1);
        if (nextlocation != ' ' || nextlocation1 != ' ' || nextlocation2 != ' ')
        {
            int count = rand() % 3 + 1;
            if (count == 1)
            {
                random_Direction[index] = "right";
            }
            if (count == 2)
            {
                random_Direction[index] = "left";
            }
            if (count == 3)
            {
                random_Direction[index] = "down";
            }
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[1] = enemyCoordinates[1] - 1;
            printEnemy(enemyCoordinates);
        }
    }
    if (random_Direction[index] == "down")
    {
        char nextlocation = getCharAtxy(enemyCoordinates[0], enemyCoordinates[1] + 3);
        char nextlocation1 = getCharAtxy(enemyCoordinates[0] + 1, enemyCoordinates[1] + 3);
        char nextlocation2 = getCharAtxy(enemyCoordinates[0] + 2, enemyCoordinates[1] + 3);
        if (nextlocation != ' ' || nextlocation1 != ' ' || nextlocation2 != ' ')
        {
            int count = rand() % 3 + 1;
            if (count == 1)
            {
                random_Direction[index] = "right";
            }
            if (count == 2)
            {
                random_Direction[index] = "left";
            }
            if (count == 3)
            {
                random_Direction[index] = "up";
            }
        }
        if (nextlocation == ' ' && nextlocation1 == ' ' && nextlocation2 == ' ')
        {
            eraseplayers(enemyCoordinates);
            enemyCoordinates[1] = enemyCoordinates[1] + 1;
            printEnemy(enemyCoordinates);
        }
    }
}
void smartFire(int enemyCoordinates[2], int jeffCoordinates[2])
{
    // function for enemy fire when player is near enemy (right/left) side
    int diff_x = jeffCoordinates[0] - enemyCoordinates[0];
    int diff_y = jeffCoordinates[1] - enemyCoordinates[1];
    if ((diff_y > -3 && diff_y < 3) && (diff_x < 0 && diff_x > -10))
    {
        generateBulletLeftE(enemyCoordinates);
    }
    if ((diff_y > -3 && diff_y < 3) && (diff_x > 0 && diff_x < 10))
    {
        generateBulletRightE(enemyCoordinates);
    }
}
void printBulletE(int x, int y)
{
    gotoxy(x, y);
    cout << char(196);
}
void eraseBullet(int x, int y)
{
    gotoxy(x, y);
    cout << " ";
}
void generateBulletRightE(int enemyCoordinates[2])
{
    EbulletR[E_bulletCountR][0] = enemyCoordinates[0] + 3;
    EbulletR[E_bulletCountR][1] = enemyCoordinates[1] + 1;
    char nextlocation = getCharAtxy(EbulletR[E_bulletCountR][0],
EbulletR[E_bulletCountR][1]);
    if (nextlocation == ' ')
    {
        ErightCount[E_bulletCountR] = 0;
        printBulletE(EbulletR[E_bulletCountR][0], EbulletR[E_bulletCountR][1]);
        E_bulletCountR++;
    }
}
void generateBulletLeftE(int enemyCoordinates[2])
{
    EbulletL[E_bulletCountL][0] = enemyCoordinates[0] - 1;
    EbulletL[E_bulletCountL][1] = enemyCoordinates[1] + 1;
    char nextlocation = getCharAtxy(EbulletL[E_bulletCountL][0],
EbulletL[E_bulletCountL][1]);
    if (nextlocation == ' ')
    {

        EleftCount[E_bulletCountL] = 0;
        printBulletE(EbulletL[E_bulletCountL][0], EbulletL[E_bulletCountL][1]);
        E_bulletCountL++;
    }
}
void moveBulletRightE()
{
    for (int x = 0; x < E_bulletCountR; x++)
    {
        ErightCount[x]++;
        char next = getCharAtxy(EbulletR[x][0] + 1, EbulletR[x][1]);
        if (next != ' ' || ErightCount[x] == 7)
        {
            eraseBullet(EbulletR[x][0], EbulletR[x][1]);
            removeBulletFromArrayER(x);
        }
        else
        {
            eraseBullet(EbulletR[x][0], EbulletR[x][1]);
            EbulletR[x][0] = EbulletR[x][0] + 1;
            printBulletE(EbulletR[x][0], EbulletR[x][1]);
        }
    }
}
void moveBulletLeftE()
{
    for (int x = 0; x < E_bulletCountL; x++)
    {
        EleftCount[x]++;
        char next = getCharAtxy(EbulletL[x][0] - 1, EbulletL[x][1]);
        if (next != ' ' || EleftCount[x] == 6)
        {
            eraseBullet(EbulletL[x][0], EbulletL[x][1]);
            removeBulletFromArrayEL(x);
        }
        else
        {
            eraseBullet(EbulletL[x][0], EbulletL[x][1]);
            EbulletL[x][0] = EbulletL[x][0] - 1;
            printBulletE(EbulletL[x][0], EbulletL[x][1]);
        }
    }
}
void removeBulletFromArrayER(int index) // enemy right
{
    E_bulletCountR--;
    for (int i = 0; i < 2; i++)
    {
        EbulletR[index][i] = EbulletR[E_bulletCountR][i];
    }
    ErightCount[index] = ErightCount[E_bulletCountR];
}
void removeBulletFromArrayEL(int index) // enemy left
{
    E_bulletCountL--;
    for (int i = 0; i < 2; i++)
    {
        EbulletL[index][i] = EbulletL[E_bulletCountL][i];
    }
    EleftCount[index] = EleftCount[E_bulletCountL];
}
void ScoreBoard(int keys, int health, int lives, int score)
{

    gotoxy(35, 41);
    cout << " Lives\t:\t" << lives << "  ";
    gotoxy(35, 42);
    cout << " Health\t:\t" << health << "  ";
    gotoxy(35, 43);
    cout << " Keys\t:\t" << keys;
    gotoxy(35, 44);
    cout << " Scorez\t:\t" << score;
}

void generateBulletUp(int jeffCoordinates[2])
{
    // (hConsole, 1);
    PbulletU[P_bulletCountU][0] = jeffCoordinates[0] + 1;
    PbulletU[P_bulletCountU][1] = jeffCoordinates[1] - 1;
    char nextlocation = getCharAtxy(PbulletU[P_bulletCountU][0], PbulletU[P_bulletCountU][1]);
    if (nextlocation != '#')
    {
        printBullet(PbulletU[P_bulletCountU][0], PbulletU[P_bulletCountU][1]);
        PUpCount[P_bulletCountU] = 0;
        P_bulletCountU++;
    }
}
void generateBulletDown(int jeffCoordinates[2])
{

    PbulletD[P_bulletCountD][0] = jeffCoordinates[0] + 1;
    PbulletD[P_bulletCountD][1] = jeffCoordinates[1] + 3;
    char nextlocation = getCharAtxy(PbulletD[P_bulletCountD][0], PbulletD[P_bulletCountD][1]);
    if (nextlocation != '#')
    {
        printBullet(PbulletD[P_bulletCountD][0], PbulletD[P_bulletCountD][1]);
        PDownCount[P_bulletCountD] = 0;
        P_bulletCountD++;
    }
}
void generateBulletLeft(int jeffCoordinates[2])
{

    PbulletL[P_bulletCountL][0] = jeffCoordinates[0] - 1;
    PbulletL[P_bulletCountL][1] = jeffCoordinates[1] + 1;
    char nextlocation = getCharAtxy(PbulletL[P_bulletCountL][0], PbulletL[P_bulletCountL][1]);
    if (nextlocation != '#')
    {
        printBullet(PbulletL[P_bulletCountL][0], PbulletL[P_bulletCountL][1]);
        PleftCount[P_bulletCountL] = 0;
        P_bulletCountL++;
    }
}
void generateBulletRight(int jeffCoordinates[2])
{
    PbulletR[P_bulletCountR][0] = jeffCoordinates[0] + 3;
    PbulletR[P_bulletCountR][1] = jeffCoordinates[1] + 1;
    char nextlocation = getCharAtxy(PbulletR[P_bulletCountR][0], PbulletR[P_bulletCountR][1]);
    if (nextlocation != '#')
    {
        printBullet(PbulletR[P_bulletCountR][0], PbulletR[P_bulletCountR][1]);
        PrightCount[P_bulletCountR] = 0;
        P_bulletCountR++;
    }
}
void moveBulletRight()
{
    for (int x = 0; x < P_bulletCountR; x++)
    {
        PrightCount[x]++;
        char next = getCharAtxy(PbulletR[x][0] + 1, PbulletR[x][1]);
        if (next != ' ' || PrightCount[x] == 10)
        {
            eraseBullet(PbulletR[x][0], PbulletR[x][1]);
            removeBulletFromArrayR(x);
        }
        // if (PrightCount[x] == 10)
        // {
        //     eraseBullet(PbulletR[x][0], PbulletR[x][1]);
        //     removeBulletFromArrayR(x);
        // }
        else
        {
            eraseBullet(PbulletR[x][0], PbulletR[x][1]);
            PbulletR[x][0] = PbulletR[x][0] + 1;
            printBullet(PbulletR[x][0], PbulletR[x][1]);
        }
    }
}
void moveBulletLeft()
{
    for (int x = 0; x < P_bulletCountL; x++)
    {
        PleftCount[x]++;
        char next = getCharAtxy(PbulletL[x][0] - 1, PbulletL[x][1]);
        if (next != ' ')
        {
            eraseBullet(PbulletL[x][0], PbulletL[x][1]);
            removeBulletFromArrayL(x);
        }
        else
        {
            eraseBullet(PbulletL[x][0], PbulletL[x][1]);
            PbulletL[x][0] = PbulletL[x][0] - 1;
            printBullet(PbulletL[x][0], PbulletL[x][1]);
        }
        if (PleftCount[x] == 10)
        {
            eraseBullet(PbulletL[x][0], PbulletL[x][1]);
            removeBulletFromArrayL(x);
        }
    }
}
void moveBulletDown()
{
    for (int x = 0; x < P_bulletCountD; x++)
    {
        PDownCount[x]++;
        char next = getCharAtxy(PbulletD[x][0], PbulletD[x][1] + 1);
        if (next != ' ' || PDownCount[x] == 10)
        {
            eraseBullet(PbulletD[x][0], PbulletD[x][1]);
            removeBulletFromArrayD(x);
        }
        else
        {
            eraseBullet(PbulletD[x][0], PbulletD[x][1]);
            PbulletD[x][1] = PbulletD[x][1] + 1;
            printBullet(PbulletD[x][0], PbulletD[x][1]);
        }
    }
}
void moveBulletUp()
{
    for (int x = 0; x < P_bulletCountU; x++)
    {
        PUpCount[x]++;
        char next = getCharAtxy(PbulletU[x][0], PbulletU[x][1] - 1);
        if (next != ' ' || PUpCount[x] == 10)
        {
            eraseBullet(PbulletU[x][0], PbulletU[x][1]);
            removeBulletFromArrayU(x);
        }
        else if (next == ' ')
        {
            eraseBullet(PbulletU[x][0], PbulletU[x][1]);
            PbulletU[x][1] = PbulletU[x][1] - 1;
            printBullet(PbulletU[x][0], PbulletU[x][1]);
        }
        PUpCount[x]++;
    }
}
void removeBulletFromArrayU(int index)
```

```
{
   P_bulletCountU--;
   for (int x = 0; x < 2; x++)
   {
      PbulletU[index][x] = PbulletU[P_bulletCountU][x];
   }
   PUpCount[index] = PUpCount[P_bulletCountU];
}
void removeBulletFromArrayD(int index)
{
   P_bulletCountD--;
   for (int x = 0; x < 2; x++)
   {
      PbulletD[index][x] = PbulletD[P_bulletCountD][x];
   }
   PDownCount[index] = PDownCount[P_bulletCountD];
}
void removeBulletFromArrayR(int index)
{
   P_bulletCountR--;
   for (int x = 0; x < 2; x++)
   {
      PbulletR[index][x] = PbulletR[P_bulletCountR][x];
   }
   PrightCount[index] = PrightCount[P_bulletCountR];
}
void removeBulletFromArrayL(int index)
{
   P_bulletCountL--;
   for (int x = 0; x < 2; x++)
   {
      PbulletL[index][x] = PbulletL[P_bulletCountL][x];
   }
   PleftCount[index] = PleftCount[P_bulletCountL];
```

```
}
void printBullet(int x, int y)
{
    gotoxy(x, y);
    cout << ".";
}
void playerHealth(int jeffCoordinates[2], int &Health)
{
    for (int x = 0; x < E_bulletCountL; x++)
    {
        if ((EbulletL[x][0] - 1 == jeffCoordinates[0] && EbulletL[x][1] == jeffCoordinates[1]) ||
(EbulletL[x][0] - 1 == jeffCoordinates[0] + 2 && EbulletL[x][1] == jeffCoordinates[1]))
        {
            Health--;
        }
        if ((EbulletL[x][0] - 1 == jeffCoordinates[0] && EbulletL[x][1] == jeffCoordinates[1] +
1) || (EbulletL[x][0] - 1 == jeffCoordinates[0] + 2 && EbulletL[x][1] == jeffCoordinates[1] +
1))
        {
            Health--;
        }
        if ((EbulletL[x][0] - 1 == jeffCoordinates[0] && EbulletL[x][1] == jeffCoordinates[1] +
2) || (EbulletL[x][0] - 1 == jeffCoordinates[0] + 2 && EbulletL[x][1] == jeffCoordinates[1] +
2))
        {
            Health--;
        }
    }
    for (int x = 0; x < E_bulletCountR; x++)
    {
        if ((EbulletR[x][0] + 1 == jeffCoordinates[0] && EbulletR[x][1] == jeffCoordinates[1])
|| (EbulletR[x][0] + 1 == jeffCoordinates[0] + 2 && EbulletR[x][1] == jeffCoordinates[1]))
        {
            Health--;
        }
        if ((EbulletR[x][0] + 1 == jeffCoordinates[0] && EbulletR[x][1] == jeffCoordinates[1] + 1) || (EbulletR[x][0] + 1 == jeffCoordinates[0] + 2 && EbulletR[x][1] == jeffCoordinates[1] + 1))
        {
            Health--;
        }
        if ((EbulletR[x][0] + 1 == jeffCoordinates[0] && EbulletR[x][1] == jeffCoordinates[1] + 2) || (EbulletR[x][0] + 1 == jeffCoordinates[0] + 2 && EbulletR[x][1] == jeffCoordinates[1] + 2))
        {
            Health--;
        }
    }
}
bool EnemyCollisionwithBullet(int enemyCoordinates[2], int &Score)
{
    for (int x = 0; x < P_bulletCountR; x++)
    {
        if ((PbulletR[x][0] + 1 == enemyCoordinates[0] && PbulletR[x][1] == enemyCoordinates[1]) || (PbulletR[x][0] + 1 == enemyCoordinates[0] && PbulletR[x][1] == enemyCoordinates[1] + 1) || (PbulletR[x][0] + 1 == enemyCoordinates[0] && PbulletR[x][1] == enemyCoordinates[1] + 2))
        {
            printenemyunderweb(enemyCoordinates);
            Score++;
            return true;
        }
    }
    for (int x = 0; x < P_bulletCountL; x++)
    {
        if ((PbulletL[x][0] - 1 == enemyCoordinates[0] + 2 && PbulletL[x][1] == enemyCoordinates[1]) || (PbulletL[x][0] - 1 == enemyCoordinates[0] + 2 && PbulletL[x][1] == enemyCoordinates[1] + 1) || (PbulletL[x][0] - 1 == enemyCoordinates[0] + 2 && PbulletL[x][1] == enemyCoordinates[1] + 2))
        {
            printenemyunderweb(enemyCoordinates);
            Score++;
            return true;
        }
    }
    for (int x = 0; x < P_bulletCountU; x++)
    {
        if ((PbulletU[x][0] == enemyCoordinates[0] && PbulletU[x][1] - 1 == enemyCoordinates[1] + 2) || (PbulletU[x][0] == enemyCoordinates[0] + 1 && PbulletU[x][1] - 1 == enemyCoordinates[1] + 2) || (PbulletU[x][0] == enemyCoordinates[0] + 2 && PbulletU[x][1] - 1 == enemyCoordinates[1] + 2))
        {
            printenemyunderweb(enemyCoordinates);
            Score++;
            return true;
        }
    }
    for (int x = 0; x < P_bulletCountD; x++)
    {
        if ((PbulletD[x][0] == enemyCoordinates[0] && PbulletD[x][1] + 1 == enemyCoordinates[1]) || (PbulletD[x][0] == enemyCoordinates[0] + 1 && PbulletD[x][1] + 1 == enemyCoordinates[1]) || (PbulletU[x][0] == enemyCoordinates[0] + 2 && PbulletU[x][1] + 1 == enemyCoordinates[1]))
        {
            printenemyunderweb(enemyCoordinates);
            Score++;
            return true;
        }
    }
    return false;
}
bool nameCheck(string word)
{
    int i = 0;
    while (i < word.length())
    {

        if ((word[i] > 64 && word[i] < 91) || (word[i] > 96 && word[i] < 123))
        {
            i++;
        }
        else
        {
            cout << "\n Invalid Input!!!";
            cout << "\n Name only contains aphabets..!!!\n ";
            getch();
            return false;
        }
    }
    return true;
}
bool nameExists(string name, string players[][7], int playercount)
{
    for (int i = 0; i < playercount; i++)
    {
        if (name == players[i][0])
        {
            cout << "\n Welcome Back!!! \n";
            cout << " Press any key to continue your Game...";
            getch();
            return true;
        }
    }
    return false;
}
int userindex(string name, string players[][7], int playerscount)
{
    for (int i = 0; i < playerscount; i++)
    {
        if (name == players[i][0])
        {
            return i;
        }
    }
}
void loadPlayerDataLevel1(string players[][7], int &playercount)
{
    fstream file;
    string line;
    file.open("level1Data.txt", ios::in);
    while (getline(file, line))
    {
        players[playercount][0] = getfield(line, 1); // name
        // other values like health,coordinates etc..
        players[playercount][1] = (getfield(line, 2));
        players[playercount][2] = (getfield(line, 3));
        players[playercount][3] = (getfield(line, 4));
        players[playercount][4] = (getfield(line, 5));
        players[playercount][5] = (getfield(line, 6));
        players[playercount][6] = (getfield(line, 7));
        playercount++;
    }
    file.close();
}
void storePlayerDataLevel1(string players[][7], int playercount)
{
    fstream file;
    file.open("level1Data.txt", ios::out);
    for (int i = 0; i < playercount; i++)
    {
        file << players[i][0] << ",";
        file << players[i][1] << ",";
        file << players[i][2] << ",";
        file << players[i][3] << ",";
        file << players[i][4] << ",";
        file << players[i][5] << ",";
        file << players[i][6] << ",";
        file << endl;
    }
    file.close();
}
void loadPlayerDataLevel2(string players[][7], int &playercount)
{
    fstream file;
    string line;
    file.open("level3Data.txt", ios::in);
    while (getline(file, line))
    {
        players[playercount][0] = getfield(line, 1); // name
        // other values like health,coordinates etc..
        players[playercount][1] = (getfield(line, 2));
        players[playercount][2] = (getfield(line, 3));
        players[playercount][3] = (getfield(line, 4));
        players[playercount][4] = (getfield(line, 5));
        players[playercount][5] = (getfield(line, 6));
        players[playercount][6] = (getfield(line, 7));
        playercount++;
    }
    file.close();
}
void storePlayerDataLevel2(string players[][7], int playercount)
{
    fstream file;
    file.open("level3Data.txt", ios::out);
    for (int i = 0; i < playercount; i++)
    {
        file << players[i][0] << ",";
        file << players[i][1] << ",";
        file << players[i][2] << ",";
        file << players[i][3] << ",";
        file << players[i][4] << ",";
        file << players[i][5] << ",";
        file << players[i][6] << ",";
        file << endl;
    }
    file.close();
}
void removePlayerfromArray(int index, int &playercount, string players[][7])
{
    playercount--;
    for (int i = 0; i < 7; i++)
    {
        players[index][i] = players[playercount][i];
        // players[index][1] = players[playercount][1];
        // players[index][2] = players[playercount][2];
        // players[index][3] = players[playercount][3];
        // players[index][4] = players[playercount][4];
        // players[index][5] = players[playercount][5];
        // players[index][6] = players[playercount][6];
    }
}
void printBox(int x, int y, int gotox, int gotoy)
{
    // x rows
    //  y coloms

    for (int i = 0; i < y; i++)
    {
        for (int j = 0; j < x; j++)
        {
            gotoxy(gotox + j, gotoy + i);
            if (i == 0 || i == y - 1 || j == 0 || j == x - 1)
            {
                cout << "#";
            }
            else
            {
                cout << " ";
            }
        }
    }
}
```