**QATAR FIFA MANIA**



Session: 2022 – 2026

**Submitted by:**

Saba Shahdin 2022-CS-112

**Supervised by:**

Sir Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

Table of Contents

Story of Qatar Fifa Mania**3**

Game Character Description3

Game Object Description4

**Rules And interaction4**

Goal of the Game4

Wireframe of game5

Data Structure6

Function Prototype 9

Complete code of game13

**Story of Qatar FIFA Mania:**

Al Rihla is ambitious about playing. He loves to play any sport whether it is cricket or football. But inspired by **FIFA WORLD CUP 2022**. He wants to play football and for 2 months he has been practicing to get perfection in scoring a goal daily.

He plays in a football stadium full of adventures to find pleasure as well as to get skills. As he moves he finds **food pallets** which increase his score and encourages him to go ahead. As he goes on he finds **energy pallets** that increase his energy and gives the power to fire his enemies. He has to go to the **goalpost** protecting himself from the firings of the **defenders** and from the **goalkeeper** who is ready to push him away from the goalpost.

After facing all these hardships on the field he succeeded to score a **goal** which makes him feel better as his practice is going on and becoming better day by day. Al Rihla is happy after killing defenders with his power and making the goalkeeper fail in his efforts by scoring the goal. He will soon be the best goal-scoring player. He will make it happen with his efforts.

**Game Characters Description**

**Players:**

There is one human player in the Game.

**Al-Rihla:**  
Al Rihla is the human ambitious player of the game. He is a blue color throughout. He loves to play football and finds himself happy scoring goals and eating food pallets as well as energy pallets which encourages him and makes him powerful enough to defeat his enemies.

**Enemies:**There are 3 enemies in the game.

**Defender 1:**

Defender 1 is the enemy of Al Rihla which moves in the horizontal direction and fires in the upward and downward direction to distract al Rihla from going to the goalpost

**Defender 2:**

Defender 2 is the enemy of al Rihla which fires in vertical direction to distract Al Rihla from going to goalpost.

**Defender 3:**

Defender 3 is al Rihla’s enemy, chasing him in the field and distracting him from his goal.

**Goal Keeper:**

Goal keeper is the worst enemy of Al Rihla and is known for stopping Al Rihla from going to goalpost patrolling actively in front of it

**Game Objects Description**

Following are the Objects in the Game

* **Energy Pallet:**

Energy pallets are the objects in the field that increases Al Rihla's energy and gives him the power of firing his enemy by pressing space bar in the left direction, numpad 0 to fire right, numpad1 to fire upward, and numpad2 to fire downward.

* **Food Pallets:**

Less than ‘<’ are the food pallets which increase Al Rihla 's score and give him confidence in going on

* **Walls:**

Walls are the boundaries for the defenders and Al Rihla and the goalkeeper has to go in the goal post to win a goal and the goalkeeper only moves the goal post

**Rules & Interactions:**

* Al-Rihla can eat **Food Pallets** that have been put across the field.
* If Al Rihla eats **Energy Pallets**, he can fire at the defender.
* His score increases by 1 after eating food pallets and 5 by reaching into the goal post to protect themselves from defenders and the goalkeeper in the field.
* His score also increases if his bullet hits any enemy.
* He has only three lives for achieving his goal. If player hits by chasing the enemy his life decreases by 1 and the game terminates if his life becomes 0
* If he hits by fire his health decreases.
* If you want to pause the game press escape the key and go to menu and then select your desired option.

**Goal of the Qatar FIFA Mania:**

The goal of the game is to eat the food pallets to increase score and energy pallets to fire enemy. The major purpose is to go to the **goalpost** avoiding defenders and goalkeeper.

* **Wireframes of the Game:**

****

Figure 1 Main Screen

****

Figure 2 Rules

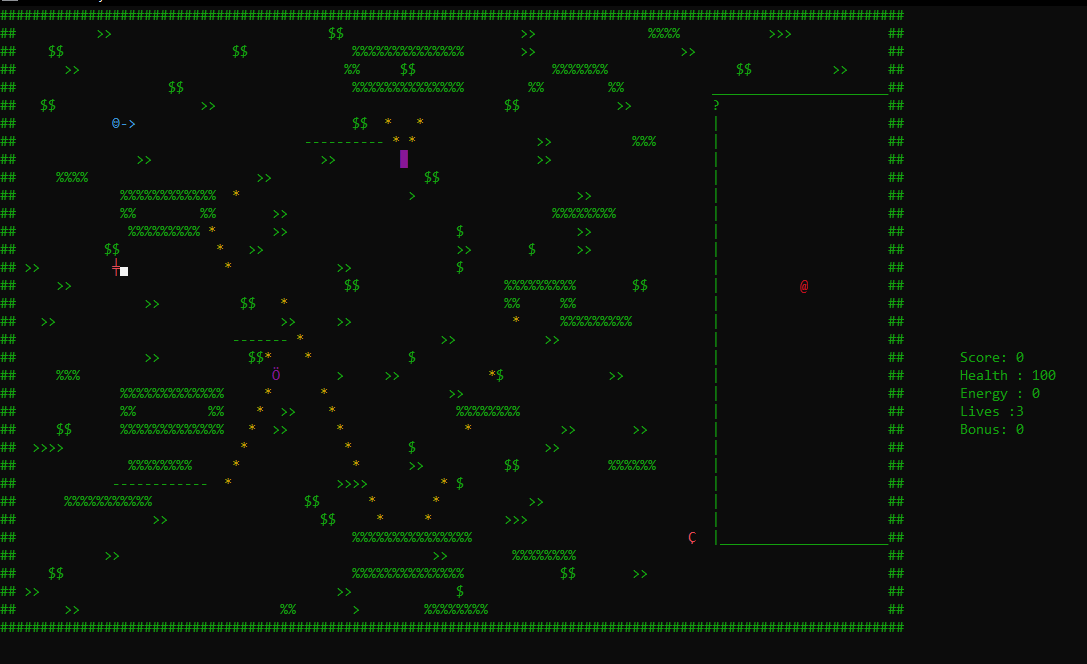


Figure 3 Game Maze

* **Data Structures:**
* **Main Variable**

bool gameRunning = true;

* **Player Location**

int px = 14;

int py = 6;

* **Player Print**

char ball[3] = {1, '-', '>'};

* **Player Bullet**

int bulletX[1000];

int bulletY[1000];

bool isBulletActive[1000];

int bulletCount = 0;

int bulletRX[1000];

int bulletRY[1000];

bool isBulletActiveR[1000];

int bulletCountR = 0;

int bulletLX[1000];

int bulletLY[1000];

bool isBulletActiveL[1000];

int bulletCountL = 0;

int bulletDX[1000];

int bulletDY[1000];

bool isBulletActiveD[1000];

int bulletCountD = 0;

* **Enemy Bullet**

int bullet2X[1000];

int bullet2Y[1000];

bool isBulletActiveEnemy[1000];

int bulletCounter = 0;

int bullet3X[1000];

int bullet3Y[1000];

bool isBulletActiveEnemy2[1000];

int bulletCounter2 = 0;

int bullet4X[1000];

int bullet4Y[1000];

bool isBulletActiveEnemy3[1000];

int bulletCounter3 = 0;

int bullet5X[1000];

int bullet5Y[1000];

bool isBulletActiveEnemy4[1000];

int bulletCounter4 = 0;

* **Enemy1 Location**

int enemyX = 50;

int enemyY = 10;

* **Enemy2 Location**

int enemy2X = 13;

int enemy2Y = 20;

* **Enemy3 Location**

int goalX = 86;

int goalY = 8;

* **Chasing Enemy**

int dx = 20;

int dy = 15;

int direction;

int chasingDistance = 13;

char nextLocation;

* **Print Characters**

char defender3 = 219;

char defender4 = 153;

char defender5 = 216;

char goal = 128;

char ball1 = 64;

* **Credentials**

int health1 = 100;

int score = 0;

int energy = 0;

int lives = 3;

int bonus = 0;

string option;

* **2d Array of Maze**

char maze[36][114];

**Function Prototypes**

* **Main Function**

void header();

char menu();

void mazer();

void rules();

void printadd();

* **Supporting Function**

char getCharAtxy(short int x, short int y);

void gotoxy(int x, int y);

void clearScreen();

* **Player move Functiom**

void erase(int x, int y);

void printball(int x, int y);

void playerMove();

void moveLeft();

void moveRight();

void moveDown();

void moveUp();

* **Player Bullet left**

void moveBullet();

void generateBullet();

void removeBulletFromArray(int index);

void printBullet(int x, int y);

void eraseBullet(int x, int y);

void makeBulletInactive(int index);

* **Enemy 1 bullet**

void generateBulletEnemy1();

void moveBulletEnemy();

void makeBulletInactiveEnemy1(int index);

void printBulletEnemy1(int enemyX, int enemyY);

void eraseBulletEnemy1(int enemyX, int enemyY);

* **Move enemy1**

void printDefender(int x, int y);

void eraseDefender(int x, int y);

void defender2();

* **Player bullet right**

void moveBulletLeft();

void generateBulletLeft();

void removeBulletFromArrayLeft(int index);

void printBulletLeft(int x, int y);

void eraseBulletLeft(int x, int y);

void makeBulletInactiveLeft(int index);

* **Player bullet up**

void moveBulletUP();

void generateBulletUP();

void removeBulletFromArrayUP(int index);

void printBulletUP(int x, int y);

void eraseBulletUP(int x, int y);

void makeBulletInactiveUP(int index);

* **Player bullet down**

void moveBulletDOWN();

void generateBulletDOWN();

void removeBulletFromArrayDOWN(int index);

void printBulletDOWN(int x, int y);

void eraseBulletDOWN(int x, int y);

void makeBulletInactiveDOWN(int index);

void generateBulletEnemy2();

* **Enemy1 bullet down**

void moveBulletEnemy2();

void makeBulletInactiveEnemy2(int index);

void printBulletEnemy2(int enemyX, int enemyY);

void eraseBulletEnemy2(int enemyX, int enemyY);

* **Move enemy2**

void printDefender2(int x, int y);

void eraseDefender2(int x, int y);

void defender();

* **Enemy2 bullet left**

void generateBulletEnemy3();

void moveBulletEnemy3();

void makeBulletInactiveEnemy3(int index);

void printBulletEnemy3(int enemy2X, int enemy2Y);

void eraseBulletEnemy3(int enemy2X, int enemy2Y);

* **Enemy2 bullet right**

void generateBulletEnemy4();

void moveBulletEnemy4();

void makeBulletInactiveEnemy4(int index);

void printBulletEnemy4(int enemy2X, int enemy2Y);

void eraseBulletEnemy4(int enemy2X, int enemy2Y);

* **Credientals**

int energyIncrease(int energy);

int addScore();

int health();

int life();

void print();

int addBonus();

* **GoalKeeper move**

void printGoalKeeper(int x, int y);

void eraseGoalKeeper(int x, int y);

void goalKeeper();

* **Collison with player**

void bulletCollisionWithPlayer();

void bulletCollisionWithPlayerLeft();

void bulletCollisionWithPlayerUp();

void bulletCollisionWithPlayerDown();

* **Chasing enemy**

void moveEnemyLeft();

void moveEnemyRight();

void moveEnemyDown();

void moveEnemyUp();

void moveEnemy();

void eraseEnemy(int x, int y);

void printEnemy(int x, int y);

* **Collision with enemy**

void bulletCollisionWithEnemy();

void bulletCollisionWithEnemyLeft();

void bulletCollisionWithEnemyUp();

void bulletCollisionWithEnemyDown();

* **Collison of chase**

void enemyWithPlayerRight();

void enemyWithPlayerLeft();

void enemyWithPlayerUp();

void enemyWithPlayerDown();

* **Collion with chase and bullet**

void bulletCollisionWithChase();

void bulletCollisionWithChaseLeft();

void bulletCollisionWithChaseUp();

void bulletCollisionWithChaseDown();

* **File maze**

void loadMaze();

void loadPause();

void storePause();

* **Credeietals**

void storeScore();

void loadScore();

string parseScore(string record, int field);

* **File location**

void storeLocation();

void loadLocation();

string parseLocation(string record, int field);

* **File bullet**

void storeBullet();

void loadBullet();

string parseBullet(string record, int field);

* **Complete Code**

//Library files

#include <iostream>

#include <windows.h>

#include <conio.h>

#include <math.h>

#include <fstream>

#include <sstream>

using namespace std;

//Function prototype

//main Function

void header();

char menu();

void mazer();

void rules();

void printadd();

//Supporting Function

char getCharAtxy(short int x, short int y);

void gotoxy(int x, int y);

void clearScreen();

//Player move Functiom

void erase(int x, int y);

void printball(int x, int y);

void playerMove();

void moveLeft();

void moveRight();

void moveDown();

void moveUp();

//Player Bullet left

void moveBullet();

void generateBullet();

void removeBulletFromArray(int index);

void printBullet(int x, int y);

void eraseBullet(int x, int y);

void makeBulletInactive(int index);

//enemy 1 bullet

void generateBulletEnemy1();

void moveBulletEnemy();

void makeBulletInactiveEnemy1(int index);

void printBulletEnemy1(int enemyX, int enemyY);

void eraseBulletEnemy1(int enemyX, int enemyY);

//move enemy1

void printDefender(int x, int y);

void eraseDefender(int x, int y);

void defender2();

//player bullet right

void moveBulletLeft();

void generateBulletLeft();

void removeBulletFromArrayLeft(int index);

void printBulletLeft(int x, int y);

void eraseBulletLeft(int x, int y);

void makeBulletInactiveLeft(int index);

//player bullet up

void moveBulletUP();

void generateBulletUP();

void removeBulletFromArrayUP(int index);

void printBulletUP(int x, int y);

void eraseBulletUP(int x, int y);

void makeBulletInactiveUP(int index);

//player bullet down

void moveBulletDOWN();

void generateBulletDOWN();

void removeBulletFromArrayDOWN(int index);

void printBulletDOWN(int x, int y);

void eraseBulletDOWN(int x, int y);

void makeBulletInactiveDOWN(int index);

void generateBulletEnemy2();

//enemy1 bullet down

void moveBulletEnemy2();

void makeBulletInactiveEnemy2(int index);

void printBulletEnemy2(int enemyX, int enemyY);

void eraseBulletEnemy2(int enemyX, int enemyY);

//move enemy2

void printDefender2(int x, int y);

void eraseDefender2(int x, int y);

void defender();

//enemy2 bullet left

void generateBulletEnemy3();

void moveBulletEnemy3();

void makeBulletInactiveEnemy3(int index);

void printBulletEnemy3(int enemy2X, int enemy2Y);

void eraseBulletEnemy3(int enemy2X, int enemy2Y);

//enemy2 bullet right

void generateBulletEnemy4();

void moveBulletEnemy4();

void makeBulletInactiveEnemy4(int index);

void printBulletEnemy4(int enemy2X, int enemy2Y);

void eraseBulletEnemy4(int enemy2X, int enemy2Y);

//credientals

int energyIncrease(int energy);

int addScore();

int health();

int life();

void print();

int addBonus();

//goalKeeper move

void printGoalKeeper(int x, int y);

void eraseGoalKeeper(int x, int y);

void goalKeeper();

//collison with player

void bulletCollisionWithPlayer();

void bulletCollisionWithPlayerLeft();

void bulletCollisionWithPlayerUp();

void bulletCollisionWithPlayerDown();

//chasing enemy

void moveEnemyLeft();

void moveEnemyRight();

void moveEnemyDown();

void moveEnemyUp();

void moveEnemy();

void eraseEnemy(int x, int y);

void printEnemy(int x, int y);

//collision with enemy

void bulletCollisionWithEnemy();

void bulletCollisionWithEnemyLeft();

void bulletCollisionWithEnemyUp();

void bulletCollisionWithEnemyDown();

//collison of chase

void enemyWithPlayerRight();

void enemyWithPlayerLeft();

void enemyWithPlayerUp();

void enemyWithPlayerDown();

//collion with chase and bullet

void bulletCollisionWithChase();

void bulletCollisionWithChaseLeft();

void bulletCollisionWithChaseUp();

void bulletCollisionWithChaseDown();

//file maze

void loadMaze();

void loadPause();

void storePause();

//credeietals

void storeScore();

void loadScore();

string parseScore(string record, int field);

// file location

void storeLocation();

void loadLocation();

string parseLocation(string record, int field);

//file bullet

void storeBullet();

void loadBullet();

string parseBullet(string record, int field);

// data structure

// main variable

bool gameRunning = true;

// player location

int px = 14;

int py = 6;

//player print

char ball[3] = {1, '-', '>'};

//playerBullet

int bulletX[1000];

int bulletY[1000];

bool isBulletActive[1000];

int bulletCount = 0;

int bulletRX[1000];

int bulletRY[1000];

bool isBulletActiveR[1000];

int bulletCountR = 0;

int bulletLX[1000];

int bulletLY[1000];

bool isBulletActiveL[1000];

int bulletCountL = 0;

int bulletDX[1000];

int bulletDY[1000];

bool isBulletActiveD[1000];

int bulletCountD = 0;

//enemyBullet

int bullet2X[1000];

int bullet2Y[1000];

bool isBulletActiveEnemy[1000];

int bulletCounter = 0;

int bullet3X[1000];

int bullet3Y[1000];

bool isBulletActiveEnemy2[1000];

int bulletCounter2 = 0;

int bullet4X[1000];

int bullet4Y[1000];

bool isBulletActiveEnemy3[1000];

int bulletCounter3 = 0;

int bullet5X[1000];

int bullet5Y[1000];

bool isBulletActiveEnemy4[1000];

int bulletCounter4 = 0;

//enemy1 location

int enemyX = 50;

int enemyY = 10;

//enemy2 location

int enemy2X = 13;

int enemy2Y = 20;

//enemy3 location

int goalX = 86;

int goalY = 8;

//chasing enemy

int dx = 20;

int dy = 15;

int direction;

int chasingDistance = 13;

char nextLocation;

char ball1 = 64;

//print characters

char defender3 = 219;

char defender4 = 153;

char defender5 = 216;

char goal = 128;

//credientials

int health1 = 100;

int score = 0;

int energy = 0;

int lives = 3;

int bonus = 0;

string option;

//2d array of maze

char maze[36][114];

//main body

main()

{

system("cls");

header();

option = menu();

clearScreen();

header();

//menu options

while (option != "5")

{

if (option == "1")

{

system ("cls");

header();

rules();

clearScreen();

header();

option = menu();

}

// New game;

else if (option == "2")

{

loadMaze();

system("cls");

gameRunning = true;

mazer();

print();

printball(px, py);

printDefender(enemyX, enemyY);

printDefender2(enemy2X, enemy2Y);

printGoalKeeper(enemyX, enemyY);

printEnemy(dx, dy);

printadd();

while (gameRunning == true)

{

Sleep(200);

playerMove();

defender2();

defender();

goalKeeper();

moveEnemy();

}

}

// pause

else if (option == "3")

{

storePause();

storeScore();

storeLocation();

storeBullet();

system("cls");

header();

option = menu();

}

// resume;

else if (option == "4")

{

loadPause();

loadScore();

loadLocation();

loadBullet();

system("cls");

gameRunning = true;

mazer();

print();

printball(px, py);

printDefender(enemyX, enemyY);

printDefender2(enemy2X, enemy2Y);

printGoalKeeper(enemyX, enemyY);

printEnemy(dx, dy);

printadd();

while (gameRunning == true)

{

Sleep(200);

playerMove();

defender2();

defender();

goalKeeper();

moveEnemy();

}

}

}

}

//game Title

void header()

{

int k = 2;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

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

cout << " .' `. / \\ | \_ \_ | / \\ |\_ \_\_ \\ |\_ \_\_ ||\_ \_||\_ \_\_ | / \\ " << endl;

cout << " / .-. \\ / \_ \\ |\_/ | | \\\_| / \_ \\ | |\_\_) | | |\_ \\\_| | | | |\_ \\\_| / \_ \\ " << endl;

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

cout << " \\ `-' \\\_ \_/ / \\ \\\_ \_| |\_ \_/ / \\ \\\_ \_| | \\ \\\_ \_| |\_ \_| |\_ \_| |\_ \_/ / \\ \\\_ " << endl;

cout << " `.\_\_\_.\\\_\_||\_\_\_\_| |\_\_\_\_||\_\_\_\_\_||\_\_\_\_| |\_\_\_\_||\_\_\_\_|\_|\_\_\_|\_\_\_|\_\_\_\_\_| |\_\_\_\_\_||\_\_\_\_\_||\_\_\_\_| |\_\_\_\_| " << endl;

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

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

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

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

cout << " |\_\_\_\_\_||\_\_\_\_\_||\_\_\_\_| |\_\_\_\_||\_\_\_\_\_|\\\_\_\_\_||\_\_\_\_\_||\_\_\_\_| |\_\_\_\_| " << endl;

cout << endl;

cout << endl;

cout << endl;

}

//menu

char menu()

{

cout << " Menu " << endl;

cout << " ----------------------------------" << endl;

cout << " 1 . Rules" << endl;

cout << " 2 . Start a new game " << endl;

cout << " 3 . Pause game " << endl;

cout << " 4 . Resume previous game " << endl;

cout << " 5 . Exit " << endl;

cout << " Entre any option : ";

char option;

cin >> option;

return option;

}

// print maze

void mazer()

{

int k = 2;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

for (int row = 0; row < 36; row++)

{

for (int col = 0; col < 114; col++)

{

cout << maze[row][col];

}

cout << endl;

}

}

// print credientals

void print()

{

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(120, 20);

cout << "Health : " << health1;

gotoxy(120, 21);

cout << "Energy : " << energy;

gotoxy(120, 22);

cout << "Lives :" << lives;

gotoxy(120, 23);

cout << "Bonus: " << bonus;

}

// print rules

void rules()

{

cout << " Rules " << endl;

cout << " ----------------------------------" << endl;

cout << " 1. Use space bar to fire right direction." << endl;

cout << " 2. Use num pad 0 to fire left direction." << endl;

cout << " 3. Use num pad 1 to fire up direction." << endl;

cout << " 4. Use num pad 2 to fire down direction." << endl;

cout << " 5. You can fire only if you have energy." << endl;

cout << " 6. Energy can be gained by collecting $." << endl;

cout << " 7. score increase if you bullet hit any enemy or you eat >" << endl;

cout << " 8. health decreases if you hit by any of enemy or his bullet." << endl;

cout << " 9. you gained bonus if you ate @ in goalpost from the left direction and you go to your initial position." << endl;

cout << " 10.You have 3 lives to play." << endl;

cout << " 11.You can exit by pressing escape." << endl;

}

// supportive

void clearScreen()

{

cout << "Press Any Key to Continue.." << endl;

getch();

system("cls");

}

// bonus location

void printadd()

{

int k = 4;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

gotoxy(100, 15);

cout << "@";

}

// print ball

void printball(int x, int y)

{

int k = 3;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

gotoxy(x, y);

for (int x = 0; x < 3; x++)

{

cout << ball[x];

}

}

void erase(int x, int y)

{

gotoxy(x, y);

for (int x = 0; x < 3; x++)

{

cout << " ";

}

}

// move ball

void playerMove()

{

char nextLocation = getCharAtxy(px, py);

if (GetAsyncKeyState(VK\_LEFT))

{

moveLeft();

}

else if (GetAsyncKeyState(VK\_RIGHT))

{

moveRight();

}

if (GetAsyncKeyState(VK\_UP))

{

moveUp();

}

else if (GetAsyncKeyState(VK\_DOWN))

{

moveDown();

}

else if (GetAsyncKeyState(VK\_SPACE) && energy > 0)

{

generateBullet();

}

if (GetAsyncKeyState(VK\_NUMPAD0) && energy > 0)

{

generateBulletLeft();

}

if (GetAsyncKeyState(VK\_NUMPAD1) && energy > 0)

{

generateBulletUP();

}

if (GetAsyncKeyState(VK\_NUMPAD2) && energy > 0)

{

generateBulletDOWN();

}

// terminate game running

else if (GetAsyncKeyState(VK\_ESCAPE))

{

gameRunning = false;

system("cls");

header();

option = menu();

}

// changes in function

generateBulletEnemy1();

generateBulletEnemy2();

generateBulletEnemy3();

generateBulletEnemy4();

moveBulletEnemy();

moveBulletEnemy2();

moveBulletEnemy3();

moveBulletEnemy4();

moveBullet();

bulletCollisionWithEnemy();

moveBulletLeft();

moveBulletUP();

moveBulletDOWN();

bulletCollisionWithEnemyLeft();

bulletCollisionWithEnemyUp();

bulletCollisionWithEnemyDown();

bulletCollisionWithPlayer();

bulletCollisionWithPlayerLeft();

bulletCollisionWithPlayerUp();

bulletCollisionWithPlayerDown();

bulletCollisionWithChase();

bulletCollisionWithChaseLeft();

bulletCollisionWithChaseUp();

bulletCollisionWithChaseDown();

enemyWithPlayerRight();

enemyWithPlayerUp();

enemyWithPlayerDown();

enemyWithPlayerLeft();

}

// increase credits

int energyIncrease(int energy)

{

int result = energy;

result++;

return result;

}

int addScore()

{

score = score + 1;

return score;

}

int health()

{

int result;

health1 = health1 - 1;

if (health1 == 990)

{

health1 = health1 % 10;

}

health1 = health1 - 1;

return health1;

}

int life()

{

lives = lives - 1;

if (lives == 0)

{

cout << " You Fail.";

gameRunning = false;

clearScreen();

header();

option = menu();

}

return lives;

}

int addBonus()

{

bonus = bonus + 5;

return bonus;

}

// player move left

void moveLeft()

{

char nextLocation = getCharAtxy(px - 1, py);

if(nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

maze[py -1][px] = ' ';

}

if (nextLocation == ' ' || nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

erase(px, py);

px = px - 1;

printball(px, py);

}

if (nextLocation == '$')

{

energy = energyIncrease(energy);

gotoxy(120, 21);

cout << "Energy : " << energy;

}

if (nextLocation == '>')

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (nextLocation == '@')

{

bonus = addBonus();

gotoxy(120, 23);

cout << "Bonus: " << bonus;

erase(px, py);

gotoxy(px = 14, py = 6);

}

if (nextLocation == defender3 || nextLocation == defender4 || nextLocation == goal)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

void moveRight()

{

char nextLocation = getCharAtxy(px + 3, py);

if(nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

maze[py][px +3] = ' ';

}

if (nextLocation == ' ' || nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

erase(px, py);

px = px + 1;

printball(px, py);

}

if (nextLocation == '$')

{

energy = energyIncrease(energy);

gotoxy(120, 21);

cout << "Energy : " << energy;

}

if (nextLocation == '>')

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (nextLocation == '@')

{

bonus = addBonus();

gotoxy(120, 23);

cout << "Bonus: " << bonus;

erase(px, py);

gotoxy(px = 14, py = 6);

}

if (nextLocation == defender3 || nextLocation == defender4 || nextLocation == goal)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

void moveUp()

{

char nextLocation = getCharAtxy(px, py - 1);

if( nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

maze[py][px - 4] = ' ';

}

if (nextLocation == ' ' || nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

erase(px, py);

py = py - 1;

printball(px, py);

}

if (nextLocation == '$')

{

energy = energyIncrease(energy);

gotoxy(120, 21);

cout << "Energy : " << energy;

}

if (nextLocation == '>')

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (nextLocation == '@')

{

bonus = addBonus();

gotoxy(120, 23);

cout << "Bonus: " << bonus;

erase(px, py);

gotoxy(px = 14, py = 6);

}

if (nextLocation == defender3 || nextLocation == defender4 || nextLocation == goal)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

void moveDown()

{

char nextLocation = getCharAtxy(px, py + 1);

if(nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

maze[py][px ] = ' ';

}

if (nextLocation == ' ' || nextLocation == '>' || nextLocation == '$' || nextLocation == '?')

{

erase(px, py);

py = py + 1;

printball(px, py);

}

if (nextLocation == '$')

{

energy = energyIncrease(energy);

gotoxy(120, 21);

cout << "Energy : " << energy;

}

if (nextLocation == '>')

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (nextLocation == defender3 || nextLocation == defender4 || nextLocation == goal)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

if (nextLocation == '@')

{

bonus = addBonus();

gotoxy(120, 23);

cout << "Bonus: " << bonus;

erase(px, py);

gotoxy(px = 14, py = 6);

}

}

void clear(int x, int y, char previous)

{

gotoxy(x, y);

cout << previous;

}

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);

}

// player bullet left

void generateBullet()

{

bulletX[bulletCount] = px + 3;

bulletY[bulletCount] = py;

isBulletActive[bulletCount] = true;

gotoxy(px + 3, py);

cout << "-";

bulletCount++;

}

void removeBulletFromArray(int index)

{

for (int x = index; x < bulletCount - 1; x++)

{

bulletX[x] = bulletX[x + 1];

bulletY[x] = bulletY[x + 1];

}

bulletCount--;

}

void moveBullet()

{

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

{

if (isBulletActive[x] == true)

{

char next = getCharAtxy(bulletX[x] + 1, bulletY[x] + 1);

if (next != ' ')

{

eraseBullet(bulletX[x], bulletY[x]);

makeBulletInactive(x);

}

else

{

eraseBullet(bulletX[x], bulletY[x]);

bulletX[x] = bulletX[x] + 1;

printBullet(bulletX[x], bulletY[x]);

}

}

}

}

void printBullet(int x, int y)

{

int k = 4;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

gotoxy(x, y);

cout << "-";

}

void eraseBullet(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void makeBulletInactive(int index)

{

isBulletActive[index] = false;

}

// print bullet right

void generateBulletLeft()

{

bulletRX[bulletCountR] = px - 1;

bulletRY[bulletCountR] = py;

isBulletActiveR[bulletCountR] = true;

gotoxy(px - 1, py);

cout << "-";

bulletCountR++;

}

void removeBulletFromArrayLeft(int index)

{

for (int x = index; x < bulletCountR - 1; x++)

{

bulletRX[x] = bulletRX[x + 1];

bulletRY[x] = bulletRY[x + 1];

}

bulletCountR--;

}

void moveBulletLeft()

{

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

{

if (isBulletActiveR[x] == true)

{

char next = getCharAtxy(bulletRX[x] - 1, bulletRY[x] - 1);

if (next != ' ')

{

eraseBulletLeft(bulletRX[x], bulletRY[x]);

makeBulletInactiveLeft(x);

}

else

{

eraseBulletLeft(bulletRX[x], bulletRY[x]);

bulletRX[x] = bulletRX[x] - 1;

printBulletLeft(bulletRX[x], bulletRY[x]);

}

}

}

}

void printBulletLeft(int x, int y)

{

gotoxy(x, y);

cout << "-";

}

void eraseBulletLeft(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void makeBulletInactiveLeft(int index)

{

isBulletActiveR[index] = false;

}

// print bullet up

void generateBulletUP()

{

bulletLX[bulletCountL] = px;

bulletLY[bulletCountL] = py - 1;

isBulletActiveL[bulletCountL] = true;

gotoxy(px, py - 1);

cout << "-";

bulletCountL++;

}

void removeBulletFromArrayUP(int index)

{

for (int x = index; x < bulletCountL - 1; x++)

{

bulletLX[x] = bulletLX[x + 1];

bulletLY[x] = bulletLY[x + 1];

}

bulletCountL--;

}

void moveBulletUP()

{

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

{

if (isBulletActiveL[x] == true)

{

char next = getCharAtxy(bulletLX[x] + 1, bulletLY[x] - 1);

if (next != ' ')

{

eraseBulletUP(bulletLX[x], bulletLY[x]);

makeBulletInactiveUP(x);

}

else

{

eraseBulletUP(bulletLX[x], bulletLY[x]);

bulletLY[x] = bulletLY[x] - 1;

printBulletUP(bulletLX[x], bulletLY[x]);

}

}

}

}

void printBulletUP(int x, int y)

{

gotoxy(x, y);

cout << "-";

}

void eraseBulletUP(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void makeBulletInactiveUP(int index)

{

isBulletActiveL[index] = false;

}

// print bullet down

void generateBulletDOWN()

{

bulletDX[bulletCountD] = px;

bulletDY[bulletCountD] = py + 1;

isBulletActiveD[bulletCountD] = true;

gotoxy(px, py + 1);

cout << "-";

bulletCountD++;

}

void removeBulletFromArrayDOWN(int index)

{

for (int x = index; x < bulletCountD - 1; x++)

{

bulletDX[x] = bulletDX[x + 1];

bulletDY[x] = bulletDY[x + 1];

}

bulletCountD--;

}

void moveBulletDOWN()

{

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

{

if (isBulletActiveD[x] == true)

{

char next = getCharAtxy(bulletDX[x] + 1, bulletDY[x] + 1);

if (next != ' ')

{

eraseBulletDOWN(bulletDX[x], bulletDY[x]);

makeBulletInactiveDOWN(x);

}

else

{

eraseBulletDOWN(bulletDX[x], bulletDY[x]);

bulletDY[x] = bulletDY[x] + 1;

printBulletDOWN(bulletDX[x], bulletDY[x]);

}

}

}

}

void printBulletDOWN(int x, int y)

{

gotoxy(x, y);

cout << "-";

}

void eraseBulletDOWN(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void makeBulletInactiveDOWN(int index)

{

isBulletActiveD[index] = false;

}

// enemy bullet up

void generateBulletEnemy1()

{

bullet2X[bulletCounter] = enemyX;

bullet2Y[bulletCounter] = enemyY;

isBulletActiveEnemy[bulletCounter] = true;

gotoxy(enemyX, enemyY);

cout << "\*";

bulletCounter++;

}

void makeBulletInactiveEnemy1(int index)

{

isBulletActiveEnemy[index] = false;

}

void printBulletEnemy1(int enemyX, int enemyY)

{

int k = 6;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

gotoxy(enemyX, enemyY);

cout << "\*";

}

void eraseBulletEnemy1(int enemyX, int enemyY)

{

gotoxy(enemyX, enemyY);

cout << " ";

}

void moveBulletEnemy()

{

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

{

if (isBulletActiveEnemy[x] == true)

{

char next = getCharAtxy(bullet2X[x] - 1, bullet2Y[x]);

if (next != ' ')

{

eraseBulletEnemy1(bullet2X[x], bullet2Y[x]);

makeBulletInactiveEnemy1(x);

}

else

{

eraseBulletEnemy1(bullet2X[x], bullet2Y[x]);

bullet2X[x] = bullet2X[x] - 1;

printBulletEnemy1(bullet2X[x], bullet2Y[x]);

}

}

}

}

// print defender

void printDefender(int x, int y)

{

int k = 5;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

gotoxy(x, y);

cout << defender3;

}

void eraseDefender(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void defender2()

{

string direction = "Down";

if (direction == "Up")

{

char next = getCharAtxy(enemyX, enemyY - 1);

if (next == ' ' || next == '>' || next == '$')

{

eraseDefender(enemyX, enemyY);

enemyY--;

printDefender(enemyX, enemyY);

}

if (next == '#' || next == '%')

{

direction = "Down";

eraseDefender(enemyX, enemyY);

enemyY = 6;

}

}

if (direction == "Down")

{

char next = getCharAtxy(enemyX, enemyY + 1);

if (next == ' ' || next == '>' || next == '$')

{

eraseDefender(enemyX, enemyY);

enemyY++;

printDefender(enemyX, enemyY);

}

if (next == '#' || next == '%')

{

direction = "Up";

eraseDefender(enemyX, enemyY);

enemyY = 6;

}

}

}

// move bullet right

void generateBulletEnemy2()

{

bullet3X[bulletCounter2] = enemyX;

bullet3Y[bulletCounter2] = enemyY;

isBulletActiveEnemy2[bulletCounter2] = true;

gotoxy(enemyX, enemyY);

cout << "\*";

bulletCounter2++;

}

void makeBulletInactiveEnemy2(int index)

{

isBulletActiveEnemy2[index] = false;

}

void printBulletEnemy2(int enemyX, int enemyY)

{

gotoxy(enemyX, enemyY);

cout << "\*";

}

void eraseBulletEnemy2(int enemyX, int enemyY)

{

gotoxy(enemyX, enemyY);

cout << " ";

}

void moveBulletEnemy2()

{

for (int x = 0; x < bulletCounter2; x++)

{

if (isBulletActiveEnemy2[x] == true)

{

char next = getCharAtxy(bullet3X[x] + 1, bullet3Y[x]);

if (next != ' ')

{

eraseBulletEnemy2(bullet3X[x], bullet3Y[x]);

makeBulletInactiveEnemy2(x);

}

else

{

eraseBulletEnemy2(bullet3X[x], bullet3Y[x]);

bullet3X[x] = bullet3X[x] + 1;

printBulletEnemy2(bullet3X[x], bullet3Y[x]);

}

}

}

}

// print enemy2

void printDefender2(int x, int y)

{

gotoxy(x, y);

cout << defender4;

}

void eraseDefender2(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

// move enemy2

void defender()

{

string direction = "Right";

if (direction == "Left")

{

char next = getCharAtxy(enemy2X - 1, enemy2Y);

if (next == ' ' || next == '>' || next == '$')

{

eraseDefender2(enemy2X, enemy2Y);

enemy2X = enemy2X - 1;

printDefender2(enemy2X, enemy2Y);

}

if (next == '#' || next == '%' || next == '|')

{

direction = "Right";

eraseDefender2(enemy2X, enemy2Y);

enemy2X = 11;

}

}

if (direction == "Right")

{

char next = getCharAtxy(enemy2X + 1, enemy2Y);

if (next == ' ' || next == '>' || next == '$')

{

eraseDefender2(enemy2X, enemy2Y);

enemy2X = enemy2X + 1;

printDefender2(enemy2X, enemy2Y);

}

if (next == '#' || next == '%' || next == '|')

{

direction = "Left";

eraseDefender2(enemy2X, enemy2Y);

enemy2X = 11;

}

}

}

// move bullet down

void generateBulletEnemy3()

{

bullet4X[bulletCounter3] = enemy2X;

bullet4Y[bulletCounter3] = enemy2Y;

isBulletActiveEnemy3[bulletCounter3] = true;

gotoxy(enemy2X, enemy2Y);

cout << "\*";

bulletCounter3++;

}

void makeBulletInactiveEnemy3(int index)

{

isBulletActiveEnemy3[index] = false;

}

void printBulletEnemy3(int enemy2X, int enemy2Y)

{

gotoxy(enemy2X, enemy2Y);

cout << "\*";

}

void eraseBulletEnemy3(int enemy2X, int enemy2Y)

{

gotoxy(enemy2X, enemy2Y);

cout << " ";

}

void moveBulletEnemy3()

{

for (int x = 0; x < bulletCounter3; x++)

{

if (isBulletActiveEnemy3[x] == true)

{

char next = getCharAtxy(bullet4X[x], bullet4Y[x] + 1);

if (next != ' ')

{

eraseBulletEnemy3(bullet4X[x], bullet4Y[x]);

makeBulletInactiveEnemy3(x);

}

else

{

eraseBulletEnemy3(bullet4X[x], bullet4Y[x]);

bullet4Y[x] = bullet4Y[x] + 1;

printBulletEnemy3(bullet4X[x], bullet4Y[x]);

}

}

}

}

// move bullet up

void generateBulletEnemy4()

{

bullet5X[bulletCounter4] = enemy2X;

bullet5Y[bulletCounter4] = enemy2Y;

isBulletActiveEnemy4[bulletCounter4] = true;

gotoxy(enemy2X, enemy2Y);

cout << "\*";

bulletCounter4++;

}

void makeBulletInactiveEnemy4(int index)

{

isBulletActiveEnemy4[index] = false;

}

void printBulletEnemy4(int enemy2X, int enemy2Y)

{

gotoxy(enemy2X, enemy2Y);

cout << "\*";

}

void eraseBulletEnemy4(int enemy2X, int enemy2Y)

{

gotoxy(enemy2X, enemy2Y);

cout << " ";

}

void moveBulletEnemy4()

{

for (int x = 0; x < bulletCounter4; x++)

{

if (isBulletActiveEnemy4[x] == true)

{

char next = getCharAtxy(bullet5X[x], bullet5Y[x] - 1);

if (next != ' ')

{

eraseBulletEnemy4(bullet5X[x], bullet5Y[x]);

makeBulletInactiveEnemy4(x);

}

else

{

eraseBulletEnemy4(bullet5X[x], bullet5Y[x]);

bullet5Y[x] = bullet5Y[x] - 1;

printBulletEnemy4(bullet5X[x], bullet5Y[x]);

}

}

}

}

// mmove and print goal keeper

void printGoalKeeper(int x, int y)

{

int k = 12;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, k);

gotoxy(x, y);

cout << goal;

}

void eraseGoalKeeper(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void goalKeeper()

{

string direction = "Down";

if (direction == "Up")

{

char next = getCharAtxy(goalX, goalY - 1);

if (next == ' ' || next == '>' || next == '$')

{

eraseGoalKeeper(goalX, goalY);

goalY--;

printGoalKeeper(goalX, goalY);

}

if (next == '#')

{

direction = "Down";

eraseGoalKeeper(goalX, goalY);

goalY = 6;

}

}

if (direction == "Down")

{

char next = getCharAtxy(goalX, goalY + 1);

if (next == ' ' || next == '>' || next == '$')

{

eraseGoalKeeper(goalX, goalY);

goalY++;

printGoalKeeper(goalX, goalY);

}

if (next == '#')

{

direction = "Up";

eraseGoalKeeper(goalX, goalY);

goalY = 6;

}

}

}

// collision player bullet

void bulletCollisionWithEnemy()

{

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

{

if (isBulletActive[x] == true)

{

if ((bulletX[x] + 1 == enemyX && (bulletY[x] == enemyY)) || (bulletX[x] + 1 == enemy2X && (bulletY[x] == enemy2Y)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (((enemyX - 1 == bulletX[x] && enemyY + 1 == bulletY[x])) || ((enemy2X - 1 == bulletX[x] && enemy2Y + 1 == bulletY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

}

}

}

// bullet collision chase

void bulletCollisionWithChase()

{

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

{

if (isBulletActive[x] == true)

{

if ((bulletX[x] + 1 == dx && (bulletY[x] == dy)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

}

if (((dx - 1 == bulletX[x] && dy + 1 == bulletY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

}

}

void bulletCollisionWithChaseLeft()

{

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

{

if (isBulletActive[x] == true)

{

if ((bulletRX[x] - 1 == dx && (bulletRY[x] == dy)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

if (((dx + 1 == bulletRX[x] && dy + 1 == bulletRY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

}

}

}

void bulletCollisionWithChaseUp()

{

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

{

if (isBulletActiveL[x] == true)

{

if (bulletLX[x] == dx && (bulletLY[x] - 1 == dy))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

if (((dx + 1 == bulletLX[x] && dy - 1 == bulletLY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

}

}

}

void bulletCollisionWithChaseDown()

{

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

{

if (isBulletActiveL[x] == true)

{

if ((bulletRX[x] - 1 == dx && (bulletRY[x] == dy)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

if (((dx + 1 == bulletRX[x] && dy + 1 == bulletRY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

gotoxy(dx = 20, dy = 15);

}

}

}

}

void bulletCollisionWithEnemyLeft()

{

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

{

if (isBulletActiveR[x] == true)

{

if ((bulletRX[x] - 1 == enemyX && (bulletRY[x] == enemyY)) || (bulletRX[x] - 1 == enemy2X && (bulletRY[x] == enemy2Y)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (((enemyX + 1 == bulletRX[x] && enemyY + 1 == bulletRY[x])) || ((enemy2X + 1 == bulletRX[x] && enemy2Y + 1 == bulletRY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

}

}

}

// bullet collision enemy

void bulletCollisionWithEnemyUp()

{

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

{

if (isBulletActiveL[x] == true)

{

if ((bulletLX[x] == enemyX && (bulletLY[x] - 1 == enemyY)) || (bulletLX[x] == enemy2X && (bulletLY[x] - 1 == enemy2Y)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (((enemyX + 1 == bulletLX[x] && enemyY - 1 == bulletLY[x])) || ((enemy2X + 1 == bulletLX[x] && enemy2Y - 1 == bulletLY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

}

}

}

void bulletCollisionWithEnemyDown()

{

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

{

if (isBulletActiveD[x] == true)

{

if ((bulletDX[x] == enemyX && (bulletDY[x] + 1 == enemyY)) || (bulletDX[x] == enemy2X && (bulletDY[x] + 1 == enemyY)))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

if (((enemyX + 1 == bulletDX[x] && enemyY + 1 == bulletDY[x])) || ((enemy2X + 1 == bulletDX[x] && enemy2Y + 1 == bulletDY[x])))

{

score = addScore();

gotoxy(120, 19);

cout << "Score: " << score;

}

}

}

}

void bulletCollisionWithPlayer()

{

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

{

if (isBulletActiveEnemy[x] == true)

{

if (bullet2X[x] + 1 == px && bullet2Y[x] == py)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

if (px + 1 == bullet2X[x] && py + 1 == bullet2Y[x])

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

}

}

void bulletCollisionWithPlayerLeft()

{

for (int x = 0; x < bulletCounter2; x++)

{

if (isBulletActiveEnemy2[x] == true)

{

if (bullet3X[x] - 1 == px && bullet3Y[x] == py)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

if (px + 1 == bullet3X[x] && py + 1 == bullet3Y[x])

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

}

}

void bulletCollisionWithPlayerUp()

{

for (int x = 0; x < bulletCounter3; x++)

{

if (isBulletActiveEnemy3[x] == true)

{

if (bullet4X[x] == px && bullet4Y[x] + 1 == py)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

if (px + 1 == bullet4X[x] && py - 1 == bullet4Y[x])

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

}

}

void bulletCollisionWithPlayerDown()

{

for (int x = 0; x < bulletCounter4; x++)

{

if (isBulletActiveEnemy4[x] == true)

{

if (bullet5X[x] == px && bullet5Y[x] + 1 == py)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

if (px + 1 == bullet5X[x] && py - 1 == bullet5Y[x])

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

}

}

// chasing enemy move

void eraseEnemy(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void printEnemy(int x, int y)

{

gotoxy(x, y);

cout << defender5;

}

void dirctionEnemy()

{

direction = sqrt(pow((dx - px), 2) + pow((dy - py), 2));

if (direction <= chasingDistance)

{

moveEnemy();

}

}

void moveEnemy()

{

if (px < dx)

{

moveEnemyLeft();

}

else if (px > dx)

{

moveEnemyRight();

}

else if (py < dy)

{

moveEnemyUp();

}

else if (py > dy)

{

moveEnemyDown();

}

}

void moveEnemyLeft()

{

char nextLocation = getCharAtxy(dx - 1, dy);

if (nextLocation == '%' || nextLocation == '$' || nextLocation == '>' || nextLocation == ball[3] || nextLocation == '|' || nextLocation == '#' || nextLocation == '\_')

{

eraseEnemy(dx, dy);

dy = dy + 1;

printEnemy(dx, dy);

}

else

{

eraseEnemy(dx, dy);

dx = dx - 3;

printEnemy(dx, dy);

}

}

void moveEnemyRight()

{

char nextLocation = getCharAtxy(dx + 1, dy);

if (nextLocation == '%' || nextLocation == '$' || nextLocation == '>' || nextLocation == ball[3] || nextLocation == '|' || nextLocation == '#' || nextLocation == '\_')

{

eraseEnemy(dx, dy);

dy = dy - 1;

printEnemy(dx, dy);

}

else

{

eraseEnemy(dx, dy);

dx = dx + 1;

printEnemy(dx, dy);

}

}

void moveEnemyUp()

{

char nextLocation = getCharAtxy(dx, dy - 1);

if (nextLocation == '%' || nextLocation == '$' || nextLocation == '>' || nextLocation == ball[3] || nextLocation == '|' || nextLocation == '# ' || nextLocation == '\_')

{

eraseEnemy(dx, dy);

dx = dx - 1;

printEnemy(dx, dy);

}

else

{

eraseEnemy(dx, dy);

dy = dy - 1;

printEnemy(dx, dy);

}

}

void moveEnemyDown()

{

char nextLocation = getCharAtxy(dx, dy + 1);

if (nextLocation == '%' || nextLocation == '$' || nextLocation == '>' || nextLocation == ball[3] || nextLocation == '|' || nextLocation == '# ' || nextLocation == '\_')

{

eraseEnemy(dx, dy);

dx = dx + 1;

printEnemy(dx, dy);

}

else

{

eraseEnemy(dx, dy);

dy = dy + 1;

printEnemy(dx, dy);

}

}

// collision player chasing

void enemyWithPlayerRight()

{

if (px + 1 == dx && py == dy)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

if (dx + 1 == px && dy + 1 == py)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

}

void enemyWithPlayerLeft()

{

if (px - 1 == dx && py == dy)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

if (dx + 1 == px && dy + 1 == py)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

}

void enemyWithPlayerUp()

{

if (px == dx && py + 1 == dy)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

if (dx + 1 == px && dy - 1 == py)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

}

void enemyWithPlayerDown()

{

if (px == dx && py + 1 == dy)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

if (dx + 1 == px && dy - 1 == py)

{

lives = life();

gotoxy(120, 22);

cout << "Lives :" << lives;

erase(px, py);

eraseEnemy(dx, dy);

gotoxy(px = 14, py = 6);

gotoxy(dx = 20, dy = 15);

}

}

// collision with enimies

void playerCollision()

{

if (px == enemyX || px == enemy2X || py == enemyY || py == enemy2Y || px == goalX || py == goalY)

{

health1 = health();

gotoxy(120, 20);

cout << "Health : " << health1;

}

}

// file with maze store in it

void loadMaze()

{

fstream file;

string word;

int row = 0;

file.open("mazes.txt", ios ::in);

while (getline(file, word))

{

for (int x = 0; x < word.length(); x++)

{

maze[row][x] = word[x];

}

row++;

}

file.close();

}

// file to store updated

void storePause()

{

maze[6][15] = ' ';

fstream file;

file.open("mazes1.txt", ios ::out);

for (int row = 0; row < 36; row++)

{

for (int col = 0; col < 114; col++)

{

file << maze[row][col];

}

file << endl;

}

file.close();

}

void loadPause()

{

fstream file;

string word;

int row = 0;

file.open("mazes1.txt", ios ::in);

while (getline(file, word))

{

for (int x = 0; x < word.length(); x++)

{

maze[row][x] = word[x];

}

row++;

}

file.close();

}

// file to store credientals

void storeScore()

{

fstream file;

file.open("score.txt", ios ::out);

file << score << ",";

file << energy << ",";

file << health1 << ",";

file << lives << ",";

file << bonus << endl;

file.close();

}

void loadScore()

{

fstream file;

string word;

file.open("score.txt", ios ::in);

while (getline(file, word))

{

stringstream temp(parseScore(word, 1));

temp >> score;

stringstream temp1(parseScore(word, 2));

temp1 >> energy;

stringstream temp2(parseScore(word, 3));

temp2 >> health1;

stringstream temp3(parseScore(word, 4));

temp3 >> lives;

stringstream temp4(parseScore(word, 5));

temp4 >> bonus;

}

file.close();

}

string parseScore(string record, int field)

{

int commaCounter1 = 1;

string score;

for (int x = 0; x < record.length(); x++)

{

if (record[x] == ',')

{

commaCounter1 = commaCounter1 + 1;

}

else if (commaCounter1 == field)

{

score = score + record[x];

}

}

return score;

}

// file to store location

void storeLocation()

{

fstream file;

file.open("location.txt", ios ::out);

file << px << ",";

file << py << ",";

file << dx << ",";

file << dy << ",";

file << enemyX << ",";

file << enemyY << ",";

file << enemy2X << ",";

file << enemy2Y << ",";

file << goalX << ",";

file << goalY << endl;

file.close();

}

void loadLocation()

{

fstream file;

string word;

file.open("location.txt", ios ::in);

while (getline(file, word))

{

stringstream temp(parseScore(word, 1));

temp >> py;

stringstream temp1(parseScore(word, 2));

temp1 >> py;

stringstream temp2(parseScore(word, 3));

temp2 >> dx;

stringstream temp3(parseScore(word, 4));

temp3 >> dy;

stringstream temp4(parseScore(word, 5));

temp4 >> enemyX;

stringstream temp5(parseScore(word, 6));

temp5 >> enemyY;

stringstream temp6(parseScore(word, 7));

temp6 >> enemy2X;

stringstream temp7(parseScore(word, 8));

temp7 >> enemy2Y;

stringstream temp8(parseScore(word, 9));

temp8 >> goalX;

stringstream temp9(parseScore(word, 10));

temp9 >> goalY;

}

file.close();

}

string parseLocation(string record, int field)

{

int commaCounter1 = 1;

string score;

for (int x = 0; x < record.length(); x++)

{

if (record[x] == ',')

{

commaCounter1 = commaCounter1 + 1;

}

else if (commaCounter1 == field)

{

score = score + record[x];

}

}

return score;

}

// file to store location bullet

void storeBullet()

{

fstream file;

file.open("bullet.txt", ios ::out);

// for (int x =0 ; x < bulletCounter ; x++)

//{

file << bulletX[bulletCount] << ",";

file << bulletY[bulletCount] << ",";

file << bullet2X[bulletCounter] << ",";

file << bullet2Y[bulletCounter] << ",";

file << bulletRX[bulletCountR] << ",";

file << bulletRY[bulletCountR] << ",";

file << bulletLX[bulletCountL] << ",";

file << bulletLY[bulletCountL] << ",";

file << bulletDX[bulletCountD] << ",";

file << bulletDY[bulletCountD] << ",";

file << bullet3X[bulletCounter2] << ",";

file << bullet3Y[bulletCounter2] << ",";

file << bullet4X[bulletCounter3] << ",";

file << bullet4Y[bulletCounter3] << ",";

file << bullet5X[bulletCounter4] << ",";

file << bullet5Y[bulletCounter4] << endl;

file.close();

}

void loadBullet()

{

fstream file;

string word;

file.open("bullet.txt", ios ::in);

while (getline(file, word))

{

stringstream temp(parseScore(word, 1));

temp >> bulletX[bulletCount];

stringstream temp1(parseScore(word, 2));

temp1 >> bulletY[bulletCount];

stringstream temp2(parseScore(word, 3));

temp2 >> bullet2X[bulletCounter];

stringstream temp3(parseScore(word, 4));

temp3 >> bullet2Y[bulletCounter];

stringstream temp4(parseScore(word, 5));

temp4 >> bulletRX[bulletCountR];

stringstream temp5(parseScore(word, 6));

temp5 >> bulletRY[bulletCountR];

stringstream temp6(parseScore(word, 7));

temp6 >> bulletLX[bulletCountL];

stringstream temp7(parseScore(word, 8));

temp7 >> bulletLY[bulletCountL];

stringstream temp8(parseScore(word, 9));

temp8 >> bulletDX[bulletCountD];

stringstream temp9(parseScore(word, 10));

temp9 >> bulletDY[bulletCountD];

stringstream tempO(parseScore(word, 11));

tempO >> bullet3X[bulletCounter2];

stringstream tempR(parseScore(word, 12));

tempR >> bullet3Y[bulletCounter2];

stringstream tempo(parseScore(word, 13));

tempo >> bullet4X[bulletCounter3];

stringstream tempa(parseScore(word, 14));

tempa >> bullet4Y[bulletCounter3];

stringstream tempr(parseScore(word, 15));

tempr >> bullet5X[bulletCounter4];

stringstream temp0(parseScore(word, 16));

temp0 >> bullet5Y[bulletCounter4];

}

file.close();

}

string parseBullet(string record, int field)

{

int commaCounter1 = 1;

string score;

for (int x = 0; x < record.length(); x++)

{

if (record[x] == ',')

{

commaCounter1 = commaCounter1 + 1;

}

else if (commaCounter1 == field)

{

score = score + record[x];

}

}

return score;

}

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria 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 **Short Description and Story Writing of Game** - **Game Characters** Description - **Rules** & Interactions - **Goal** of the Game **- Screenshot** of the Game - **Data Structures** Used in the Game - **Functions** Prototype - **Full Code** | | | | |
| Project Complexity  **Grade:** | Project has at least 1 Player and 3 enemies. Proper use of gotoxy() function. Health system, Firing System and lives decreasing system. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Randomness  **Grade:** | Objects are produced randomly in the game. | meet more than 80% of the criteria given. | meet more than 50% of the criteria given. | Objects are appearing in the same pattern |
| Code Style  **Grade:** | All Code style criteria 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. |
| Idea Novelty and Creativity  **Grade:** | Idea is unique of the game | Idea is merged by combining other different games | Same idea as a previous game | Could not implement the existing game idea. |
| Data Structure (2D Arrays)  **Grade:** | 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. |
| File Handling  **Grade:** | Game maze is loaded and the updated maze is stored in the file | Game maze is loaded and partial data is stored in the file. | Game maze is just loaded but the updated game configuration is not stored in the maze. | Project do not contain file handling |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- There is no global variable defined. Arrays and variables are passed as parameters to the functions. Functions exhibit single responsibility principle. | | | | |
| Screen flickering  **Grade:** | There is no Screen flickering. | Maze is not flickering but the characters are flickering at great speed | Flickering is done at lot of places | Screen is flickering at all places |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | 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. |

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria 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 **Short Description and Story Writing of Game** - **Game Characters** Description - **Rules** & Interactions - **Goal** of the Game **- Screenshot** of the Game - **Data Structures** Used in the Game - **Functions** Prototype - **Full Code** | | | | |
| Project Complexity  **Grade:** | Project has at least 1 Player and 3 enemies. Proper use of gotoxy() function. Health system, Firing System and lives decreasing system. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Randomness  **Grade:** | Objects are produced randomly in the game. | meet more than 80% of the criteria given. | meet more than 50% of the criteria given. | Objects are appearing in the same pattern |
| Code Style  **Grade:** | All Code style criteria 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. |
| Idea Novelty and Creativity  **Grade:** | Idea is unique of the game | Idea is merged by combining other different games | Same idea as a previous game | Could not implement the existing game idea. |
| Data Structure (2D Arrays)  **Grade:** | 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. |
| File Handling  **Grade:** | Game maze is loaded and the updated maze is stored in the file | Game maze is loaded and partial data is stored in the file. | Game maze is just loaded but the updated game configuration is not stored in the maze. | Project do not contain file handling |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- There is no global variable defined. Arrays and variables are passed as parameters to the functions. Functions exhibit single responsibility principle. | | | | |
| Screen flickering  **Grade:** | There is no Screen flickering. | Maze is not flickering but the characters are flickering at great speed | Flickering is done at lot of places | Screen is flickering at all places |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | 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. |

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