Dungeon Escape Game

Session: 2025 – 2029

Submitted by:

Saman Aslam 2025(S)-CS-78

Supervised by:

Mr. Laeeq Khan Niazi

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Contents

Project Description :

The **Dungeon Escape Game** is a console-based adventure game developed in C++ where a player navigates through three levels of a dungeon. The player must collect hidden keys to unlock doors and avoid moving enemies ("skeletons") and traps. The game challenges the player's strategic movement and skills.

Key gameplay features include:

- **Random traps** that reduce lives or teleport the player.
- **Multiple levels** with increasing difficulty.
- **Moving enemies** with distinct movement patterns.
- **Persistent scoring and life system** displayed in real-time.

Users of Application :

Player

- Can navigate the dungeon using arrow keys.
- Can collect keys and score points.
- Must avoid traps and skeletons.
- Can progress through levels by unlocking doors.
- Can view score, level, and lives during gameplay.
- Can end the game anytime with Escape key.

Functional Requirements

1. **Movement Controls**
    - Move player character using arrow keys (up, down, left, right).
    - Prevent movement into walls.
2. **Level Progression**

- Load new level when player reaches the door with required keys.

3. **Enemy AI**
    - Skeletons patrol horizontally and vertically.
    - Collision with skeletons reduces player lives.
4. **Traps**
    - Hidden traps revealed on contact.
    - Some traps deduct lives; others teleport player.
5. **Game States**
    - Win condition when all keys are collected and player exits.
    - Lose condition when lives reach zero.
6. **Display**
    - Show lives, level, score, and messages.
7. **File Handling**
    - Load maps for each level from text files.

Functions Working Flow :

Main Game Loop:

1. Load initial map.
2. Draw map and place player.
3. Loop while game active:
    - Check for level changes.
    - Move skeletons.
    - Detect collisions.
    - Update score and lives.
    - Respond to keyboard input.
    - End if win/loss condition met.

Skeleton Movement:

- Each skeleton alternates between horizontal and vertical patrol.
- On hitting walls or obstacles, they reverse direction.

Traps:

- 10 traps per game.
- On collision:
    - Type 0: Deduct life.
    - Type 1: Teleport to random safe location.

Functions Prototype

```
//.........functions prototypes
void gotoxy(int x, int y);
void printmap();
void loadmap1();
void loadmap2();
void loadmap3();
void moveleft();
void moveright();
void moveup();
void movedown();
void scoredisplay();
void scorecount(int x , int y);
void keycount();
void leveldisplay();
void levelandmap();
void livesdisplay();
void livescount();
bool gameover();
void skeleton1();
void skeleton2();
void skeleton3();
void skeleton4();
void positionreset();
void checktrap();
void clearmessage();
//.........prototypes end
```

Data Structures :

2D Array:

- `char map[20][63];` — Current level layout.

Player State:

- `int playerX, playerY;`
- `int score, lives, level, keys;`

Skeleton Positions:

- Separate variables for each skeleton's coordinates and movement state.

Traps:

- `trapX[]`, `trapY[]`, `trapType[]`, `trapRevealed[]`

Complete Code with Wireframes :

```
#include <iostream>
#include <windows.h>
#include <fstream>
using namespace std;

//.........functions prototypes
void gotoxy(int x, int y);
void printmap();
void loadmap1();
void loadmap2();
void loadmap3();
void moveleft();
void moveright();
void moveup();
void movedown();
void scoredisplay();
void scorecount(int x , int y);
void keycount();
void leveldisplay();
void levelandmap();
void livesdisplay();
void livescount();
bool gameover();
void skeleton1();
void skeleton2();
void skeleton3();
void skeleton4();
void positionreset();
void checktrap();
void clearmessage();
//.........prototypes end

//.........data structures
char map[20][63] = {};
int playerX = 2;
int playerY = 2;
int score = 0;
int lives = 5;
int level = 1;
```

```
int keys = 0;
int Skeleton1x = 2;
int Skeleton1y = 23;
bool skeleton1Horizontally = true;
bool skeleton1Vertically = true;
bool alternatemovement1 = true;
int Skeleton2x = 14;
int Skeleton2y = 33;
bool skeleton2Horizontally = true;
bool skeleton2Vertically = true;
bool alternatemovement2 = true;
int Skeleton3x = 7;
int Skeleton3y = 41;
bool skeleton3Horizontally = true;
bool skeleton3Vertically = true;
bool alternatemovement3 = true;
int Skeleton4x = 16;
int Skeleton4y = 10;
bool skeleton4Horizontally = true;
bool skeleton4Vertically = true;
bool alternatemovement4 = true;
const int TRAP_COUNT = 10;
int trapX[TRAP_COUNT] = {3, 5, 6, 8, 10, 12, 14, 15, 16, 18};
int trapY[TRAP_COUNT] = {6, 7, 18, 24, 30, 36, 42, 48, 52, 58};
int trapType[TRAP_COUNT] = {0, 0, 1, 1, 0, 0, 1, 0, 1, 0};
bool trapRevealed[TRAP_COUNT] = {false};
//.........data structures end

//.........main function
main()
{
    bool game = true;
    system("cls");
    loadmap1();
    system("Color 06");
    printmap();
    gotoxy(playerY,playerX);
    cout<<'P';
    while(game)
    {
        clearmessage();
```

```
        Sleep(100);
        levelandmap();
        if(level == 2)
        {
            skeleton3();
        }
        if(level == 3)
        {
            skeleton3();
            skeleton4();
        }
        scoredisplay();
        leveldisplay();
        livesdisplay();
        keycount();
        skeleton1();
        skeleton2();
        livescount();
        if(gameover())
        {
            game = false;
        }
        if (GetAsyncKeyState(VK_LEFT))
        {
            moveleft();
        }
        if (GetAsyncKeyState(VK_RIGHT))
        {
            moveright();
        }
        if (GetAsyncKeyState(VK_UP))
        {
            moveup();
        }
        if (GetAsyncKeyState(VK_DOWN))
        {
            movedown();
        }
        if (GetAsyncKeyState(VK_ESCAPE))
        {
            system("cls");
```

```
            gotoxy(20 , 5);
            cout<<"Game Over!";
            gotoxy(20 , 8);
            cout<<"Total score : "<<score;
            game = false; // Stop the game
        }
    }
}
//.........main end

//.........functions implementation
void gotoxy(int x, int y)
{
COORD coordinates;
coordinates.X = x;
coordinates.Y = y;
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinates);
}
void printmap()
{
    for (int r = 0; r < 20; r++)
    {
        for (int c = 0; c < 63; c++)
        {
            bool isTrapHere = false;
            for (int t = 0; t < TRAP_COUNT; t++)
            {
                if (r == trapX[t] && c == trapY[t] && trapRevealed[t])
                {
                    cout << '@';
                    isTrapHere = true;
                    break;
                }
            }
            if (!isTrapHere)
            {
                cout << map[r][c];
            }
        }
        cout << endl;
    }
```

```
}
void loadmap1()
{
    fstream read;
    string line;
    read.open("maplvl1.txt",ios::in);
    for(int i = 0 ; i < 20 ; i ++)
    {
        getline(read , line);
        for(int j = 0 ; j < 63 ; j ++)
        {
            map[i][j] = line[j];
        }
    }
    read.close();
}
void loadmap2()
{
    fstream read;
    string line;
    read.open("maplvl2.txt",ios::in);
    for(int i = 0 ; i < 20 ; i ++)
    {
        getline(read , line);
        for(int j = 0 ; j < 63 ; j ++)
        {
            map[i][j] = line[j];
        }
    }
    read.close();
    for (int t = 0; t < TRAP_COUNT; t++)
    {
        trapRevealed[t] = false;
    }
}
void loadmap3()
{
    fstream read;
    string line;
    read.open("maplvl3.txt",ios::in);
    for(int i = 0 ; i < 20 ; i ++)
```

```
    {
        getline(read , line);
        for(int j = 0 ; j < 63 ; j ++)
        {
            map[i][j] = line[j];
        }
    }
    read.close();
    for (int t = 0; t < TRAP_COUNT; t++)
    {
        trapRevealed[t] = false;
    }
}
void moveleft()
{
    if(map[playerX][playerY-1] != '#')
    {
        gotoxy(playerY,playerX);
        cout<<' ';
        playerY = playerY - 1;
        checktrap();
        gotoxy(playerY,playerX);
        cout<<'P';
        scorecount(playerX,playerY);
    }
}
void moveright()
{
    if(map[playerX][playerY+1] != '#')
    {
        gotoxy(playerY,playerX);
        cout<<' ';
        playerY = playerY + 1;
        checktrap();
        gotoxy(playerY,playerX);
        cout<<'P';
        scorecount(playerX,playerY);
    }
}
void moveup()
{
```

```
    if(map[playerX-1][playerY] != '#')
    {
        gotoxy(playerY,playerX);
        cout<<' ';
        playerX = playerX - 1;
        checktrap();
        gotoxy(playerY,playerX);
        cout<<'P';
        scorecount(playerX,playerY);
    }
}
void movedown()
{
    if(map[playerX+1][playerY] != '#')
    {
        gotoxy(playerY,playerX);
        cout<<' ';
        playerX = playerX + 1;
        checktrap();
        gotoxy(playerY,playerX);
        cout<<'P';
        scorecount(playerX,playerY);
    }
}
void scorecount(int x , int y)
{
    if(map[x][y] == '*')
    {
        score ++ ;
        map[x][y] = ' ';
    }
}
void scoredisplay()
{
    gotoxy( 66 , 15);
    cout<<"Score : "<<score;
}
void keycount()
{
    if(map[playerX][playerY] == '%')
    {
```

```
        map[playerX][playerY]=' ';
        keys = keys + 1;
        clearmessage();
        gotoxy(66 , 19);
        cout<<"Key collected!";
    }
}
void leveldisplay()
{
    gotoxy( 66 , 5);
    cout<<"Level : "<<level;
}
void levelandmap()
{
    if(keys >= 1 && keys < 3 && (map[playerX][playerY] == 'D' || map[playerX +
1][playerY] == 'D' || map[playerX - 1][playerY] == 'D' ||map[playerX][playerY + 1] ==
'D' ||map[playerX][playerY - 1] == 'D') )
    {
        system("cls");
        level ++ ;
        loadmap2();
        system("Color 05");
        printmap();
        positionreset();
    }
    else if(keys >= 3 && keys < 6 && (map[playerX][playerY] == 'D' || map[playerX +
1][playerY] == 'D' || map[playerX - 1][playerY] == 'D' ||map[playerX][playerY + 1] ==
'D' ||map[playerX][playerY - 1] == 'D') )
    {
        system("cls");
        level ++ ;
        loadmap3();
        system("Color 04");
        printmap();
        positionreset();
    }
}
bool gameover()
{
    if(keys == 6 && (map[playerX][playerY] == 'D' || map[playerX + 1][playerY] == 'D'
|| map[playerX - 1][playerY] == 'D' ||map[playerX][playerY + 1] == 'D'
```

```
||map[playerX][playerY - 1] == 'D') )
    {
        system("cls");
        gotoxy(20 , 5);
        cout<<"WIN!!"<<endl<<"Collected all keys."<<endl<<"Hurraayyy!";
        gotoxy(20 , 8);
        cout<<"Total score : "<<score;
        return true;
    }
    if(lives == 0)
    {
        system("cls");
        gotoxy(20 , 5);
        cout<<"Game Over!";
        gotoxy(20 , 8);
        cout<<"Total score : "<<score;
        return true;
    }
    return false;
}
void livesdisplay()
{
    gotoxy( 66 , 10);
    cout<<"Lives : "<<lives<<endl;
}
void livescount()
{
    if((playerX == Skeleton1x && playerY == Skeleton1y) ||(playerX == Skeleton2x &&
playerY == Skeleton2y) || (playerX == Skeleton3x && playerY == Skeleton3y) || (playerX
== Skeleton4x && playerY == Skeleton4y))
    {
        lives = lives - 1;

        positionreset();
    }
}
void positionreset()
{
    if(map[playerX][playerY] != '$')
    {
        gotoxy(playerY , playerX);
```

```
        cout<<' ';
        map[playerX][playerY] = ' ';
    }
    playerX = 2 ;
    playerY = 2 ;
    gotoxy(playerY , playerX);
    cout<<'P';
    map[playerX][playerY] = 'P';
}
void skeleton1()
{
    if(alternatemovement1)
    {
        if(map[Skeleton1x][Skeleton1y + 1] == '#' || map[Skeleton1x][Skeleton1y + 1]
== '|' || map[Skeleton1x][Skeleton1y + 1] == 'D'|| map[Skeleton1x][Skeleton1y + 1] ==
'_' )
        {
            skeleton1Horizontally = !skeleton1Horizontally;
        }
        else if(map[Skeleton1x][Skeleton1y - 1] == '#' || map[Skeleton1x][Skeleton1y -
1] == '|' || map[Skeleton1x][Skeleton1y - 1] == 'D' || map[Skeleton1x][Skeleton1y - 1]
== '_' )
        {
            skeleton1Horizontally = !skeleton1Horizontally;
        }
        if(skeleton1Horizontally)
        {
            if(map [Skeleton1x][Skeleton1y + 1] != '#' && map[Skeleton1x][Skeleton1y +
1] != '|' && map[Skeleton1x][Skeleton1y + 1] != 'D' && map[Skeleton1x][Skeleton1y + 1]
!= '_')
            {
                gotoxy( Skeleton1y ,Skeleton1x );
                cout<<map[Skeleton1x][Skeleton1y];
                Skeleton1y = Skeleton1y + 1;
                gotoxy(Skeleton1y , Skeleton1x);
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton1x][Skeleton1y - 1] != '#' && map[Skeleton1x][Skeleton1y -
```

```
1] != '|' && map[Skeleton1x][Skeleton1y - 1] != 'D' && map[Skeleton1x][Skeleton1y - 1]
!= '_')
            {
                gotoxy(Skeleton1y ,Skeleton1x);
                cout<<map[Skeleton1x][Skeleton1y];
                Skeleton1y = Skeleton1y - 1;
                gotoxy(Skeleton1y,Skeleton1x);
                cout<< '$';
            }
        }
    }

    else
    {
        if(map[Skeleton1x - 1][Skeleton1y ] == '#' || map[Skeleton1x - 1][Skeleton1y]
== '|' || map[Skeleton1x - 1][Skeleton1y] == 'D' || map[Skeleton1x - 1][Skeleton1y] ==
'_')
        {
            skeleton1Vertically = !skeleton1Vertically;
        }
        else if(map[Skeleton1x + 1][Skeleton1y ] == '#' || map[Skeleton1x +
1][Skeleton1y] == '|' || map[Skeleton1x + 1][Skeleton1y] == 'D'|| map[Skeleton1x +
1][Skeleton1y] == '_')
        {
            skeleton1Vertically = !skeleton1Vertically;
        }
        if(skeleton1Vertically)
        {
            if(map[Skeleton1x - 1][Skeleton1y] != '#' && map[Skeleton1x -
1][Skeleton1y] != '|' && map[Skeleton1x - 1][Skeleton1y] != 'D' && map[Skeleton1x -
1][Skeleton1y] != '_' )
            {
                gotoxy(Skeleton1y, Skeleton1x);
                cout<<map[Skeleton1x][Skeleton1y];
                Skeleton1x = Skeleton1x - 1;
                gotoxy(Skeleton1y , Skeleton1x);
                cout<< '$';
            }
        }
        else
        {
```

```
            if(map[Skeleton1x + 1][Skeleton1y] != '#' && map[Skeleton1x +
1][Skeleton1y] != '|' && map[Skeleton1x + 1][Skeleton1y] != 'D' && map[Skeleton1x +
1][Skeleton1y] != '_')
            {
                gotoxy(Skeleton1y , Skeleton1x);
                cout<<map[Skeleton1x][Skeleton1y];
                Skeleton1x = Skeleton1x + 1;
                gotoxy(Skeleton1y ,Skeleton1x);
                cout<< '$';
            }
        }
    }

    alternatemovement1 = !alternatemovement1;
}
void skeleton2()
{
    if(alternatemovement2)
    {
        if(map[Skeleton2x][Skeleton2y + 1] == '#' || map[Skeleton2x][Skeleton2y + 1]
== '|' || map[Skeleton2x][Skeleton2y + 1] == 'D' || map[Skeleton2x][Skeleton2y + 1] ==
'_'  )
        {
            skeleton2Horizontally = !skeleton2Horizontally;
        }
        else if(map[Skeleton2x][Skeleton2y - 1] == '#' || map[Skeleton2x][Skeleton2y -
1] == '|' || map[Skeleton2x][Skeleton2y - 1] == 'D'  || map[Skeleton2x][Skeleton2y -
1] == '_' )
        {
            skeleton2Horizontally = !skeleton2Horizontally;
        }
        if(skeleton2Horizontally)
        {
            if(map [Skeleton2x][Skeleton2y + 1] != '#' && map[Skeleton2x][Skeleton2y +
1] != '|' && map[Skeleton2x][Skeleton2y + 1] != 'D' && map[Skeleton2x][Skeleton2y + 1]
!= '_')
            {
                gotoxy( Skeleton2y ,Skeleton2x );
                cout<<map[Skeleton2x][Skeleton2y];
                Skeleton2y = Skeleton2y + 1;
                gotoxy(Skeleton2y , Skeleton2x);
```

```
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton2x][Skeleton2y- 1] != '#' && map[Skeleton2x][Skeleton2y -
1] != '|' && map[Skeleton2x][Skeleton2y - 1] != 'D' && map[Skeleton2x][Skeleton2y - 1]
!= '_' )
            {
                gotoxy(Skeleton2y ,Skeleton2x);
                cout<<map[Skeleton2x][Skeleton2y];
                Skeleton2y = Skeleton2y - 1;
                gotoxy(Skeleton2y,Skeleton2x);
                cout<< '$';
            }
        }
    }

    else
    {
        if(map[Skeleton2x - 1][Skeleton2y ] == '#' || map[Skeleton2x - 1][Skeleton2y ]
== '|' || map[Skeleton2x - 1][Skeleton2y ] == '_' || map[Skeleton2x - 1][Skeleton2y ]
== 'D' )
        {
            skeleton2Vertically = !skeleton2Vertically;
        }
        else if(map[Skeleton2x + 1][Skeleton2y ] == '#' || map[Skeleton2x +
1][Skeleton2y ] == '_' || map[Skeleton2x + 1][Skeleton2y ] == '|' || map[Skeleton2x +
1][Skeleton2y ] == 'D')
        {
            skeleton2Vertically = !skeleton2Vertically;
        }
        if(skeleton2Vertically)
        {
            if(map[Skeleton2x - 1][Skeleton2y] != '#' && map[Skeleton2x -
1][Skeleton2y ] != '|' && map[Skeleton2x - 1][Skeleton2y ] != '_' && map[Skeleton2x -
1][Skeleton2y ] != 'D')
            {
                gotoxy(Skeleton2y, Skeleton2x);
                cout<<map[Skeleton2x][Skeleton2y];
                Skeleton2x = Skeleton2x - 1;
```

```
                gotoxy(Skeleton2y , Skeleton2x);
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton2x + 1][Skeleton2y] != '#' && map[Skeleton2x +
1][Skeleton2y ] != '_' && map[Skeleton2x + 1][Skeleton2y ] != '|' && map[Skeleton2x +
1][Skeleton2y ] != 'D')
            {
                gotoxy(Skeleton2y , Skeleton2x);
                cout<<map[Skeleton2x][Skeleton2y];
                Skeleton2x = Skeleton2x + 1;
                gotoxy(Skeleton2y ,Skeleton2x);
                cout<< '$';
            }
        }
    }

    alternatemovement2 = !alternatemovement2;
}
void skeleton3()
{
    if(alternatemovement3)
    {
        if(map[Skeleton3x][Skeleton3y + 1] == '#' || map[Skeleton3x][Skeleton3y + 1]
== '|' || map[Skeleton3x][Skeleton3y + 1] == 'D' || map[Skeleton3x][Skeleton3y + 1] ==
'_')
        {
            skeleton3Horizontally = !skeleton3Horizontally;
        }
        else if(map[Skeleton3x][Skeleton3y - 1] == '#' || map[Skeleton3x][Skeleton3y -
1] == '|'  || map[Skeleton3x][Skeleton3y - 1] == 'D' || map[Skeleton3x][Skeleton3y -
1] == '_' )
        {
            skeleton3Horizontally = !skeleton3Horizontally;
        }
        if(skeleton3Horizontally)
        {
            if(map [Skeleton3x][Skeleton3y + 1] != '#' && map[Skeleton3x][Skeleton3y +
1] != '|' && map[Skeleton3x][Skeleton3y + 1] != 'D' && map[Skeleton3x][Skeleton3y + 1]
```

```
!= '_' )
            {
                gotoxy( Skeleton3y ,Skeleton3x );
                cout<<map[Skeleton3x][Skeleton3y];
                Skeleton3y = Skeleton3y + 1;
                gotoxy(Skeleton3y , Skeleton3x);
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton3x][Skeleton3y- 1] != '#' && map[Skeleton3x][Skeleton3y -
1] != '|' && map[Skeleton3x][Skeleton3y - 1] != 'D' && map[Skeleton3x][Skeleton3y - 1]
!= '_' )
            {
                gotoxy(Skeleton3y ,Skeleton3x);
                cout<<map[Skeleton3x][Skeleton3y];
                Skeleton3y = Skeleton3y - 1;
                gotoxy(Skeleton3y,Skeleton3x);
                cout<< '$';
            }
        }
    }

    else
    {
        if(map[Skeleton3x - 1][Skeleton3y ] == '#' || map[Skeleton3x - 1][Skeleton3y ]
== '|' ||map[Skeleton3x - 1][Skeleton3y ] == '_' ||map[Skeleton3x - 1][Skeleton3y ] ==
'D')
        {
            skeleton3Vertically = !skeleton3Vertically;
        }
        else if(map[Skeleton3x + 1][Skeleton3y ] == '#' || map[Skeleton3x +
1][Skeleton3y ] == '|' || map[Skeleton3x + 1][Skeleton3y ] == '_' || map[Skeleton3x +
1][Skeleton3y ] == 'D')
        {
            skeleton3Vertically = !skeleton3Vertically;
        }
        if(skeleton3Vertically)
        {
            if(map[Skeleton3x - 1][Skeleton3y] != '#' && map[Skeleton3x -
```

```
1][Skeleton3y] != '|' && map[Skeleton3x - 1][Skeleton3y] != '_' && map[Skeleton3x -
1][Skeleton3y] != 'D')
            {
                gotoxy(Skeleton3y, Skeleton3x);
                cout<<map[Skeleton3x][Skeleton3y];
                Skeleton3x = Skeleton3x - 1;
                gotoxy(Skeleton3y , Skeleton3x);
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton3x + 1][Skeleton3y] != '#' && map[Skeleton3x +
1][Skeleton3y] != '|' && map[Skeleton3x + 1][Skeleton3y] != '_' && map[Skeleton3x +
1][Skeleton3y] != 'D')
            {
                gotoxy(Skeleton3y , Skeleton3x);
                cout<<map[Skeleton3x][Skeleton3y];
                Skeleton3x = Skeleton3x + 1;
                gotoxy(Skeleton3y ,Skeleton3x);
                cout<< '$';
            }
        }
    }

    alternatemovement3 = !alternatemovement3;
}
void skeleton4()
{
    if(alternatemovement4)
    {
        if(map[Skeleton4x][Skeleton4y + 1] == '#' || map[Skeleton4x][Skeleton4y + 1]
== '|' || map[Skeleton4x][Skeleton4y + 1] == 'D' || map[Skeleton4x][Skeleton4y + 1] ==
'_'  )
        {
            skeleton4Horizontally = !skeleton4Horizontally;
        }
        else if(map[Skeleton4x][Skeleton4y - 1] == '#' || map[Skeleton4x][Skeleton4y -
1] == '|' || map[Skeleton4x][Skeleton4y - 1] == 'D'  || map[Skeleton4x][Skeleton4y -
1] == '_' )
        {
```

```
            skeleton4Horizontally = !skeleton4Horizontally;
        }
        if(skeleton4Horizontally)
        {
            if(map [Skeleton4x][Skeleton4y + 1] != '#' && map[Skeleton4x][Skeleton4y +
1] != '|' && map[Skeleton4x][Skeleton4y + 1] != 'D' && map[Skeleton4x][Skeleton4y + 1]
!= '_')
            {
                gotoxy( Skeleton4y ,Skeleton4x );
                cout<<map[Skeleton4x][Skeleton4y];
                Skeleton4y = Skeleton4y + 1;
                gotoxy(Skeleton4y , Skeleton4x);
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton4x][Skeleton4y- 1] != '#' && map[Skeleton4x][Skeleton4y -
1] != '|' && map[Skeleton4x][Skeleton4y - 1] != 'D' && map[Skeleton4x][Skeleton4y - 1]
!= '_' )
            {
                gotoxy(Skeleton4y ,Skeleton4x);
                cout<<map[Skeleton4x][Skeleton4y];
                Skeleton4y = Skeleton4y - 1;
                gotoxy(Skeleton4y,Skeleton4x);
                cout<< '$';
            }
        }
    }

    else
    {
        if(map[Skeleton4x - 1][Skeleton4y ] == '#' || map[Skeleton4x - 1][Skeleton4y ]
== '|' || map[Skeleton4x - 1][Skeleton4y ] == 'D' || map[Skeleton4x - 1][Skeleton4y ]
== '_' )
        {
            skeleton4Vertically = !skeleton4Vertically;
        }
        else if(map[Skeleton4x + 1][Skeleton4y ] == '#' || map[Skeleton4x +
1][Skeleton4y ] == '|' || map[Skeleton4x + 1][Skeleton4y ] == 'D' || map[Skeleton4x +
1][Skeleton4y ] == '_')
```

```
        {
            skeleton4Vertically = !skeleton4Vertically;
        }
        if(skeleton4Vertically)
        {
            if(map[Skeleton4x - 1][Skeleton4y] != '#' && map[Skeleton4x -
1][Skeleton4y] != 'D' && map[Skeleton4x - 1][Skeleton4y] != '|' && map[Skeleton4x -
1][Skeleton4y] != '_')
            {
                gotoxy(Skeleton4y, Skeleton4x);
                cout<<map[Skeleton4x][Skeleton4y];
                Skeleton4x = Skeleton4x - 1;
                gotoxy(Skeleton4y , Skeleton4x);
                cout<< '$';
            }
        }
        else
        {
            if(map[Skeleton4x + 1][Skeleton4y] != '#' && map[Skeleton4x +
1][Skeleton4y] != 'D' && map[Skeleton4x + 1][Skeleton4y] != '|' && map[Skeleton4x +
1][Skeleton4y] != '_')
            {
                gotoxy(Skeleton4y , Skeleton4x);
                cout<<map[Skeleton4x][Skeleton4y];
                Skeleton4x = Skeleton4x + 1;
                gotoxy(Skeleton4y ,Skeleton4x);
                cout<< '$';
            }
        }
    }

    alternatemovement4 = !alternatemovement4;
}
void checktrap()
{
    for (int t = 0; t < TRAP_COUNT; t++)
    {
        if (playerX == trapX[t] && playerY == trapY[t] && !trapRevealed[t])
        {
            trapRevealed[t] = true;
            gotoxy(trapY[t], trapX[t]);
```

```
            cout << '@';
            if (trapType[t] == 0)
            {
                lives--;
                clearmessage();
                gotoxy(66, 19);
                cout << "Trap! Lost 1 life.";
                Sleep(800);
            }
            else if (trapType[t] == 1)
            {
                clearmessage();
                gotoxy(66, 19);
                cout << "Teleport trap!";
                Sleep(800);
                int newplayerX[3] = {2, 5, 8};
                int newplayerY[3] = {2, 10, 20};
                int index = rand() % 3;
                gotoxy(playerY, playerX);
                cout << ' ';
                playerX = newplayerX[index];
                playerY = newplayerY[index];
                gotoxy(playerY, playerX);
                cout << 'P';
            }
        }
    }
}
void clearmessage()
{
    gotoxy(66, 19);
    cout << "                              ";
}
//.........implementations end
```

Game Wireframes :

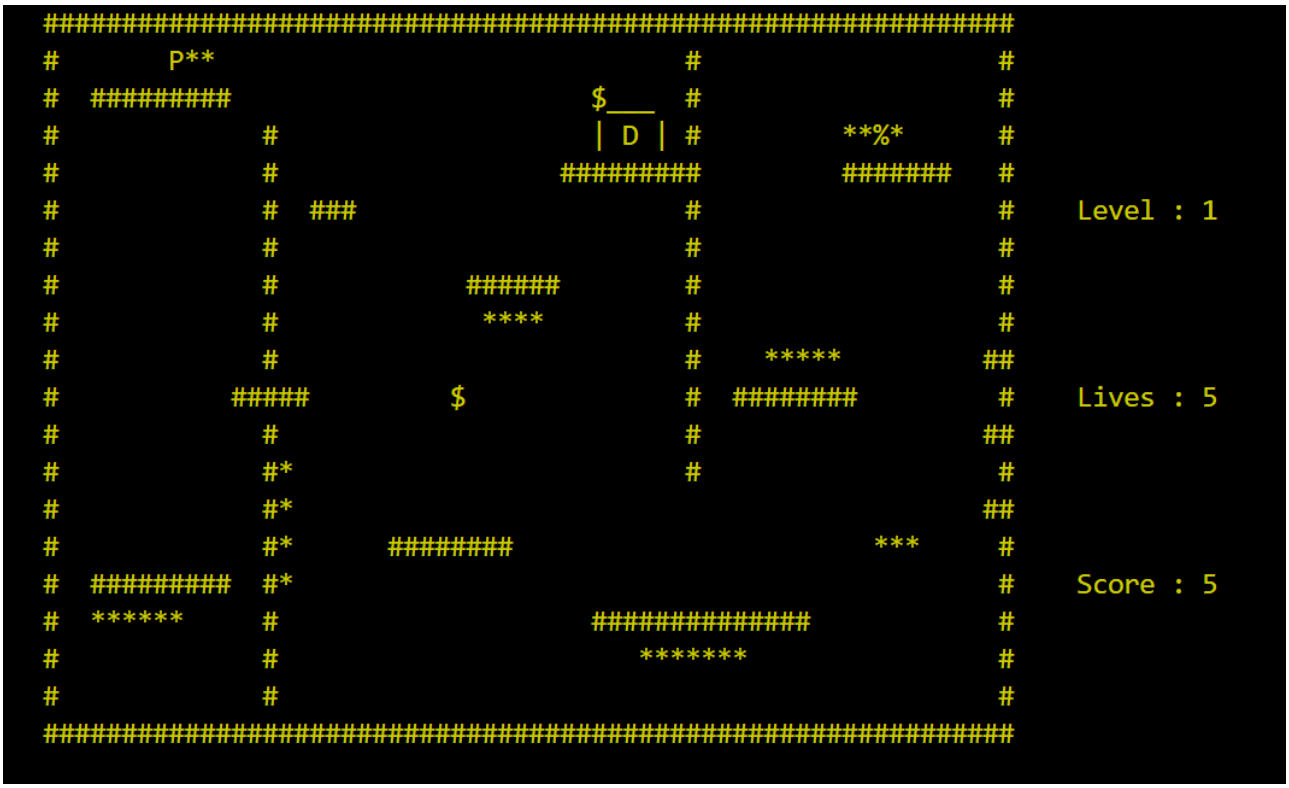

Fig: Level 1

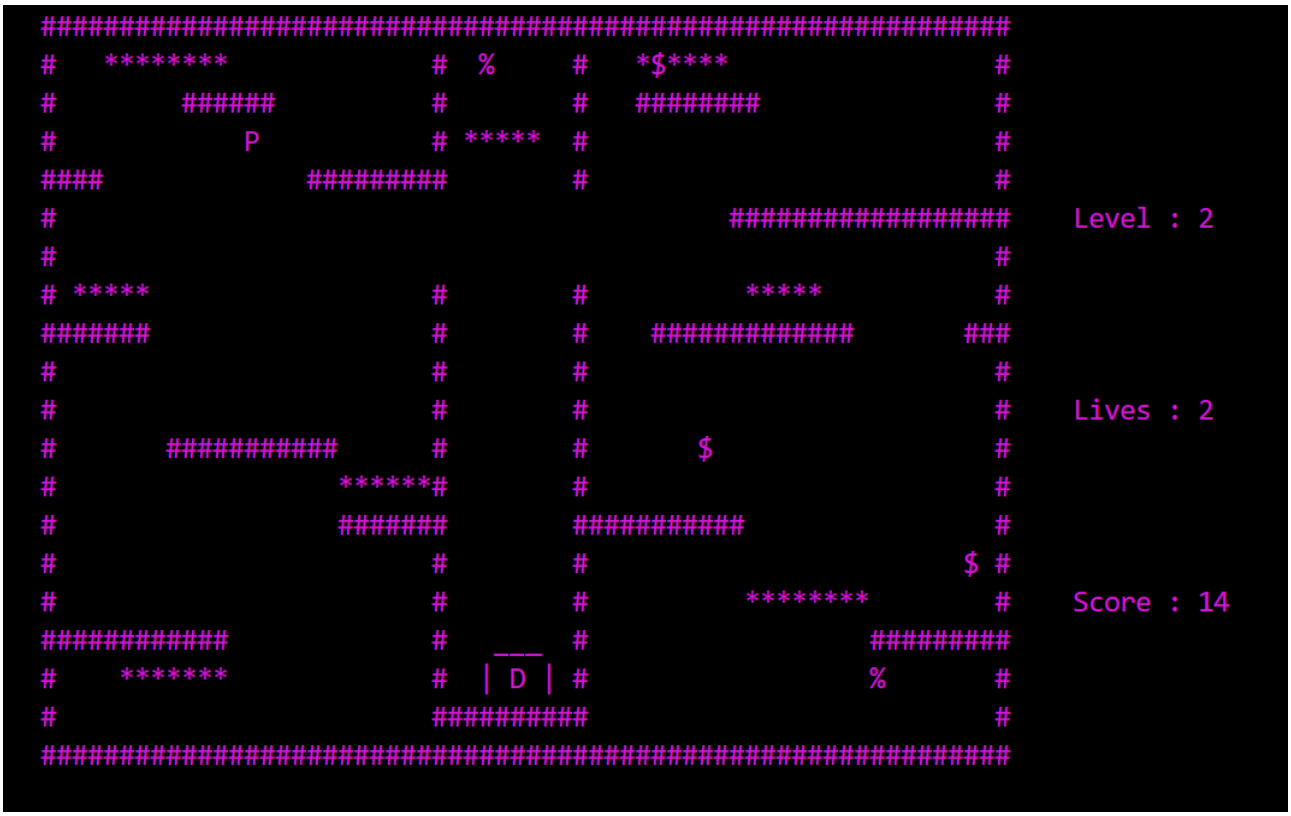

Fig : Level 2

Fig : Level 3

Fig : Game Over

Test Cases :

Test Case A: Level Progression

Description:
Player collects required keys and advances through levels.

Steps:

1. Start game.
2. Collect 1–3 keys.
3. Reach door tile (D).
4. Verify next level loads.

Expected Result:
New map loaded, player position reset.

Test Case B: Trap Collision

Description:
Player steps on hidden trap.

Steps:

1. Move to known trap location.
2. Observe lives decrement or teleport.

Expected Result:
Trap revealed, lives adjusted or teleport triggered.

Test Case C: Skeleton Collision

Description:
Player collides with a skeleton.

Steps:

1. Allow skeleton to move into player.
2. Observe lives decrement and player reset.

Expected Result:
Life decremented by 1, player position reset.

Conclusion and Future Improvements

The **Dungeon Escape Game** successfully demonstrates an interactive console game in C++ using:

- 2D arrays for map handling,
- Dynamic AI behavior for enemies,
- File input/output for map storage,
- Real-time user input processing.

Future Enhancements:

- Sound effects and background music.
- Graphical interface using libraries like SFML.
- Save/load game progress.
- More complex enemy behaviors.
- Additional trap types and power-ups.

Student Reg. No. : 2025(S)-CS-78 **Student Name.** Saman Aslam

	A-Extensive Evidence	**B-Convincing Evidence**	**C-Limited Evidence**	**D-No Evidence**
Documentation Formatting **Grade:**	A			
Documentation Formatting Criteria: In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to giv **guidelines**. Project **Poster** is professionally design and well presented				
Documentation Contents **Grade:**	A			
Documentation Contents Criteria: Title Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Da** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is you Planning.				
Project Complexity **Grade:**	A			
Code Style **Grade:**	A			
Code Style Criteria: Consistent code style. Code is well indented. Variable and Function names are well defined. White Spaces are well used. Comments are added.				
Code Documentation Mapping **Grade:**	A			
Data Structure (Arrays) **Grade:**	A			
Sorting Features **Grade:**	A			
Modularity **Grade:**	A			
Modularity criteria: Functions are defined for each major feature. Functions are independent (identify from parameter list and return types Demo Data Functionality Added-At least Two Unit Tests are defined.				
Validations **Grade:**		B		
Recommendation Feature	A			
Presentation and Demo **Grade:**	A			
Student Understanding with the Code. **Grade:**	A			
Checked by:	Self Check			

Student Reg. No. : 2025(S)-CS-78 **Student Name: Saman Aslam**

	A-Extensive Evidence	B-Convincing Evidence	C-Limited Evidence	D-No Evidence
Documentation Formatting **Grade:**				
Documentation Formatting Criteria: In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to giv **guidelines**. Project **Poster** is professionally design and well presented				
Documentation Contents **Grade:**				
Documentation Contents Criteria: Title Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Da** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is you Planning.				
Project Complexity **Grade:**				
Code Style **Grade:**				
Code Style Criteria: Consistent code style. Code is well indented. Variable and Function names are well defined. White Spaces are well used. Comments are added.				
Code Documentation Mapping **Grade:**				
Data Structure (Arrays) **Grade:**				
Sorting Features **Grade:**				
Modularity **Grade:**				
Modularity criteria: Functions are defined for each major feature. Functions are independent (identify from parameter list and return types Demo Data Functionality Added-At least Two Unit Tests are defined.				
Validations **Grade:**				
Recommendation Feature				
Presentation and Demo **Grade:**				
Student Understanding with the Code. **Grade:**				
Checked by:	Click or tap here to enter text.			