**Thief and Police chase down Game**



Session: 2021

**Submitted by:**

Muhammad Umair Shahid 2021-CS-144

**Supervised by:**

Ms. Maida Shahid

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Short Description**

This is a game with a thief and Policemen. Police has to chase the thief.

**Game Characters**

There is one thief which is acting as a player. There are 3 Policemen which have to caught the thief and one of them has a function to kill thief. When thief is close to this policeman, the policeman starts to fire. Remaining, one is appearing randomly from maze from left side and other is from right side. Thief has specific time to save itself from these Police officers.

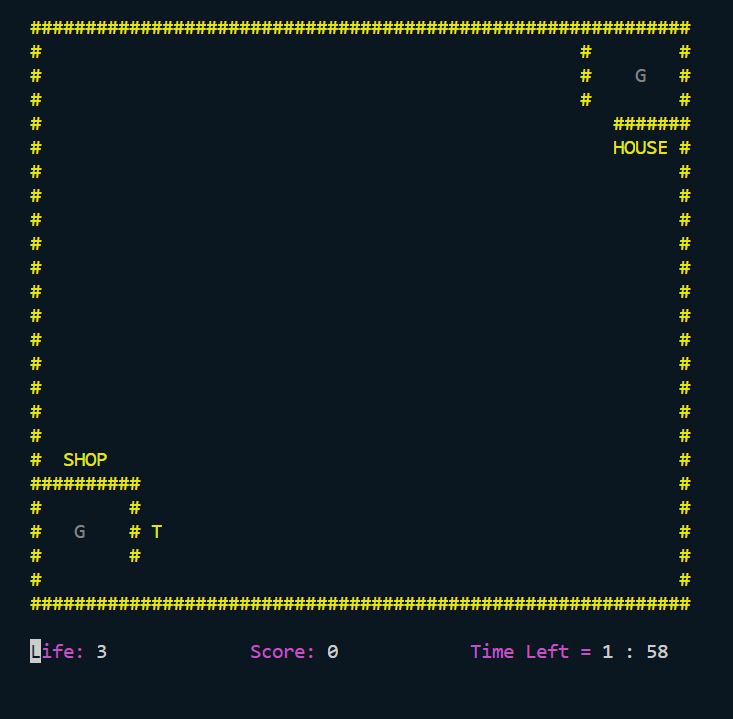
**Rules & Interactions**

* Thief is control by right, left, up and down keys.
* On the sides there are Obstacles.
* If fire by specific policeman is hit to the thief, life will decrease.
* If game is complete in specific time than specific score is allocated to the player.
* There is specific time to complete game otherwise game will win by the thief.
* Players can play game 3 times.

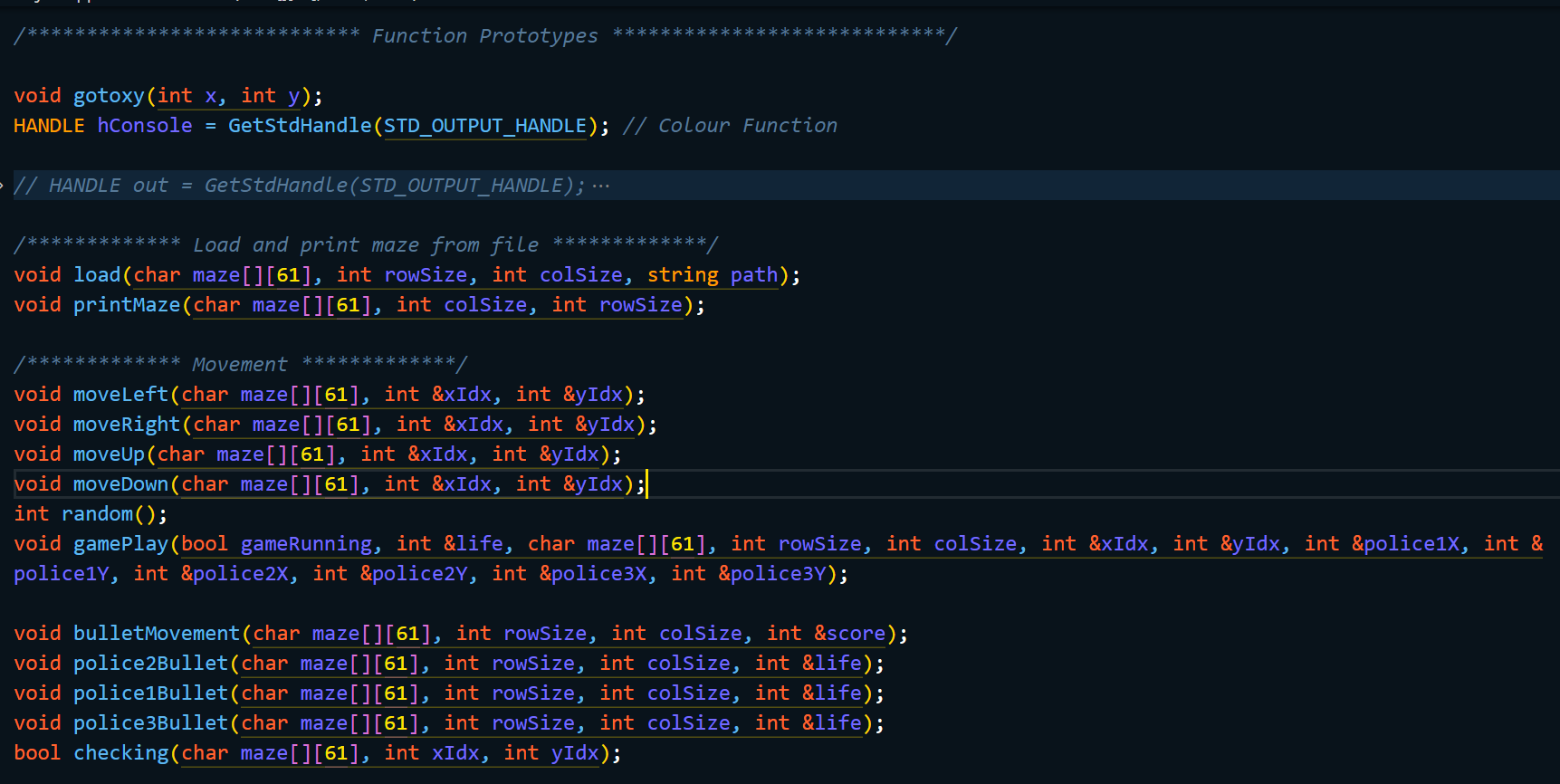
**Goal of the Game**

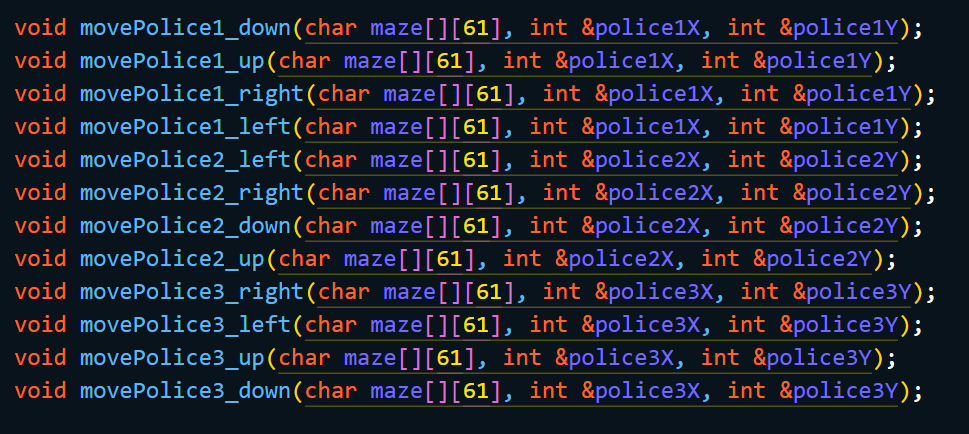
Goal of the game is to save and complete game as early as possible and get high scores.

**Main Screen**



**Functions Prototypes**





**Complete Code**

#include <iostream> // input output

#include <windows.h> // gotoxy,colour

#include <fstream> // File Handling

#include <ctime> // Srand function

#include <conio.h> // system ClS , getch

using namespace std;

/\*\*\*\*\*\*\*\*\*\* Function Prototypes \*\*\*\*\*\*\*\*\*\*/

void gotoxy(int x, int y);

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE); // Colour Function

// HANDLE out = GetStdHandle(STD\_OUTPUT\_HANDLE);

// CONSOLE\_CURSOR\_INFO cursorInfo;

// GetConsoleCursorInfo(out, &cursorInfo);

// cursorInfo.bVisible = false; // set the cursor visibility

// SetConsoleCursorInfo(out, &cursorInfo);

/\*\*\*\*\* Load and print maze from file \*\*\*\*\*/

void load(char maze[][61], int rowSize, int colSize, string path);

void printMaze(char maze[][61], int colSize, int rowSize);

/\*\*\*\*\* Movement \*\*\*\*\*/

void moveLeft(char maze[][61], int &xIdx, int &yIdx);

void moveRight(char maze[][61], int &xIdx, int &yIdx);

void moveUp(char maze[][61], int &xIdx, int &yIdx);

void moveDown(char maze[][61], int &xIdx, int &yIdx);

int random();

void gamePlay(bool gameRunning, int &life, char maze[][61], int rowSize, int colSize, int &xIdx, int &yIdx, int &police1X, int &police1Y, int &police2X, int &police2Y, int &police3X, int &police3Y);

void bulletMovement(char maze[][61], int rowSize, int colSize, int &score);

void police2Bullet(char maze[][61], int rowSize, int colSize, int &life);

void police1Bullet(char maze[][61], int rowSize, int colSize, int &life);

void police3Bullet(char maze[][61], int rowSize, int colSize, int &life);

bool checking(char maze[][61], int xIdx, int yIdx);

void movePolice1\_down(char maze[][61], int &police1X, int &police1Y);

void movePolice1\_up(char maze[][61], int &police1X, int &police1Y);

void movePolice1\_right(char maze[][61], int &police1X, int &police1Y);

void movePolice1\_left(char maze[][61], int &police1X, int &police1Y);

void movePolice2\_left(char maze[][61], int &police2X, int &police2Y);

void movePolice2\_right(char maze[][61], int &police2X, int &police2Y);

void movePolice2\_down(char maze[][61], int &police2X, int &police2Y);

void movePolice2\_up(char maze[][61], int &police2X, int &police2Y);

void movePolice3\_right(char maze[][61], int &police3X, int &police3Y);

void movePolice3\_left(char maze[][61], int &police3X, int &police3Y);

void movePolice3\_up(char maze[][61], int &police3X, int &police3Y);

void movePolice3\_down(char maze[][61], int &police3X, int &police3Y);

int score = 0;

bool Eat = false;

int thing1X = 21;

int thing1Y = 4;

int thing2X = 2;

int thing2Y = 55;

/\*\*\*\*\*\*\*\*\*\* Main Function \*\*\*\*\*\*\*\*\*\*/

main()

{

char maze[25][61]; // 2d array for maze

string path = "maze.txt";

int rowSize = sizeof(maze) / sizeof(maze[0]);

int colSize = sizeof(maze[0]) / sizeof(maze[0][0]);

load(maze, rowSize, colSize, path);

int xIdx = 21; // Printing Player at specific position

int yIdx = 11;

maze[xIdx][yIdx] = 'T';

int police1X = 2; // Printing Police at specific position

int police1Y = 49;

int police2X = 14; // Printing Police at specific position

int police2Y = 1;

int police3X = 12; // Printing Police at specific position

int police3Y = 58;

system("CLS");

SetConsoleTextAttribute(hConsole, 6); // Yellow

printMaze(maze, colSize, rowSize); // End of Load and Printing objects and maze

SetConsoleTextAttribute(hConsole, 7); // Whtie

maze[thing1X][thing1Y] = 'G';

SetConsoleTextAttribute(hConsole, 8); // Gray

gotoxy(thing1Y, thing1X);

cout << "G";

maze[thing2X][thing2Y] = 'G';

SetConsoleTextAttribute(hConsole, 8); // Gray

gotoxy(thing2Y, thing2X);

cout << "G";

int life = 3;

gamePlay(1, life, maze, rowSize, colSize, xIdx, yIdx, police1X, police1Y, police2X, police2Y, police3X, police3Y);

if (life == 0)

{

SetConsoleTextAttribute(hConsole, 5); // Purple

gotoxy(0, 26);

cout << "Life: ";

SetConsoleTextAttribute(hConsole, 7); // Whtie

cout << "0";

SetConsoleTextAttribute(hConsole, 2); // Green

gotoxy(70, 13);

cout << "YOU LOSE!!!";

getch();

SetConsoleTextAttribute(hConsole, 7); // Whtie

}

}

// Functions Definations

void gotoxy(int x, int y)

{

COORD coordinates;

coordinates.X = x;

coordinates.Y = y;

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

}

/\*\*\*\*\* Load and print maze from file \*\*\*\*\*/

void load(char maze[][61], int rowSize, int colSize, string path)

{

fstream myFile;

string record;

myFile.open(path, ios::in);

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

{

getline(myFile, record);

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

{

maze[row][col] = record[col];

}

}

}

void printMaze(char maze[][61], int colSize, int rowSize)

{

for (int x = 0; x < rowSize; x++)

{

for (int y = 0; y < colSize; y++)

{

cout << maze[x][y];

}

cout << endl;

}

}

/\*\*\*\*\* Movement \*\*\*\*\*/

void moveLeft(char maze[][61], int &xIdx, int &yIdx)

{

if (maze[xIdx][yIdx - 1] == ' ' || maze[xIdx][yIdx - 1] == 'G') // checking and Printing space

{

maze[xIdx][yIdx] = ' ';

gotoxy(yIdx, xIdx);

cout << " ";

yIdx--;

SetConsoleTextAttribute(hConsole, 1); // Blue

maze[xIdx][yIdx] = 'T'; // Printing object

gotoxy(yIdx, xIdx);

cout << 'T';

}

}

void moveRight(char maze[][61], int &xIdx, int &yIdx)

{

if (maze[xIdx][yIdx + 1] == ' ' || maze[xIdx][yIdx + 1] == 'G') // checking and Printing space

{

maze[xIdx][yIdx] = ' ';

gotoxy(yIdx, xIdx);

cout << " ";

yIdx++;

SetConsoleTextAttribute(hConsole, 1); // Blue

maze[xIdx][yIdx] = 'T'; // Printing object

gotoxy(yIdx, xIdx);

cout << 'T';

}

}

void moveUp(char maze[][61], int &xIdx, int &yIdx)

{

if (maze[xIdx - 1][yIdx] == ' ' || maze[xIdx - 1][yIdx] == 'G') // checking and Printing space

{

maze[xIdx][yIdx] = ' ';

gotoxy(yIdx, xIdx);

cout << " ";

xIdx--;

SetConsoleTextAttribute(hConsole, 1); // Blue

maze[xIdx][yIdx] = 'T'; // Printing object

gotoxy(yIdx, xIdx);

cout << 'T';

}

}

void moveDown(char maze[][61], int &xIdx, int &yIdx)

{

if (maze[xIdx + 1][yIdx] == ' ' || maze[xIdx + 1][yIdx] == 'G') // checking and Printing space

{

maze[xIdx][yIdx] = ' ';

gotoxy(yIdx, xIdx);

cout << " ";

xIdx++;

SetConsoleTextAttribute(hConsole, 1); // Blue

maze[xIdx][yIdx] = 'T'; // Printing object

gotoxy(yIdx, xIdx);

cout << 'T';

}

}

int random()

{

srand(time(0));

int result = rand() % 4;

return result;

}

void gamePlay(bool gameRunning, int &life, char maze[][61], int rowSize, int colSize, int &xIdx, int &yIdx, int &police1X, int &police1Y, int &police2X, int &police2Y, int &police3X, int &police3Y)

{

int seconds = 120;

int counter = 0;

int position;

time\_t end = time(NULL) + 120;

while (time(NULL) <= end && gameRunning == true && life > 0 && (Eat != true || maze[thing1X][thing1Y] == 'G'))

{

Sleep(170);

Eat = checking(maze, xIdx, yIdx);

if (Eat == true && maze[thing1X][thing1Y] != 'G' && life > 0)

{

SetConsoleTextAttribute(hConsole, 2); // Green

gotoxy(70, 13);

cout << "YOU WIN!!!";

getch();

}

SetConsoleTextAttribute(hConsole, 5); // Purple

gotoxy(0, 26);

cout << "Life: ";

SetConsoleTextAttribute(hConsole, 7); // Whtie

cout << life;

SetConsoleTextAttribute(hConsole, 5); // Purple

gotoxy(20, 26);

cout << "Score: ";

SetConsoleTextAttribute(hConsole, 7); // Whtie

cout << score;

int minutes = seconds / 60;

int sec = seconds % 60;

SetConsoleTextAttribute(hConsole, 5); // Purple

gotoxy(40, 26);

cout << "Time Left = ";

SetConsoleTextAttribute(hConsole, 7); // Whtie

gotoxy(40, 39);

cout << minutes << " : " << sec;

position = random();

if (maze[thing1X][thing1Y] != 'G')

{

if (position == 0)

{

movePolice1\_left(maze, police1X, police1Y);

movePolice2\_right(maze, police2X, police2Y);

movePolice3\_down(maze, police3X, police3Y);

}

if (position == 1)

{

movePolice1\_right(maze, police1X, police1Y);

movePolice2\_left(maze, police2X, police2Y);

movePolice3\_up(maze, police3X, police3Y);

}

if (position == 2)

{

movePolice1\_up(maze, police1X, police1Y);

movePolice2\_down(maze, police2X, police2Y);

movePolice3\_left(maze, police3X, police3Y);

}

if (position == 3)

{

movePolice1\_down(maze, police1X, police1Y);

movePolice2\_up(maze, police2X, police2Y);

movePolice3\_right(maze, police3X, police3Y);

}

police1Bullet(maze, rowSize, colSize, life);

police2Bullet(maze, rowSize, colSize, life);

police3Bullet(maze, rowSize, colSize, life);

}

if (GetAsyncKeyState(VK\_LEFT))

{

moveLeft(maze, xIdx, yIdx);

}

if (GetAsyncKeyState(VK\_RIGHT))

{

moveRight(maze, xIdx, yIdx);

}

if (GetAsyncKeyState(VK\_UP))

{

moveUp(maze, xIdx, yIdx);

}

if (GetAsyncKeyState(VK\_DOWN))

{

moveDown(maze, xIdx, yIdx);

}

if (GetAsyncKeyState(VK\_ESCAPE))

{

gameRunning = false; // Stop the game

}

if (GetAsyncKeyState(VK\_SPACE))

{

if (maze[xIdx - 1][yIdx] == ' ' || maze[xIdx - 1][yIdx] == 'P')

{

int TIdx = xIdx;

TIdx--;

maze[TIdx][yIdx] = '.';

SetConsoleTextAttribute(hConsole, 3); // Aqua

gotoxy(yIdx, TIdx);

cout << ".";

}

}

bulletMovement(maze, rowSize, colSize, score);

if (counter >= 5 && seconds > 0)

{

seconds--;

counter = 0;

}

counter++;

}

}

void bulletMovement(char maze[][61], int rowSize, int colSize, int &score)

{

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

{

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

{

if (maze[row][col] == '.' && (maze[row - 1][col] == ' ' || maze[row - 1][col] == 'P'))

{

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

{

score++;

}

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

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

SetConsoleTextAttribute(hConsole, 3); // Aqua

gotoxy(col, row - 1);

cout << ".";

}

else if (maze[row][col] == '.' && (maze[row - 1][col] != ' ' && maze[row - 1][col] != 'P'))

{

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

}

}

}

}

void police1Bullet(char maze[][61], int rowSize, int colSize, int &life)

{

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

{

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

{

if (maze[row][col] == '\*' && (maze[row + 1][col] == ' ' || maze[row + 1][col] == 'T'))

{

if (maze[row + 1][col] == 'T')

{

life--;

}

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

maze[row + 1][col] = '\*';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(col, row + 1);

cout << '\*';

}

else if (maze[row][col] == '\*' && (maze[row + 1][col] != ' ' && maze[row + 1][col] != 'T' && maze[row][col + 1] != 'P'))

{

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

}

}

}

}

void police2Bullet(char maze[][61], int rowSize, int colSize, int &life)

{

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

{

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

{

if (maze[row][col] == ',' && (maze[row][col - 1] == ' ' || maze[row][col - 1] == 'T'))

{

if (maze[row][col - 1] == 'T')

{

life--;

}

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

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

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(col - 1, row);

cout << ",";

}

else if (maze[row][col] == ',' && (maze[row][col - 1] != ' ' && maze[row][col - 1] != 'T' && maze[row][col + 1] != 'P'))

{

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

}

}

}

}

void police3Bullet(char maze[][61], int rowSize, int colSize, int &life)

{

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

{

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

{

if (maze[row][col] == '@' && (maze[row][col + 1] == ' ' || maze[row][col + 1] == 'T')) // Right

{

if (maze[row][col + 1] == 'T')

{

life--;

}

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

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

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(col + 1, row);

cout << "@";

}

else if (maze[row][col] == '@' && (maze[row][col + 1] != ' ' && maze[row][col + 1] != 'T'))

{

maze[row][col] = ' ';

gotoxy(col, row);

cout << " ";

}

}

}

}

bool checking(char maze[][61], int xIdx, int yIdx)

{

if (maze[xIdx][yIdx] == maze[thing2X][thing2Y])

{

return true;

}

else

{

return false;

}

}

void movePolice1\_up(char maze[][61], int &police1X, int &police1Y)

{

if (maze[police1X - 1][police1Y] == ' ') // checking and Printing space

{

maze[police1X][police1Y] = ' ';

gotoxy(police1Y, police1X);

cout << " ";

police1X--;

maze[police1X][police1Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police1Y, police1X);

cout << 'P';

}

if (maze[police1X - 1][police1Y] == ' ' || maze[police1X - 1][police1Y] == 'T')

{

int AIdx = police1X;

AIdx--;

maze[AIdx][police1Y] = '\*';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(police1Y, AIdx);

cout << "\*";

}

}

void movePolice1\_down(char maze[][61], int &police1X, int &police1Y)

{

if (maze[police1X + 1][police1Y] == ' ') // checking and Printing space

{

maze[police1X][police1Y] = ' ';

gotoxy(police1Y, police1X);

cout << " ";

police1X++;

maze[police1X][police1Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police1Y, police1X);

cout << 'P';

}

if (maze[police1X + 1][police1Y] == ' ' || maze[police1X + 1][police1Y] == 'T')

{

int AIdx = police1X;

AIdx++;

maze[police1X][police1Y] = '\*';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(police1Y, police1X);

cout << "\*";

}

}

void movePolice1\_right(char maze[][61], int &police1X, int &police1Y)

{

if (maze[police1X][police1Y + 1] == ' ') // checking and Printing space

{

maze[police1X][police1Y] = ' ';

gotoxy(police1Y, police1X);

cout << " ";

police1Y++;

maze[police1X][police1Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police1Y, police1X);

cout << 'P';

}

if (maze[police1X][police1Y + 1] == ' ' || maze[police1X][police1Y + 1] == 'T')

{

int AIdx = police1Y;

AIdx++;

maze[police1X][AIdx] = '\*';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(AIdx, police1X);

cout << "\*";

}

}

void movePolice1\_left(char maze[][61], int &police1X, int &police1Y)

{

if (maze[police1X][police1Y - 1] == ' ' || maze[police1X][police1Y - 1] == ',' || maze[police1X][police1Y - 1] == '@', maze[police1X][police1Y - 1] == '\*') // checking and Printing space

{

maze[police1X][police1Y] = ' ';

gotoxy(police1Y, police1X);

cout << " ";

police1Y--;

maze[police1X][police1Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police1Y, police1X);

cout << 'P';

}

if (maze[police1X][police1Y - 1] == ' ' || maze[police1X][police1Y - 1] == 'T')

{

int AIdx = police1Y;

AIdx--;

maze[police1X][AIdx] = '\*';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(AIdx, police1X);

cout << "\*";

}

}

void movePolice2\_up(char maze[][61], int &police2X, int &police2Y)

{

if (maze[police2X - 1][police2Y] == ' ') // checking and Printing space

{

maze[police2X][police2Y] = ' ';

gotoxy(police2Y, police2X);

cout << " ";

police2X--;

maze[police2X][police2Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police2Y, police2X);

cout << 'P';

}

if (maze[police2X - 1][police2Y] == ' ' || maze[police2X - 1][police2Y] == 'T')

{

int BIdx = police2X;

BIdx--;

maze[BIdx][police2Y] = ',';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(police2Y, BIdx);

cout << ",";

}

}

void movePolice2\_down(char maze[][61], int &police2X, int &police2Y)

{

if (maze[police2X + 1][police2Y] == ' ') // checking and Printing space

{

maze[police2X][police2Y] = ' ';

gotoxy(police2Y, police2X);

cout << " ";

police2X++;

maze[police2X][police2Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police2Y, police2X);

cout << 'P';

}

if (maze[police2X + 1][police2Y] == ' ' || maze[police2X + 1][police2Y] == 'T')

{

int BIdx = police2X;

BIdx++;

maze[BIdx][police2Y] = ',';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(police2Y, BIdx);

cout << ",";

}

}

void movePolice2\_right(char maze[][61], int &police2X, int &police2Y)

{

if (maze[police2X][police2Y + 1] == ' ') // checking and Printing space

{

maze[police2X][police2Y] = ' ';

gotoxy(police2Y, police2X);

cout << " ";

police2Y++;

maze[police2X][police2Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police2Y, police2X);

cout << 'P';

}

if (maze[police2X][police2Y + 1] == ' ' || maze[police2X][police2Y + 1] == 'T')

{

int BIdx = police2Y;

BIdx++;

maze[police2X][BIdx] = ',';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(BIdx, police2X);

cout << ",";

}

}

void movePolice2\_left(char maze[][61], int &police2X, int &police2Y)

{

if (maze[police2X][police2Y - 1] == ' ') // checking and Printing space

{

maze[police2X][police2Y] = ' ';

gotoxy(police2Y, police2X);

cout << " ";

police2Y--;

maze[police2X][police2Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police2Y, police2X);

cout << 'P';

}

if (maze[police2X][police2Y - 1] == ' ' || maze[police2X][police2Y - 1] == 'T')

{

int BIdx = police2Y;

BIdx--;

maze[police2X][BIdx] = ',';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(BIdx, police2X);

cout << ",";

}

}

void movePolice3\_up(char maze[][61], int &police3X, int &police3Y)

{

if (maze[police3X - 1][police3Y] == ' ') // checking and Printing space

{

maze[police3X][police3Y] = ' ';

gotoxy(police3Y, police3X);

cout << " ";

police3X--;

maze[police3X][police3Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police3Y, police3X);

cout << 'P';

}

if (maze[police3X - 1][police3Y] == ' ' || maze[police3X - 1][police3Y == 'T'])

{

int AIdx = police3X;

AIdx--;

maze[AIdx][police3Y] = '@';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(police3Y, AIdx);

cout << "@";

}

}

void movePolice3\_down(char maze[][61], int &police3X, int &police3Y)

{

if (maze[police3X + 1][police3Y] == ' ') // checking and Printing space

{

maze[police3X][police3Y] = ' ';

gotoxy(police3Y, police3X);

cout << " ";

police3X++;

maze[police3X][police3Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police3Y, police3X);

cout << 'P';

}

if (maze[police3X + 1][police3Y] == ' ' || maze[police3X + 1][police3Y == 'T'])

{

int AIdx = police3X;

AIdx++;

maze[AIdx][police3Y] = '@';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(police3Y, AIdx);

cout << "@";

}

}

void movePolice3\_right(char maze[][61], int &police3X, int &police3Y)

{

if (maze[police3X][police3Y + 1] == ' ') // checking and Printing space

{

maze[police3X][police3Y] = ' ';

gotoxy(police3Y, police3X);

cout << " ";

police3Y++;

maze[police3X][police3Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police3Y, police3X);

cout << 'P';

}

if (maze[police3X][police3Y + 1] == ' ' || maze[police3X][police3Y + 1] == 'T')

{

int AIdx = police3Y;

AIdx++;

maze[police3X][AIdx] = '@';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(AIdx, police3X);

cout << "@";

}

}

void movePolice3\_left(char maze[][61], int &police3X, int &police3Y)

{

if (maze[police3X][police3Y - 1] == ' ') // checking and Printing space

{

maze[police3X][police3Y] = ' ';

gotoxy(police3Y, police3X);

cout << " ";

police3Y--;

maze[police3X][police3Y] = 'P'; // Printing object

SetConsoleTextAttribute(hConsole, 4); // Red

gotoxy(police3Y, police3X);

cout << 'P';

}

if (maze[police3X][police3Y - 1] == ' ' || maze[police3X][police3Y - 1] == 'T')

{

int AIdx = police3Y;

AIdx--;

maze[police3X][AIdx] = '@';

SetConsoleTextAttribute(hConsole, 7); // white

gotoxy(AIdx, police3X);

cout << "@";

}

}

**Student Reg. No:**  2021-CS-144  **Student Name.**  Muhammad Umair Shahid

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation  Formatting **Grade:** | All the documentation meets all the criteria. |  |  |  |
| Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow**  Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project  Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50%  criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style **Grade:** | All Code style criteria is followed | All code style criteria followed but some  improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined. White Spaces are well used. Comments are added. | | | | |
| Code  Documentation Mapping **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure  (Arrays) **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features **Grade:** | Sort working 100% and generating useful report | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria  but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and  Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not  know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |