WAR ZONE BATTLE

Session 2024 – 2028

Submitted by:

SUBHAN MALIK 2024-CS-16

Supervised by:

Dr.Awais Hassan

Course:

CSC-102 Programming Fundamentals

Department of Computer Science

University of Engineering and Technology, Lahore Pakistan

Contents

1. Short Description and Story of the Game:

Short Description:

The game is an action-packed, arcade-style combat game where the player controls a tank-like character, navigating through a maze filled with enemies, machine guns, and obstacles. The goal is to survive waves of enemy attacks, destroy all enemies, and reach the finish line while maintaining the player's health and earning a high score. The gameplay includes shooting mechanics, power-ups, and tactical movement to avoid enemy fire.

Story:

In a futuristic battlefield, the world is under siege by hostile forces that have deployed advanced enemy tanks, drones, and machine guns to secure dominance. The player assumes the role of a skilled pilot tasked with infiltrating the enemy's fortified maze-like base. Armed with a powerful combat vehicle and limited ammunition, the player must strategically navigate the terrain, engage hostile units, and neutralize threats.

The mission intensifies as the player faces two elite enemy tanks, advanced machine gun units, and unpredictable bullet patterns. Health restoration items occasionally appear, giving the player a chance to recover. With the survival of the resistance depending on this mission, the player must eliminate all enemies and disable their fortifications to secure victory. Victory is achieved not only through firepower but also through tactical evasion, accurate shooting, and strategic movement.

The clock is ticking, and the enemies are relentless—will you emerge victorious, or will the maze claim another hero?

2. Game Characters Description

Player (Main Character)

Description: The player controls a tank-like combat vehicle equipped with a powerful weapon. It features a futuristic design and is highly maneuverable within the maze.

Abilities:

- Moves in four directions (up, down, left, right) to navigate the maze. ○

  Shoots projectiles to destroy enemies.
- Can collect health power-ups to restore its health.

Health: Starts with 200 points and loses health upon enemy bullet hits or collisions.

Goal: Survive enemy attacks, eliminate all threats, and complete the mission.

Enemy 1 (Tank Unit)

Description: A robotic enemy tank equipped with a large gun. It moves vertically within a designated area, posing a significant threat to the player.

Abilities: ○ Moves up and down. ○ Shoots bullets when aligned with the player horizontally.

- ○ Reverses direction upon collisions with obstacles or the player.

Health: Starts with 100 points.

Behavior: Aggressively tries to collide or shoot the player.

- **Enemy 2 (Elite Tank Unit)**

Description: A more advanced and armored version of Enemy 1, designed to challenge the player further.

Abilities:

- ○ Moves up and down within its zone. ○ Shoots bullets when aligned with the player horizontally.
- ○ Reverses direction based on boundaries or collisions.

Health: Starts with 100 points.

Behavior: Strategically mirrors the player's movements to increase difficulty.

- **Machine Gun 1**

Description: A stationary machine gun mounted on the battlefield, which moves horizontally between fixed boundaries.

Abilities:

- ○ Fires a barrage of bullets when the player is within its vertical line of sight.
- ○ Moves left and right within its zone.

Health: Starts with 50 points.

Behavior: Attempts to block the player’s path and deal consistent damage with rapid fire.

- **Machine Gun 2**

Description: A mobile, heavily armored machine gun that moves diagonally within the maze.

Abilities:

- ○ Moves in diagonal patterns (up-left, up-right, down-left, down-right).
- ○ Fires bullets when the player aligns with its vertical path.

Health: Starts with 50 points.

Behavior: Unpredictable movements make it challenging to hit, and it constantly adapts its position to target the player.

- **Pill (Health Power-Up)**

Description: A glowing power-up symbolized by "(+)" that appears randomly in the maze.

Abilities:

- Restores the player's health to full when collected.

Behavior: Appears at random positions and remains active until collected or replaced by another.

3. Game Objects Description

1. Bullets

Description: The primary weapon for both the player and enemies. Bullets are small, fastmoving projectiles that inflict damage upon collision.

Behavior:

- Fired by the player, enemies, and machine guns. ○ Moves in a straight line until hitting an obstacle or leaving the screen.
- Causes damage to the target upon collision.

2. Pile (Health Power-Up)

Description: A glowing, collectible object represented as "(+)" on the screen.

Behavior:

- Spawns randomly within the maze. ○ Restores the player's health to full when collected.
- Disappears once collected or replaced by another power-up.

3. Walls and Obstacles

Description: Stationary objects forming the maze layout, depicted as `#`, `|`, or blocks of characters.

Behavior:

- Block movement and bullets, acting as barriers.
- Serve as strategic cover during combat.

4. Maze Finish Line

Description: The end point of the maze, signifying the player's goal.

Behavior:

- Acts as the final objective of the level.
- Reaching it after eliminating all enemies marks the player's victory.

5. Enemy Tanks

Description: Hostile units (Enemy 1 and Enemy 2) with their unique appearances and mechanics.

Behavior:

- Moves within predefined paths.
- Fires bullets and attempts to collide with the player to reduce health.

6. Machine Guns

Description: Static or mobile turrets represented visually with "o===[0]".

Behavior:

- Machine Gun 1 moves horizontally. o Machine Gun 2 moves diagonally.
- Both fire bullets when the player is within their line of sight.

7. Score Counter

Description: A non-physical object displayed on the screen showing the player's current score.

Behavior:

- Increases when enemies or machine guns are destroyed.

6

8. Player's Tank

Description: The player's controlled character, equipped with a cannon for firing bullets.

Behavior:

- Moves in four directions to navigate the maze.
- Fires bullets to destroy enemies and machine guns.

9. Loading Screen

Description: A visual element displayed during the game setup or between levels.

Behavior:

- Simulates loading progress with a percentage display and an animated "LOADING GAME" message.

4. Rules & Interactions

Rules:

1. **Player Movement**:
    - The player can move in four directions (up, down, left, right) using the arrow keys.
    - Movement is restricted by walls, obstacles, and the maze's boundaries.
2. **Shooting Mechanics**:
    - The player can shoot bullets using the space bar. o A maximum of 10 bullets can be active at any time.
    - Bullets disappear upon collision with walls, obstacles, or targets.
3. **Health System**:
    - The player starts with 200 health points.
    - Health decreases when hit by enemy bullets or colliding with enemies. o Collecting a health power-up restores health to full.
    - Enemies and machine guns also have health points, which decrease when hit by bullets.
4. **Enemy Behavior**:

- Enemies and machine guns move along predefined paths and fire bullets when the player is in range.
- Enemy bullets damage the player on collision.

5. **Scoring System**:
   - Destroying enemies and machine guns increases the player's score.
   - Machine guns and tanks yield bonus points when destroyed.
6. **Game Over Conditions**:
   - The game ends if the player's health drops to zero.
   - A "Game Over" screen displays the player's final score.
7. **Victory Condition**:
   - The player wins by destroying all enemies and machine guns, then reaching the maze's finish line.

Interactions:

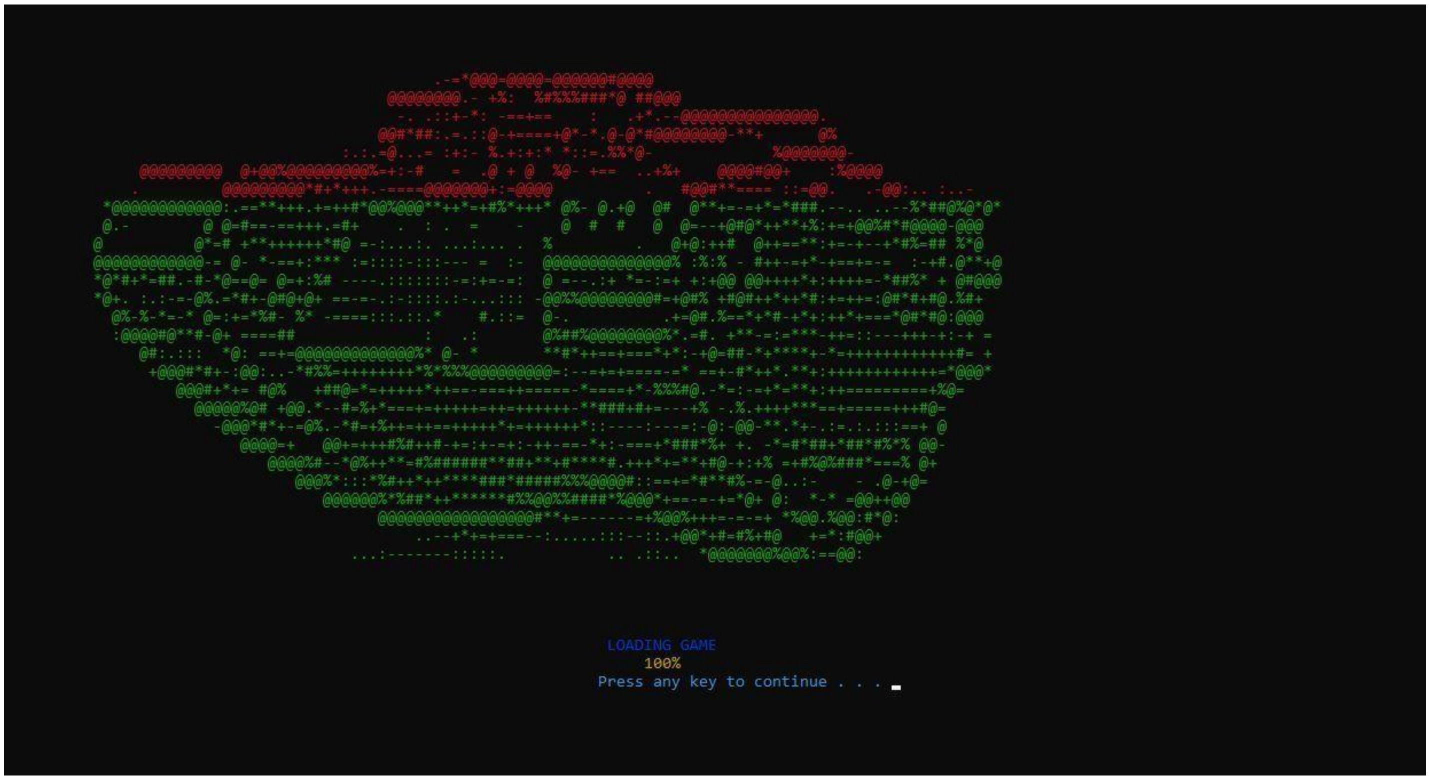
- **Player and Bullets**:
  - The player can fire bullets to attack enemies and destroy machine guns.
  - Player bullets are blocked by walls and obstacles.
- **Enemies and Player**:
  - Enemies move toward the player and fire bullets when in range.
  - Colliding with enemies reduces health for both the player and the enemy.
- **Machine Guns**: o Machine guns fire bullets at the player when aligned vertically or diagonally.
  - The player must destroy machine guns to proceed safely.
- **Health Power-Up (Pile)**:
  - The player restores health to full upon collecting a pile.
- **Maze and Walls**:
  - Walls block both player and enemy movement, acting as barriers.
  - Players can use walls for cover to avoid enemy bullets.

5. Goal of the Game

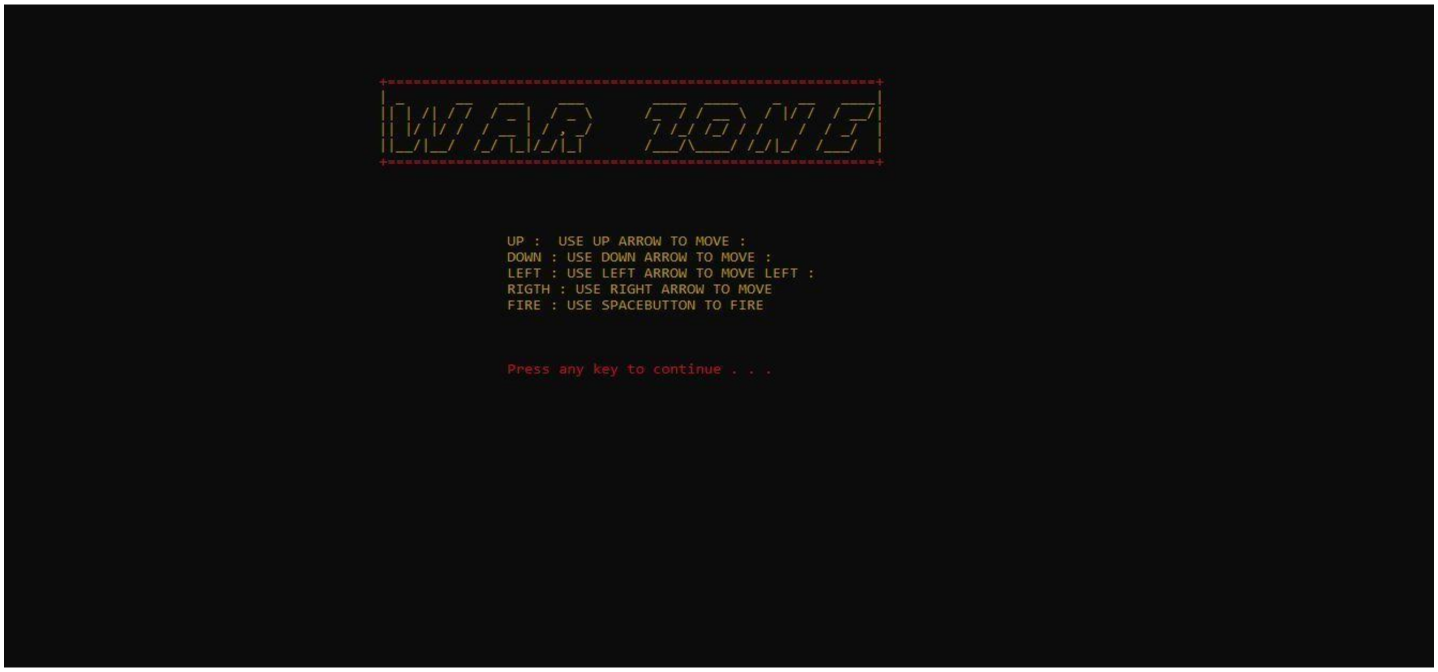
1. **Primary Objective**:
   - Survive the enemy attacks, eliminate all enemies and machine guns, and navigate through the maze to reach the finish line.
2. **Secondary Objective**:
   - Achieve the highest possible score by destroying enemies, machine guns, and collecting health power-ups strategically.
3. **Player Motivation**:
   - The player is tasked with neutralizing the enemy's forces and securing victory for their resistance group in a hostile and challenging battlefield. Success is achieved through tactical movement, accurate shooting, and effective resource management.

6. Wireframes

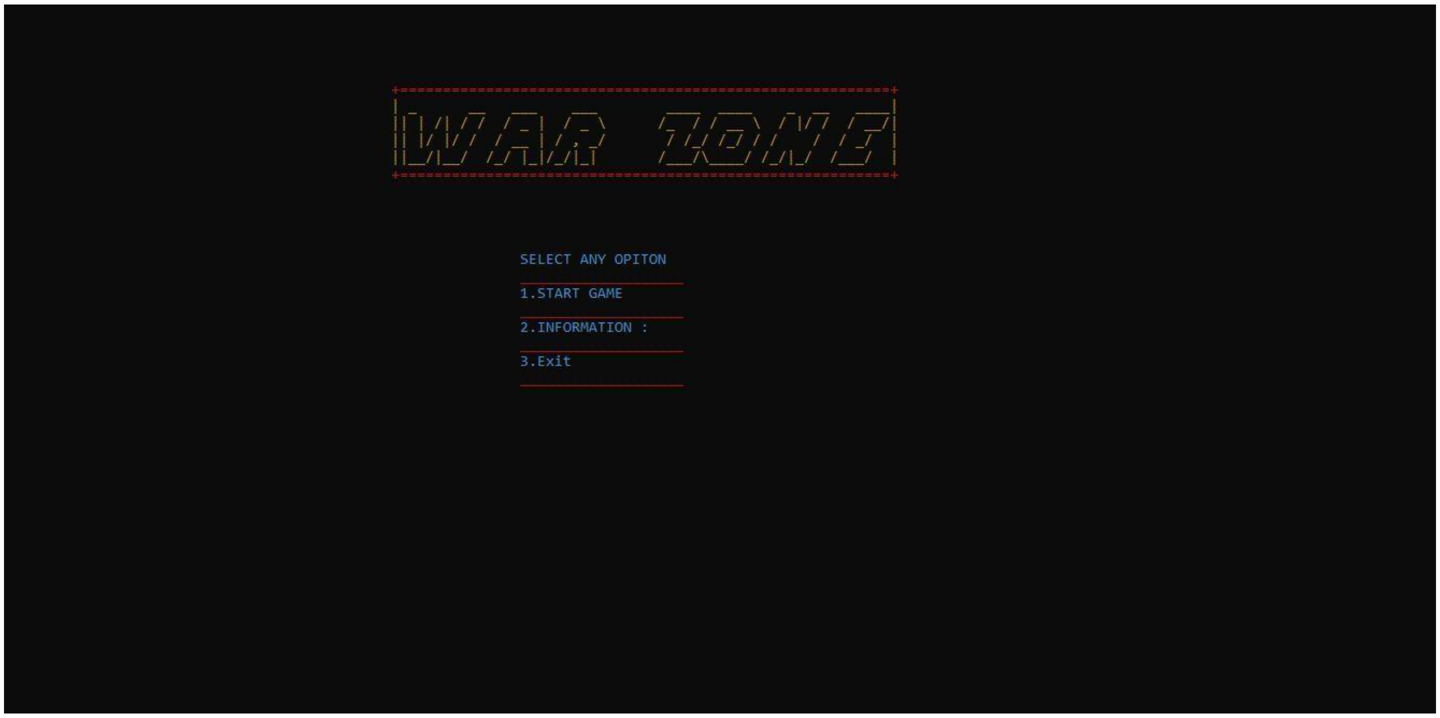
Following are the wireframes of WAR ZONE :



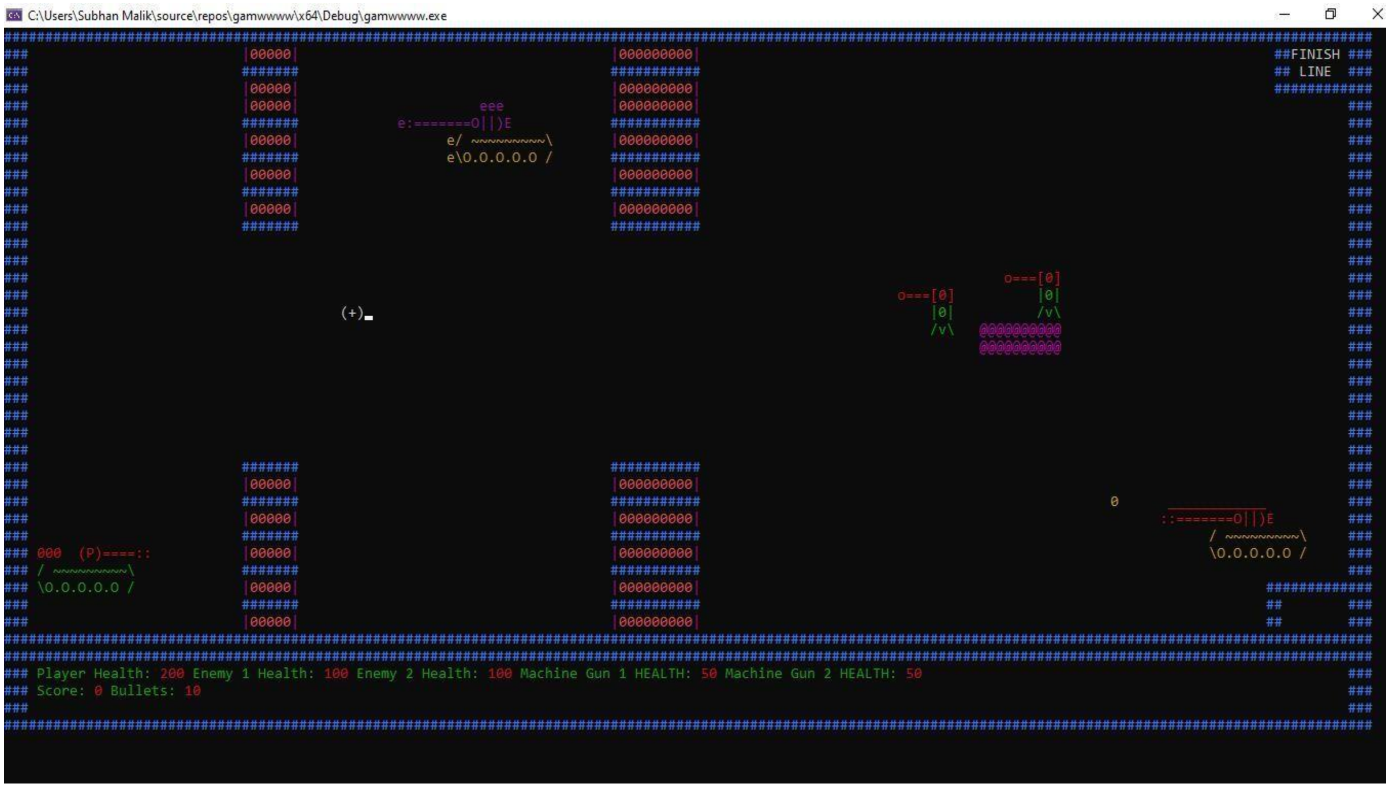
```
LOADING GAME
    100%
Press any key to continue . . .
```



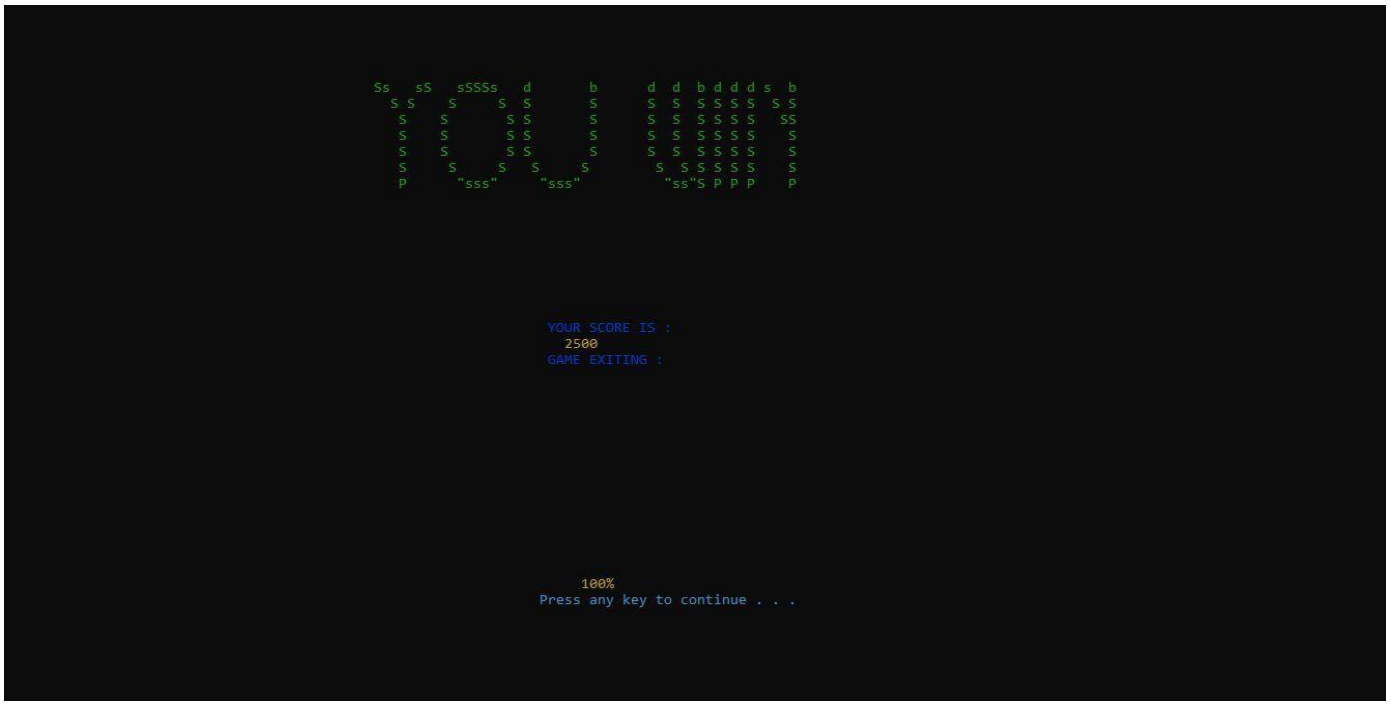
```
UP :  USE UP ARROW TO MOVE :
DOWN : USE DOWN ARROW TO MOVE :
LEFT : USE LEFT ARROW TO MOVE LEFT :
RIGTH : USE RIGHT ARROW TO MOVE
FIRE : USE SPACEBUTTON TO FIRE


Press any key to continue . . .
```



```
SELECT ANY OPITON
1.START GAME
2.INFORMATION :
3.Exit
```



```
C:\Users\Subhan Malik\source\repos\gamwwww\x64\Debug\gamwwww.exe
#######################################################################################################################################################################
###                          |00000|                                      |000000000|                                                                      ##FINISH ###
###                          #######                                      ###########                                                                      ## LINE  ###
###                          |00000|                                      |000000000|                                                                      ############
###                          |00000|                      eee             |000000000|                                                                               ###
###                          #######            e:=======O||)E            ###########                                                                               ###
###                          |00000|                  e/ ~~~~~~~~~\       |000000000|                                                                               ###
###                          #######                  e\O.O.O.O.O.O /     ###########                                                                               ###
###                          |00000|                                      |000000000|                                                                               ###
###                          #######                                      ###########                                                                               ###
###                          |00000|                                      |000000000|                                                                               ###
###                          #######                                      ###########                                                                               ###
###                                                                                                                                                                 ###
###                                                                                                                                                                 ###
###                                                                                                                       o===[0]                                   ###
###                                                                                                          o===[0]          |0|                                   ###
###                                      (+)                                                                     |0|          /v\                                   ###
###                                                                                                              /v\   @@@@@@@@@@                                   ###
###                                                                                                                    @@@@@@@@@@                                   ###
###                                                                                                                                                                 ###
###                                                                                                                                                                 ###
###                                                                                                                                                                 ###
###                                                                                                                                                                 ###
###                                                                                                                                                                 ###
###                                                                                                                                                                 ###
###                          #######                                      ###########                                                                               ###
###                          |00000|                                      |000000000|                                                                               ###
###                          #######                                      ###########                                                  0                            ###
###                          |00000|                                      |000000000|                                                        ::=======O||)E         ###
###                          #######                                      ###########                                                              / ~~~~~~~~~\     ###
### 000  (P)====::           |00000|                                      |000000000|                                                              \O.O.O.O.O.O /   ###
### / ~~~~~~~~~\             #######                                      ###########                                                                               ###
### \O.O.O.O.O.O /           |00000|                                      |000000000|                                                                     #############
###                          #######                                      ###########                                                                     ##        ###
###                          |00000|                                      |000000000|                                                                     ##        ###
#######################################################################################################################################################################
#######################################################################################################################################################################
### Player Health: 200 Enemy 1 Health: 100 Enemy 2 Health: 100 Machine Gun 1 HEALTH: 50 Machine Gun 2 HEALTH: 50                                                   ###
### Score: 0 Bullets: 10                                                                                                                                            ###
###                                                                                                                                                                 ###
#######################################################################################################################################################################
```



```
     Ss   sS   sSSSs   d       b      d  d  b d d d s  b
       S S    S     S  S       S      S  S  S S S S  S S
        S    S       S S       S      S  S  S S S S   SS
        S    S       S S       S      S  S  S S S S    S
        S    S       S S       S      S  S  S S S S    S
        S     S     S   S     S        S  S S S S S    S
        P      "sss"     "sss"          "ss"S P P P    P



                          YOUR SCORE IS :
                            2500
                          GAME EXITING :




                              100%
                         Press any key to continue . . .
```

8. Data Structure (2D Arrays)

```
const char maze[44][168] = {

    "#################################################################################
    ##################################################################################
    ############",

    "#################################################################################
    ##################################################################################
    ############",
        "###                |00000|                        |000000000|
    ##FINISH ###",
        "###                #######                         ###########
    ## LINE  ###",
        "###                |00000|                        |000000000|
    ############",
        "###                |00000|                        |000000000|
    ###",
        "###                #######                         ###########
    ###",
        "###                |00000|                        |000000000|
    ###",
        "###                #######                         ###########
    ###",
        "###                |00000|                        |000000000|
    ###",
        "###                #######                         ###########
    ###",
        "###                |00000|                        |000000000|
    ###",
        "###                #######                         ###########
    ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###
    @@@@@@@@@@                  ###",
        "###
    @@@@@@@@@@                  ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###
    ###",
        "###          #######           ###########
    ###",
        "###          |00000|           |000000000|
    ###",
        "###          #######           ###########
    ###",
        "###          |00000|           |000000000|
    ###",
        "###          #######           ###########
    ###",
        "###          |00000|           |000000000|
    ###",
        "###          #######           ###########
    ###",
        "###                          |00000|                                        |000000000|
############",

        "###                          #######                                          ###########
##        ###",

        "###                          |00000|                                        |000000000|
##        ###",

"######################################################################################
#######################################################################################
#############",

"######################################################################################
#######################################################################################
#############",

        "###
###",

        "###
###",

        "###
###",

"######################################################################################
#######################################################################################
############"

    };
```

9. Function Prototype

- char getCharAtxy(short int x, short int y);
- void gotoxy(int x, int y);
- // Main menu

- void menu();
- // Level 1 maze
- void printMaze1();
- void gameinformation();
- // Player actions
- void playererase();
- void player();
- void playermoveleft();
- void playermoveright();
- void playermoveup();
- void playermovedown();
- // Bullet mechanics
- void fireBullet();
- void moveBullets();
- // Health and power-up handling
- void generatePile();
- void printPile();
- void erasePile();
- void checkPileCollision();
- // Enemy 1 behavior
- void enemy1print();
- void enemy1erase();
- void enemy1movement(int playerX, int playerY);

- void direction1(int playerX, int playerY);
- void enemyFire(int enemyX, int enemyY, int playerY);
- void moveEnemyBullet();
- void checkEnemyBulletCollision();
- // Enemy 2 behavior
- void enemy2movement(int playerX, int playerY);
- void enemy2erase();
- void enemy2print();
- void enemy2direction(int playerX, int playerY);
- void enemy2Fire(int enemyX, int enemyY, int playerY);
- void moveEnemy2Bullet();
- void checkEnemy2BulletCollision();
- // Machine gun1 behavior
- void machineerase();
- void machinegunmovement(int playerX, int playerY);
- void machinedirection(int playerX, int playerY);
- void tankpicfor();
- void machinegun();
- void moveMachineGunBullet();
- void checkMachineGunBulletCollision();
- void machineGunFire(int gunX, int gunY, int playerY);
- //machine gun 2;
- int mx2 = 120, my2 = 27;
- string machineGun2Direction = "up-left";
- int machineGun2Health = 50;
- int machineGun2BulletX = -1, machineGun2BulletY = -1;
- bool machineGun2BulletActive = false;
- void machineGun2DirectionUpdate();
- void machinegun2movement(int playerX, int playerY);

- void machineGun2Fire(int gunX, int gunY, int playerY);
- void moveMachineGun2Bullet();
- void checkMachineGun2BulletCollision();
- void checkMachineGun2Collision();
- // Collision handling
- bool collision(int enemyX, int enemyY, int playerX, int playerY, int& enemyHealth, int& playerHealth);
- // Player position

10. Complete Code

```
#include <iostream>

#include <Windows.h>

#include <conio.h> using

namespace std;


// Utility functions for console handling char

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

gotoxy(int x, int y);

// Main menu void

menu(); // Level 1 maze

void printMaze1(); void

gameinformation();



// Player actions void

playererase(); void

player(); void

playermoveleft(); void

playermoveright(); void

playermoveup(); void

playermovedown(); //
```

```
Bullet mechanics void
fireBullet(); void
moveBullets(); //
Health and power-up
handling void
generatePile(); void
printPile(); void
erasePile(); void
checkPileCollision(); //
Enemy 1 behavior void
enemy1print(); void
enemy1erase();
void enemy1movement(int playerX, int playerY); void
direction1(int playerX, int playerY); void
enemyFire(int enemyX, int enemyY, int playerY);
void moveEnemyBullet(); void
checkEnemyBulletCollision(); // Enemy 2 behavior
void enemy2movement(int playerX, int playerY);
void enemy2erase(); void enemy2print();
void enemy2direction(int playerX, int playerY); void
enemy2Fire(int enemyX, int enemyY, int playerY);
void moveEnemy2Bullet(); void
checkEnemy2BulletCollision(); // Machine gun1
behavior void machineerase();
void machinegunmovement(int playerX, int playerY); void
machinedirection(int playerX, int playerY);
void tankpicfor(); void machinegun();
void moveMachineGunBullet(); void
checkMachineGunBulletCollision();
void machineGunFire(int gunX, int gunY, int playerY);
```

```
//machine gun 2; int mx2
= 120, my2 = 27;
string machineGun2Direction = "up-left"; int
machineGun2Health = 50;
int machineGun2BulletX = -1, machineGun2BulletY = -1;
bool machineGun2BulletActive = false; void
machineGun2DirectionUpdate();
void machinegun2movement(int playerX, int playerY);
void machineGun2Fire(int gunX, int gunY, int
playerY); void moveMachineGun2Bullet(); void
checkMachineGun2BulletCollision(); void
checkMachineGun2Collision();
// Collision handling
bool collision(int enemyX, int enemyY, int playerX, int playerY, int& enemyHealth, int&
playerHealth);
// Player position int
px = 4, py = 30;


// Enemy 1 data
int e1x = 48, e1y = 15; // Initial position
string direct1 = "down"; // Starting
direction int enemy1count = 1; // Enemy 1
bullet properties
int enemyBulletX = -1, enemyBulletY = -1; // Initial inactive position bool
enemyBulletActive = false;          // Status of the bullet
// Enemy 2 data
int e2x = 140, e2y = 5; // Initial position string
direct2 = "down"; // Initial direction // Enemy
2 bullet properties int enemy2BulletX = -1,
enemy2BulletY = -1; bool
enemy2BulletActive = false;
```

```
// Machine gun string machinegundirection = "left"; int
mx = 120, my = 15; int machinegunhealth = 50; //
Machine Gun bullet properties int machineGunBulletX
= -1, machineGunBulletY = -1; bool
machineGunBulletActive = false;
// Bullet data int
bulletX[10]; int
bulletY[10]; bool
bulletActive[10]; int
bulletLimit = 10;



int pileX, pileY; bool
pileActive = false;

// Health and score
int playerHealth = 200;    // Player's health int
enemy1Health = 100;     // Enemy 1 health int
enemy2Health = 100;     // Enemy 2 health int
score = 0;              // Player's score

//      Time      int
timeCounter = 0; int
screenWidth  =  150;
//  Screen  width  int
screenHeight  =  30;
// Screen height

// Loading scenario int
lx = 100, ly = 100;
```

```
//color code for maintence const string rest =
"\033[0m"; const string r = "\033[31m";//red
color cor const string g = "\033[32m";//green
color code; const string y = "\033[33m";//yello
const string m = "\033[35m";//mangent const
string c = "\033[36m";//cyan color code const
string b = "\033[34m";//blue const string w =
"\033[37m";//white color int main()
{


   system("cls");    tankpicfor();    cout
<< b;    gotoxy(lx = 66, ly = 35);    cout
<< "LOADING GAME" << endl;
   cout << y;    gotoxy(lx
= 70, ly = 35);
   for (int i = 0; i <= 100; i++)//for loading funciotn
   {        cout << "";
gotoxy(lx = 70, ly = 36);
cout << i << "%";

      cout << "";
      Sleep(30);
   }    cout << c;
gotoxy(lx = 65, ly = 37);
system("pause");
system("cls");    while
(true)
   {        system("cls");        menu();        cout
<< c;        gotoxy(lx = 60, ly = 15);        cout <<
```

```
"SELECT ANY OPITON" << endl;
gotoxy(lx = 60, ly = 16);        cout << r;
        cout << "____________________" << endl;
gotoxy(lx = 60, ly = 17);        cout << c;
        cout << "1.START GAME " << endl;
        cout << r;        gotoxy(lx = 60, ly = 18);
cout << "____________________" << endl;
gotoxy(lx = 60, ly = 19);        cout << c;
        cout << "2.INFORMATION :" <<
endl;        gotoxy(lx = 60, ly = 20);
cout << r;
        cout << "____________________" << endl;
gotoxy(lx = 60, ly = 21);

        cout << c;
        cout << "3.Exit " << endl;
gotoxy(lx = 60, ly = 22);        cout
<< r;
        cout << "____________________" << endl;
string opiton;        gotoxy(lx = 67, ly = 23);
cin >> opiton;        if (opiton == "1")
        {
            while (true)
            {
                // Level 1 functionality
system("cls");
printMaze1();                player();



                while (true)
                {
                    // Display stats
gotoxy(4, 38);
                    cout << g << "Player Health: " << r << playerHealth << " ";                        cout
<< g << "Enemy 1 Health: " << r << enemy1Health << " ";                        cout << g <<
"Enemy 2 Health: " << r << enemy2Health << " ";                        cout << g << "Machine Gun
1 HEALTH: " << r << machinegunhealth << " ";                        cout << g << "Machine Gun 2
HEALTH: " << r << machineGun2Health << " ";                        gotoxy(4, 39);
                    cout << g << "Score: " << r << score << " ";
cout << g << "Bullets: " << r << bulletLimit << " ";
if (playerHealth <= 0)

                    {
                        system("cls");


                        cout << r;
gotoxy(45, 5);
                        cout << "Ss   sS   sSSSs   d      b      d       sSSSs     sss. d sss       \n";
gotoxy(45, 6);
                        cout << "  S S    S     S  S      S      S       S     S  d      S           \n";
gotoxy(45, 7);
                        cout << "   S    S       S S      S      S       S Y      S           \n";
gotoxy(45, 8);
                        cout << "   S    S       S S      S      S       S   ss.  S sSSs       \n";
gotoxy(45, 9);
                        cout << "   S    S       S S      S      S       S      b S           \n";
gotoxy(45, 10);
                        cout << "   S     S     S   S     S      S       S     S      P S           \n";
gotoxy(45, 11);
                        cout << "   P      \"sss\"     \"sss\"       P sSSs   \"sss\"   ` ss'  P sSSss      \n";
gotoxy(45, 12);
```

```
            cout << "                                              \n";
gotoxy(45, 13);             cout << b;


            gotoxy(lx = 66, ly = 32);
            cout << "YOUR SCORE IS :  " << endl;
cout << y;
            gotoxy(lx = 68, ly = 33); cout
            << score;

            gotoxy(lx = 66, ly = 35);
            cout << b;
            cout << "GAME EXITING : " << endl;
cout << y;
            gotoxy(lx = 70, ly = 35);
            for (int i = 0; i <= 100; i++)//for loading funciotn
            {
              cout << "";
              gotoxy(lx = 70, ly = 36);
cout << i << "%";               cout
<< "";
              Sleep(30);
            }
cout << c;
            gotoxy(lx = 65, ly = 37);
system("pause");            break;
          }

          if (machinegunhealth <= 0 && enemy1Health <= 0 && enemy2Health <= 0)
          {
            system("cls");
cout << g;
gotoxy(45, 5);
            cout << "Ss   sS   sSSSs   d       b      d  d  b d d d s  b      \n";
gotoxy(45, 6);
            cout << "  S S    S     S  S       S      S  S  S S S S  S S      \n";
gotoxy(45, 7);
            cout << "   S    S       S S       S      S  S  S S S S   SS      \n"; gotoxy(45,
            8);

            cout << "   S    S       S S       S      S  S  S S S S    S      \n";
            gotoxy(45, 9);

            cout << "   S    S       S S       S      S  S  S S S S    S      \n";
gotoxy(45, 10);
            cout << "   S     S     S   S     S       S  S S S S S    S      \n";
gotoxy(45, 11);
            cout << "   P      \"sss\"     \"sss\"          \"ss\"S P P P    P      \n";
gotoxy(45, 12);
            cout << "                                                          \n";
cout << b;
            gotoxy(lx = 66, ly = 20);                 cout
<< "YOUR SCORE IS :  " << endl;                      cout
<< y;
            gotoxy(lx = 68, ly = 21);
cout << score;                    gotoxy(lx
= 66, ly = 22);                   cout <<
b;
            cout << "GAME EXITING : " << endl;
cout << y;
            gotoxy(lx = 70, ly = 35);
```

```
                        for (int i = 0; i <= 100; i++)//for loading funciotn
                        {                              cout <<
"";                               gotoxy(lx = 70, ly =
36);                               cout << i << "%";
cout << "";                               Sleep(30);
                        } cout <<
                        c;
                        gotoxy(lx
                        = 65, ly =
                        37);

                        system("pause");
exit(0);


                    }


                    // Player controls
                    if (GetAsyncKeyState(VK_UP)) playermoveup();
if (GetAsyncKeyState(VK_DOWN)) playermovedown();
if (GetAsyncKeyState(VK_LEFT)) playermoveleft();                        if
(GetAsyncKeyState(VK_RIGHT)) playermoveright();


                    if (_kbhit())
                    {
                        char key = _getch();
if (key == ' ') fireBullet();
                    }
                    if (timeCounter % 100 == 0 && !pileActive) { // Every 200 ticks
generatePile();
                    }
                    // Move bullets
moveBullets();

                    // Check collisions
checkPileCollision();
checkEnemyBulletCollision();
checkEnemy2BulletCollision();
checkMachineGunBulletCollision();
checkMachineGun2Collision();
                    // Move enemies
                    enemy1movement(px, py);
enemy2movement(px, py);
machinegunmovement(px, py);
machinegun2movement(px, py);
moveMachineGun2Bullet();
checkMachineGun2BulletCollision();

                    // Move enemy bullets
moveEnemyBullet();                    moveEnemy2Bullet();
moveMachineGunBullet();

                    // Print active pile
printPile();

                    // Increment time counter
timeCounter++;

                    Sleep(70); // Adjust frame rate
                }
            }
```

```
        }


        else if (opiton == "2")
        {           while (true)
{                 system("cls");
menu();
gameinformation();
cout << r;
gotoxy(lx = 60, ly = 23);
system("pause");
cout << rest;

break;
            }




        }
        else if (opiton == "3")
        {           exit(0);        }        else        {
cout << y;             gotoxy(lx = 60, ly = 23);
cout << "INVALID OPTION :" << endl;
            Sleep(1500);


        }
    }


}


void player() {     cout << r;
gotoxy(px, py + 1);     cout
```

```
<< "000  (P)====:: ";
gotoxy(px, py + 2);     cout
<< g << "/ ~~~~~~~~~\\  ";
gotoxy(px, py + 3);     cout
<< "\\O.O.O.O.O / ";     cout
<< rest;
}

// Erase the player's current position
void playererase() {     gotoxy(px,
py + 1);
    cout << "                ";
gotoxy(px, py + 2);     cout
<< "               ";
gotoxy(px, py + 3);     cout
<< "              ";
}

void playermoveleft() {     if (px > 3 &&
getCharAtxy(px - 2, py) == ' ' &&
getCharAtxy(px - 2, py + 1) == ' ' &&
getCharAtxy(px - 2, py + 2) == ' ' &&
getCharAtxy(px - 2, py + 3) == ' ') {
playererase();         px -= 3;         player();
    }
}

void playermoveright() {
    if (px < screenWidth &&
```

```
        getCharAtxy(px + 16, py) == ' ' &&
getCharAtxy(px + 16, py + 1) == ' ' &&
getCharAtxy(px + 16, py + 2) == ' ' &&
getCharAtxy(px + 16, py + 3) == ' ') {
playererase();         px += 3;         player();
    }
}

void playermoveup() {     if (py > 4 &&
getCharAtxy(px, py - 2) == ' ' &&
getCharAtxy(px + 5, py - 2) == ' ' &&
getCharAtxy(px + 10, py - 2) == ' ' &&
getCharAtxy(px + 15, py - 2) == ' ') {
playererase();         py -= 3;         player();
    }
}

void playermovedown() {     if (py <
screenHeight &&         getCharAtxy(px, py
+ 5) == ' ' &&         getCharAtxy(px + 5, py
+ 5) == ' ' &&         getCharAtxy(px + 10, py
+ 5) == ' ' &&         getCharAtxy(px + 15, py
+ 5) == ' ') {         playererase();

        py += 3;
player();
    }
}



void tankpicfor()
```

```
{


    const int consoleWidth = 100;
const int consoleHeight = 30;
const int textWidth = 88;
const int textHeight = 24;



    int startX = (consoleWidth - textWidth) / 2;
int startY = (consoleHeight - textHeight) / 2;




    cout << r;
    gotoxy(startX, startY + 1); cout << "                                    .-
=*@@@=@@@@=@@@@@@#@@@@                                    " << endl;
    gotoxy(startX, startY + 2); cout << "                              @@@@@@@@.- +%:  %#%%%###*@
##@@@                                  " << endl;
    gotoxy(startX, startY + 3); cout << "                               -. .::+-*: -==+==    :   .+*.--
@@@@@@@@@@@@@@@@.                      " << endl;
    gotoxy(startX, startY + 4); cout << "                              @@#*##:.=.::@-+====+@*-*.@-
@*#@@@@@@@@-**+      @%                      " << endl;
    gotoxy(startX, startY + 5); cout << "                          :.:.=@...= :+:- %.+:+:* *::=.%%*@-
%@@@@@@@-                     " << endl;
     gotoxy(startX, startY + 6); cout << "        @@@@@@@@@  @+@@%@@@@@@@@@@%=+:-
#   =  .@ + @  %@- +==  ..+%+    @@@@#@@+    :%@@@@                   " << endl;
gotoxy(startX, startY + 7); cout << "        .         @@@@@@@@@*#+*+++.-
====@@@@@@@+:=@@@@         .   #@@#**==== ::=@@.   .-@@:.. :..-          " << endl;
cout << g;
    gotoxy(startX, startY + 8); cout << "
*@@@@@@@@@@@@:.==**+++.+=++#*@@%@@@**++*=+#%*+++* @%- @.+@  @#
@**+=-=+*=*###.--.. ..--%*##@%@*@*        " << endl;
    gotoxy(startX, startY + 9); cout << "     @.-        @ @=#==-==+++.=#+    . : .  =    -    @  #  #   @
```

```
@=--+@#@*++**+%:+=+@@%#*#@@@@@-@@@         " << endl;
    gotoxy(startX, startY + 10); cout << "    @          @*=# +**+++++++*#@ =-:...:. ...:... .  %        .
@+@:++#  @++==**:+=-+--+*#%=## %*@          " << endl;
    gotoxy(startX, startY + 11); cout << "    @@@@@@@@@@@@@-= @- *-==+:*** :=::::-:::--- =  :-
@@@@@@@@@@@@@@@% :%:% - #++-=+*-+==+=-=  :-+#.@**+@        " << endl;
    gotoxy(startX, startY + 12); cout << "    *@*#+*=##.-#-*@==@= @=+:%# ----.:::::::-=:+=-=:  @ =--
.:+ *=-:=+ +:+@@ @@++++*+:++++=-*##%* + @#@@@        " << endl;
    gotoxy(startX, startY + 13); cout << "    *@+. :.:-=-@%.=*#+-@#@+@+ ==-=-.:-::::.:-...::: -
@@%%@@@@@@@@@#=+@#% +#@#++*++*#:+=++=:@#*#+#@.%#+          " << endl;
    gotoxy(startX, startY + 14); cout << "      @%-%-*=-* @=:+=*%#- %* -====:::.::.*    #.::=  @-.
.+=@#.%==*+*#-+*+:++*+===*@#*#@:@@@          " << endl;
    gotoxy(startX, startY + 15); cout << "      :@@@@#@**#-@+ ====##              :   .:
@%##%@@@@@@@@@%*.=#. +**-=:=***-++=::---+++-+:-+ =         " << endl;
    gotoxy(startX, startY + 16); cout << "        @#:.:::  *@: ==+=@@@@@@@@@@@@@@%* @- *
**#*++==+===*+*:-+@=##-*+****+-*=+++++++++++#= +         " << endl;
    gotoxy(startX, startY + 17); cout << "          +@@@#*#+-:@@:..-
*#%%=++++++++*%*%%%@@@@@@@@@@=:--=+=+====-=* ==+-
#*++*.**+:+++++++++++=*@@@*         " << endl;
    gotoxy(startX, startY + 18); cout << "            @@@#+*+= #@%
+##@=*=+++++*++=====++=====-*====+*-%%%#@.-*=:-=+*=**+:++=========+%@=
" << endl;
    gotoxy(startX, startY + 19); cout << "             @@@@@%@# +@@.*--
#=%+*===+=+++++=++=++++++-**###+#+=---+% -.%.++++***==+=====+++#@=              " <<
endl;
    gotoxy(startX, startY + 20); cout << "              -@@@*#*+-=@%.-
*#=+%++=++==+++++*+=++++++*::----:---=:-@:-@@-**.*+-.:=.:.:::==+ @              " << endl;
gotoxy(startX, startY + 21); cout << "                  @@@@=+   @@+=+++#%#++#-+=:+-=+:-++==-
*+:-===+*###*%+ +. -*=#*##+*##*#%*% @@-             " << endl;
    gotoxy(startX, startY + 22); cout << "                    @@@@%#--
*@%++**=#%######**##+**+#****#.+++*+=**+#@-+:+% =+#%@%###*===% @+               "
<< endl;
    gotoxy(startX, startY + 23); cout << "
@@@%*:::*%#++*++****###*#####%%%@@@@#::==+=*#**#%-=-@..:-    - .@-+@=              "
<< endl;
    gotoxy(startX, startY + 24); cout << "
@@@@@@%*%##*++******#%%@@%%####*%@@@*+==-=-+=*@+ @:  *-* =@@++@@
" << endl;
```

```
    gotoxy(startX, startY + 25); cout << "
@@@@@@@@@@@@@@@@@@@#**+=------=+%@@%+++=-=-=+ *%@@.%@@:#*@:
" << endl;

    gotoxy(startX, startY + 26); cout << "                                ..--+*+=+===--:.....:::--
::.+@@*+#=#%+#@   +=*:#@@+                  " << endl;

    gotoxy(startX, startY + 27); cout << "                          ...:-------:::::.         .. .::..
*@@@@@@@%@@%:==@@:                    " << endl;




}
void gotoxy(int x, int y)
{
    COORD coordinates;
coordinates.X     =     x;
coordinates.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinates);
}
void machinegun()
{
    cout << r;
gotoxy(mx, my);     cout
<< "o===[0]";     cout
<< g; gotoxy(mx, my +
1);

    cout << "    |0|";
gotoxy(mx, my + 2);
cout << "    /v\\";     cout
<< rest;
}
```

```
// Erase the machine gun
void machineerase() {
gotoxy(mx, my);     cout
<< "       ";
gotoxy(mx, my + 1);
cout << "       ";
gotoxy(mx, my + 2);
cout << "       ";
}


// Update the direction of the machine gun void
machinedirection(int playerX, int playerY)
{
    // Reverse direction if collision with the player
    if (collision(mx, my, playerX, playerY, machinegunhealth, playerHealth))
{         if (machinegundirection == "left") {             machinegundirection =
"right";
        }
else {
            machinegundirection = "left";
        }
    } else
    {

        // Reverse direction at boundaries         if (mx <=
115 && machinegundirection == "left") {
machinegundirection = "right";
        }
        if (mx >= 135 && machinegundirection == "right") {
machinegundirection = "left";
        }
    }
}

// Move the machine gun
void machinegunmovement(int playerX, int playerY)
{
    if (machinegunhealth <= 0)
    {
machineerase();
return;
    }

    machineerase();

    // Move based on direction     if
(machinegundirection == "left")
    {         mx
-= 2;
    }
    else if (machinegundirection == "right")
    {         mx
+= 2;
    }

    // Update direction
machinedirection(playerX, playerY);

    // Fire bullet if conditions are met
machineGunFire(mx, my, playerY);
```

```
    // Re-render the machine gun
machinegun(); } void
machinegun2()
{     cout << r;
gotoxy(mx2, my2);
cout << "o===[0]";
cout << g;
gotoxy(mx2, my2 + 1);
cout << "    |0|";
gotoxy(mx2, my2 + 2);
cout << "    /v\\";     cout
<< rest;
}

void machineerase2() {
gotoxy(mx2, my2);
    cout << "       ";
gotoxy(mx2, my2 + 1);     cout
<< "       ";     gotoxy(mx2,
my2 + 2);
    cout << "       ";
}
void machineGun2DirectionUpdate()
{
    // Reverse direction on vertical boundaries
    if (my2 <= 1 && (machineGun2Direction == "up-left" || machineGun2Direction == "up-right")) {
machineGun2Direction = (machineGun2Direction == "up-left") ? "down-left" : "down-right";
    }
    if (my2 >= screenHeight - 3 && (machineGun2Direction == "down-left" || machineGun2Direction ==
"down-right")) {
        machineGun2Direction = (machineGun2Direction == "down-left") ? "up-left" : "up-right";
    }
    // Reverse direction on horizontal boundaries
    if (mx2 <= 1 && (machineGun2Direction == "up-left" || machineGun2Direction == "down-left")) {
machineGun2Direction = (machineGun2Direction == "up-left") ? "up-right" : "down-right";
    }
    if (mx2 >= screenWidth - 8 && (machineGun2Direction == "up-right" || machineGun2Direction ==
"down-right")) {
        machineGun2Direction = (machineGun2Direction == "up-right") ? "up-left" : "down-left";
    }
}
void machinegun2movement(int playerX, int playerY) {
if (machineGun2Health <= 0) {         machineerase2();
return;
    }


    machineerase2();


    // Determine new position based on direction
int nextX = mx2;

    int nextY = my2;


    if (machineGun2Direction == "up-left") {
nextX--; nextY--;
    }
    else if (machineGun2Direction == "up-right") {
nextX++; nextY--;
    }
    else if (machineGun2Direction == "down-left") {
nextX--; nextY++;
    }
    else if (machineGun2Direction == "down-right") {
nextX++; nextY++;
    }

    // Check boundaries and obstacles
    if (nextX > 0 && nextX < screenWidth - 8 && // Horizontal boundaries
nextY > 0 && nextY < screenHeight - 3 && // Vertical boundaries
getCharAtxy(nextX, nextY) == ' ' && // Check top-left of machine gun
getCharAtxy(nextX, nextY + 1) == ' ' && // Check middle
getCharAtxy(nextX, nextY + 2) == ' ') { // Check bottom         mx2 =
nextX;         my2 = nextY;
    }
else {
        // Reverse direction if movement is blocked
        if (machineGun2Direction == "up-left") machineGun2Direction = "down-right";
else if (machineGun2Direction == "up-right") machineGun2Direction = "down-left";
else if (machineGun2Direction == "down-left") machineGun2Direction = "up-right";
else if (machineGun2Direction == "down-right") machineGun2Direction = "up-left";     }


    // Fire bullets and render Machine Gun 2
machineGun2Fire(mx2, my2, playerY);     machinegun2();
}

void machineGun2Fire(int gunX, int gunY, int playerY)
{
    if (!machineGun2BulletActive && abs(gunY - playerY) <= 3) { // Player is vertically aligned
machineGun2BulletX = gunX - 1;  // Start bullet just in front of Machine Gun 2
machineGun2BulletY = gunY + 1; // Center bullet vertically         machineGun2BulletActive =
true;
    }
```

```
}
void moveMachineGun2Bullet()
{
   if (machineGun2BulletActive) {
// Erase current bullet position
        gotoxy(machineGun2BulletX, machineGun2BulletY);
cout << " ";


        // Check for boundary or obstacle collisions
        char nextChar = getCharAtxy(machineGun2BulletX - 1, machineGun2BulletY);
if (machineGun2BulletX <= 1 || nextChar == '#' || nextChar == '|') {
machineGun2BulletActive = false; // Deactivate the bullet
        }
else {
            // Move bullet left
machineGun2BulletX--;             cout << r;
            gotoxy(machineGun2BulletX, machineGun2BulletY);
cout << "<"; // Bullet symbol             cout << rest;
        }
    }
}
void checkMachineGun2BulletCollision()
{
   if (machineGun2BulletActive &&
        px < machineGun2BulletX + 1 && px + 16 > machineGun2BulletX && // Horizontal collision
py <= machineGun2BulletY && py + 3 >= machineGun2BulletY) {   // Vertical collision
playerHealth -= 10; // Decrease player's health         machineGun2BulletActive = false; // Deactivate
the bullet         gotoxy(machineGun2BulletX, machineGun2BulletY);         cout << " "; // Erase the
bullet
    }
```

```
}
void checkMachineGun2Collision()
{
    for (int i = 0; i < bulletLimit; i++) {
if (bulletActive[i] &&
            bulletX[i] >= mx2 && bulletX[i] <= mx2 + 8 && // Horizontal collision
bulletY[i] >= my2 && bulletY[i] <= my2 + 3) { // Vertical collision
machineGun2Health -= 10; // Decrease Machine Gun 2's health
bulletActive[i] = false; // Deactivate the player's bullet
gotoxy(bulletX[i], bulletY[i]);             cout << " "; // Erase the bullet

            if (machineGun2Health <= 0) {
                score += 100;  // Increase player's score
machineerase2(); // Remove Machine Gun 2
            }
        }
    }
}




void enemy1print() {     cout <<
m;     gotoxy(e1x, e1y);     cout
<< "          eee";     gotoxy(e1x,
e1y + 1);     cout <<
"e:=======O||)E";
gotoxy(e1x, e1y + 2);     cout <<
y;     cout << "      e/
~~~~~~~~~\\";     gotoxy(e1x,
```

```
e1y + 3);     cout << "
e\\O.O.O.O.O / ";     cout << b;
}

void enemy1erase() {
gotoxy(e1x, e1y);
    cout << "                    ";
gotoxy(e1x, e1y + 1);
     cout << "                    "; gotoxy(e1x,
     e1y + 2);     cout << "                    ";
     gotoxy(e1x, e1y + 3);
    cout << "                    ";
}

void direction1(int playerX, int playerY) {     //
Reverse direction if collision with player occurs
    if (collision(e1x, e1y, playerX, playerY, enemy1Health, playerHealth))
{         if (direct1 == "up") {             direct1 = "down";
        }         else {
direct1 = "up";
        }
}     else
{
        // Regular boundary-based direction logic
if (e1y <= 3 && direct1 == "up") {
direct1 = "down";
        }
        if (e1y >= 30 && direct1 == "down") {
direct1 = "up";
        }
    }
}

void enemy1movement(int playerX, int playerY)
{ if (enemy1Health <= 0)
    {
        if (enemy1count != 0)
        {   // Enemy 1 defeated, erase completely
enemy1erase();              enemy1count = 0;
e1x = 0, e1y = 0;
        }

return;
    }

    enemy1erase();

    // Move the enemy based on direction
if (direct1 == "up")
    {         e1y
-= 2;
    }
    else if (direct1 == "down")
    {         e1y
+= 2;
    }

    // Update direction on collision or boundary
direction1(playerX, playerY);
```

```
    // Fire bullet if conditions are met
enemyFire(e1x, e1y, playerY);

    // Reprint the enemy
    enemy1print();
}

// Print Enemy 2 at its current position
void enemy2print() {     cout << r;
gotoxy(e2x, e2y);     cout << "
____________";     gotoxy(e2x, e2y +
1);     cout << " ::=======O||)E";
gotoxy(e2x, e2y + 2);     cout << y;
cout << "       / ~~~~~~~~~\\";
gotoxy(e2x, e2y + 3);     cout << "
\\O.O.O.O.O / ";     cout << rest;
}

// Erase Enemy 2 from its current position
void enemy2erase() {     gotoxy(e2x,
e2y);
    cout << "                    ";
gotoxy(e2x, e2y + 1);
    cout << "                    ";
gotoxy(e2x, e2y + 2);
    cout << "                    ";
gotoxy(e2x, e2y + 3);
    cout << "                    ";
}
```

```
// Move Enemy 2 and update its direction void enemy2movement(int playerX, int playerY)
{     if (enemy2Health <=
0)
    {
        enemy2erase();

return;
    }

    enemy2erase();

    // Move Enemy 2 based on direction
if (direct2 == "up")
    {
e2y--;
    }
    else if (direct2 == "down")
    {
e2y++;
    }

    // Update direction on collision or boundary
enemy2direction(playerX, playerY);

    // Fire bullet if conditions are met
enemy2Fire(e2x, e2y, playerY);

    // Reprint Enemy 2
enemy2print();
}
```

```
// Update Enemy 2's direction based on its position void
enemy2direction(int playerX, int playerY)
{
   // Reverse direction if collision with player occurs
   if (collision(e2x, e2y, playerX, playerY, enemy2Health, playerHealth))
{        if (direct2 == "up") {            direct2 = "down";
        }        else {
direct2 = "up";
        }
}    else
{
        // Regular boundary-based direction logic
if (e2y <= 5 && direct2 == "up") {
direct2 = "down";
        }
        if (e2y >= 28 && direct2 == "down") {
direct2 = "up";
        }
    }
}

void menu()
{



   int startX = 45;
int startY = 5;
cout << r;

   gotoxy(startX, startY); cout <<
```

```
"+==============================================================+" << endl;
cout << y;
    gotoxy(startX, startY + 1); cout << "| _      __   ___    ___        ____  ____    _  __   ____|" << endl;
gotoxy(startX, startY + 2); cout << "|| | /| / /  / _ |  / _ \\      /_  / / __ \\  / |/ /  / __/|" << endl;
gotoxy(startX, startY + 3); cout << "|| |/ |/ /  / __ | / , _/       / /_/ /_/ / /    /  / _/  |" << endl;
gotoxy(startX, startY + 4); cout << "||__/|__/  /_/ |_|/_/|_|       /___/\\____/ /_/|_/  /___/  |" << endl;     cout
<< r;
    gotoxy(startX, startY + 5); cout <<
"+==============================================================+" << endl;
cout << rest;
}
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 printMaze1()

{

    const char maze[44][168] = {


"############################################################################################
##############################################################################################
",


"############################################################################################
##############################################################################################
",
        "###                    |00000|                                |000000000|
##FINISH ###",

        "###                    #######                                ###########
## LINE  ###",

        "###                    |00000|                                |000000000|
############",

        "###                    |00000|                                |000000000|
###",

        "###                    #######                                ###########
###",

        "###                    |00000|                                |000000000|
###",

        "###                    #######                                ###########
###",

        "###                    |00000|                                |000000000|
###",

        "###                    #######                                ###########
###",

         "###                    |00000|                                |000000000|
###",

        "###                    #######                                ###########
```

```
###",
       "###
###",
       "###
###",
       "###
###",
       "###
###",
       "###
###",
       "###
@@@@@@@@@@                              ###",
       "###
@@@@@@@@@@                              ###",
       "###
###",
       "###
###",
       "###
###",
       "###
###",
       "###
###",
       "###
###",
       "###                  #######                      ###########
###",
       "###                  |00000|                      |000000000|
###",
       "###                  #######                      ###########
###",
        "###                 |00000|                      |000000000|
###",
       "###                  #######                      ###########
###",
        "###                          |00000|                                                |000000000|
###",
        "###                          #######                                                  ###########
###",
        "###                          |00000|                                                |000000000|
#############",
        "###                          #######                                                  ###########
##          ###",
        "###                          |00000|                                                |000000000|
##          ###",

"#########################################################################################
##########################################################################################
",

"#########################################################################################
##########################################################################################
",
        "###
###",
        "###
###",
        "###
###",

"#########################################################################################
##########################################################################################
"

    };


    for (int i = 0; i < 44; i++) { // Loop through rows
for (int j = 0; maze[i][j] != '\0'; j++) {

            // Add color for obstacles ('#', '|', '0', '@')
if (maze[i][j] == '#') {
                cout << "\033[1;34m" << maze[i][j] << "\033[0m";
            }
            else if (maze[i][j] == '|') {
                cout << "\033[1;35m" << maze[i][j] << "\033[0m";
            }
            else if (maze[i][j] == '0') {
                cout << "\033[1;31m" << maze[i][j] << "\033[0m";
            }
            else if (maze[i][j] == '@') {
                cout << "\033[1;35m" << maze[i][j] << "\033[0m";
            }
else {
                cout << maze[i][j];
            }
        }
        cout << endl;
    }
}

bool collision(int enemyX, int enemyY, int playerX, int playerY, int& enemyHealth, int& playerHealth)
{

    if (playerX < enemyX + 16 && playerX + 16 > enemyX &&
playerY < enemyY + 3 && playerY + 3 > enemyY)
    {
        // Reduce health of both player and enemy
playerHealth -= 10; // Reduce player health by 10
enemyHealth -= 10;  // Reduce enemy health by 10
return true;

    }    return
false;
}
```

```
void generatePile() {
    pileX = -1; // Initialize with an invalid position
pileY = -1;

    while (pileX < 0 || pileY < 0 || getCharAtxy(pileX, pileY) != ' ') {         pileX =
rand() % screenWidth; // Random X coordinate within screen width         pileY =
rand() % screenHeight; // Random Y coordinate within screen height
    }

    pileActive = true; // Activate the pile
}

void printPile()
{
    if (pileActive)
    {
        gotoxy(pileX, pileY);
        cout << "(+)"; // Represent the pile with a heart
    }
}

void erasePile()
{
    gotoxy(pileX, pileY);     cout
<< "   "; // Erase the pile
pileActive = false;

}
void fireBullet() {
    for (int i = 0; i < bulletLimit; i++) {
        if (!bulletActive[i]) {  // Check if there's space for a new bullet
bulletX[i] = px + 16; // Start the bullet at the player's right edge
bulletY[i] = py + 1;  // Center the bullet vertically             bulletActive[i]
= true;
            break; // Only fire one bullet at a time
        }
    }
}

void moveBullets() {
    for (int i = 0; i < bulletLimit; i++) {
if (bulletActive[i]) {
            // Erase the bullet's previous position
gotoxy(bulletX[i], bulletY[i]);             cout
<< " ";

            // Move the bullet to the right
bulletX[i] += 2;

            // Check if the bullet has gone off the screen (right side)
if (bulletX[i] > screenWidth) {
                bulletActive[i] = false; // Deactivate the bullet if it's out of bounds
continue;
            }

            // Check if the bullet hits an obstacle             char
nextChar = getCharAtxy(bulletX[i], bulletY[i]);             if
(nextChar == '#' || nextChar == '|' || nextChar == '@' ||
nextChar == '0') {                 bulletActive[i] = false; //
```

```
Deactivate the bullet when it hits an obstacle
continue;
            }

            // Reprint the bullet at its new position
gotoxy(bulletX[i], bulletY[i]);             cout
<< "O";

            // Check for collisions with Enemy 1
            if (bulletX[i] >= e1x && bulletX[i] <= e1x + 16 &&
bulletY[i] >= e1y && bulletY[i] <= e1y + 3) {
enemy1Health -= 10; // Decrease Enemy 1's health
bulletActive[i] = false; // Deactivate the bullet                 if
(enemy1Health <= 0) {                    score += 100;  //
Increase the score                    if (enemy1count != 0)
                    {   // Enemy 1 defeated, erase completely
enemy1erase();                        enemy1count = 0;
e1x = 0, e1y = 0;
                    }
}
continue;
            }

            // Check for collisions with Enemy 2
            if (bulletX[i] >= e2x && bulletX[i] <= e2x + 16 &&
bulletY[i] >= e2y && bulletY[i] <= e2y + 3) {
enemy2Health -= 10; // Decrease Enemy 2's health
bulletActive[i] = false; // Deactivate the bullet                 if
(enemy2Health <= 0) {                    score += 100;  // Increase the
score                    enemy2erase(); // Erase Enemy 2 completely
```

```
                }
continue;
            }

            // Check for collisions with Machine Gun 1             if
(bulletX[i] >= mx && bulletX[i] <= mx + 8 &&                 bulletY[i]
>= my && bulletY[i] <= my + 3) {                 machinegunhealth -= 10;
// Decrease Machine Gun 1's health                 bulletActive[i] = false; //
Deactivate the bullet                 if (machinegunhealth <= 0) {
score += 100;  // Increase the score                     machineerase(); //
Erase Machine Gun 1 if dead
                }
continue;
            }

            // Check for collisions with Machine Gun 2             if (bulletX[i]
>= mx2 && bulletX[i] <= mx2 + 8 &&                 bulletY[i] >= my2 &&
bulletY[i] <= my2 + 3) {                 machineGun2Health -= 10; //
Decrease Machine Gun 2's health                 bulletActive[i] = false; //
Deactivate the bullet                 if (machineGun2Health <= 0) {
score += 100;  // Increase the score                     machineerase2(); // Erase
Machine Gun 2 if dead
                }
            }
        }
    }
}

void checkPileCollision()
{
    if (pileActive &&
        px < pileX + 1 && px + 16 > pileX && // Horizontal collision
py < pileY + 1 && py + 3 > pileY) { // Vertical collision
playerHealth = 100; // Restore health         erasePile();
    }
}
void gameinformation()
{     cout << y;     gotoxy(lx
= 60, ly = 15);
    cout << "UP :  USE UP ARROW TO MOVE :" << endl;
gotoxy(lx = 60, ly = 16);
    cout << "DOWN : USE DOWN ARROW TO MOVE :" << endl;
    gotoxy(lx = 60, ly = 17);
    cout << "LEFT : USE LEFT ARROW TO MOVE LEFT :" << endl;
gotoxy(lx = 60, ly = 18);
    cout << "RIGTH : USE RIGHT ARROW TO MOVE " << endl;
    gotoxy(lx = 60, ly = 19);
    cout << "FIRE : USE SPACEBUTTON TO FIRE " << endl;
cout << rest;
}
void enemyFire(int enemyX, int enemyY, int playerY)
{
    // Fire if the player is on the same horizontal level and no bullet is active
if (!enemyBulletActive && abs(enemyY - playerY) <= 3)
    {
        enemyBulletX = enemyX - 1;  // Start bullet just in front of the enemy
enemyBulletY = enemyY + 1; // Align bullet vertically with enemy
enemyBulletActive = true;
    }
}
```

```
void moveEnemyBullet() {     if
(enemyBulletActive) {         // Erase the
current bullet         gotoxy(enemyBulletX,
enemyBulletY);         cout << " ";

        // Check if the bullet will hit an obstacle or boundary         char
nextChar = getCharAtxy(enemyBulletX - 2, enemyBulletY);         if
(enemyBulletX <= 1 || nextChar == '#' || nextChar == '|') {
enemyBulletActive = false; // Deactivate the bullet
        }
else {
            // Move the bullet to the left
enemyBulletX -= 2;             // Print
the bullet             cout << y;
            gotoxy(enemyBulletX, enemyBulletY);
cout << "0"; // Bullet symbol             cout <<
rest;
        }
    }
}
void checkEnemyBulletCollision()
{
    if (enemyBulletActive &&
        px < enemyBulletX + 1 && px + 16 > enemyBulletX && // Horizontal collision
py <= enemyBulletY && py + 3 >= enemyBulletY)     // Vertical collision
    {
        // Bullet hits the player
        playerHealth -= 10;        // Decrease player health
enemyBulletActive = false; // Deactivate the bullet
gotoxy(enemyBulletX, enemyBulletY);
```

```
        cout << " ";                // Erase the bullet
    }
}
void enemy2Fire(int enemyX, int enemyY, int playerY)
{
    if (!enemy2BulletActive && abs(enemyY - playerY) <= 3)
    {
        enemy2BulletX = enemyX - 1;  // Start bullet just in front of Enemy 2
enemy2BulletY = enemyY + 1; // Align bullet vertically with Enemy 2
enemy2BulletActive = true;
    }
}
void machineGunFire(int gunX, int gunY, int playerY)
{
    if (!machineGunBulletActive && abs(gunY - playerY) <= 3)
    {
        machineGunBulletX = gunX - 1;  // Start bullet just in front of the machine gun
machineGunBulletY = gunY + 1; // Align bullet vertically with the machine gun
machineGunBulletActive = true;
    }
}
void moveEnemy2Bullet() {
if (enemy2BulletActive) {
// Erase the current bullet
        gotoxy(enemy2BulletX, enemy2BulletY);
cout << " ";

        // Check if the bullet will hit an obstacle or boundary         char nextChar =
getCharAtxy(enemy2BulletX - 2, enemy2BulletY);         if (enemy2BulletX <= 1 ||
nextChar == '#' || nextChar == '|' || nextChar == '@') {              enemy2BulletActive =
false; // Deactivate the bullet
        }
else {
            // Move the bullet to the left
enemy2BulletX -= 2;             //
Print the bullet             cout << y;
            gotoxy(enemy2BulletX, enemy2BulletY);
cout << "0"; // Bullet symbol             cout <<
rest;
        }
    }
}

void moveMachineGunBullet() {
if (machineGunBulletActive) {
        // Erase the current bullet
        gotoxy(machineGunBulletX, machineGunBulletY);
cout << " ";

        // Check if the bullet will hit an obstacle or boundary
        char nextChar = getCharAtxy(machineGunBulletX - 2, machineGunBulletY);
if (machineGunBulletX <= 1 || nextChar == '#' || nextChar == '|' || nextChar == '@')
        {
            machineGunBulletActive = false; // Deactivate the bullet
        }
else {
            // Move the bullet to the left
machineGunBulletX-=2;
            // Print the bullet
cout << r;
```

```
            gotoxy(machineGunBulletX, machineGunBulletY);
cout << "="; // Bullet symbol             cout << rest;
        }
    }
}
void checkEnemy2BulletCollision()
{
    if (enemy2BulletActive &&
        px < enemy2BulletX + 1 && px + 16 > enemy2BulletX && // Horizontal collision
py <= enemy2BulletY && py + 3 >= enemy2BulletY)      // Vertical collision
    {
        playerHealth -= 10;        // Decrease player health
enemy2BulletActive = false; // Deactivate the bullet
gotoxy(enemy2BulletX, enemy2BulletY);
        cout << " ";               // Erase the bullet
    }
}
void checkMachineGunBulletCollision()
{
    if (machineGunBulletActive &&
        px < machineGunBulletX + 1 && px + 16 > machineGunBulletX && // Horizontal collision
py <= machineGunBulletY && py + 3 >= machineGunBulletY)      // Vertical collision
    {
        playerHealth -= 10;            // Decrease player health
machineGunBulletActive = false; // Deactivate the bullet
gotoxy(machineGunBulletX, machineGunBulletY);
        cout << " ";                   // Erase the bullet
    }
}
```

11. Weakness in the Game:

- There is a little bit blinking in my game.
- The game is not very complex.
- There are less levels in my game.

12. Future Directions:

- I will try to add more levels in my game.
- I will add graphics in my game to make it more attractive