**Truck Fight Game**

****

Session: 2022-2026

**Submitted by:**

Rida Batool

2022-CS-118

**Supervised by:**

Sir Irzam

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Contents**

[**Short Description and Story Writing of Game** 3](#_Toc130943580)

[**Game Characters Description** 3](#_Toc130943581)

[**Player:** 3](#_Toc130943582)

[**Spike:** 3](#_Toc130943583)

[**Enemies:** 3](#_Toc130943584)

[**Crusher:** 3](#_Toc130943585)

[**Steel Thunder:** 3](#_Toc130943586)

[**Blaze:** 4](#_Toc130943587)

[**Game object description** 4](#_Toc130943588)

[**Walls:** 4](#_Toc130943589)

[**Rules and Interaction:** 4](#_Toc130943590)

[**Goal of the game:** 4](#_Toc130943591)

[**Wireframes of the game:** 5](#_Toc130943592)

[**Data Structures:** 6](#_Toc130943593)

[**Function Prototypes:** 6](#_Toc130943594)

[**Complete Code:** 8](#_Toc130943595)

# **Short Description and Story Writing of Game**

Truck Fight is a console-based action-packed single-player game where players control the powerful truck, Spike, equipped with only bullets. Navigate through challenging tracks and face off against three enemy trucks: Crusher, Steel Thunder, and Blaze, in fast-paced battles. Use strategy and quick reflexes to defeat your opponents and become the ultimate truck-fighting champion!

With smooth controls, and pulse-pounding action, Truck Fight is the perfect game for console gamers who love excitement and intense single-player challenges. Put your skills to the test as you face off against three formidable enemy trucks, relying only on your trusty bullets to come out on top. Can you emerge victorious against Crusher, Steel Thunder, and Blaze in this high-stakes, bullet-riddled battle? Find out in Truck Fight!

# **Game Characters Description**

## **Player:**

### **Spike:**

Spike is the main player truck in the console-based game Truck Fight. It is equipped with only bullets, having a total of three lives and controlled by the player to defeat three enemy trucks and become the champion. Spike is the key to victory, providing an intense and exciting gaming experience.

## **Enemies:**

There are three enemies in game.

### **Crusher:**

Crusher is one of the three tricky enemy trucks in the console-based game Truck Fight. With unpredictable and random movements, this adversary keeps the player Spike on their toes. Spike must use their driving skills and weapons to anticipate Crusher's unpredictable actions and defeat this opponent to claim victory in this battle.

### **Steel Thunder:**

Steel Thunder is one of the three enemy trucks in the game Truck Fight. Appearing through different gates, Steel Thunder presents a challenge to the player Spike, who has only a set of bullets as their weapon. Spike must use its aiming skills to defeat Steel Thunder and emerge victorious in this battle.

### **Blaze:**

Blaze is one of the three enemy trucks in the game Truck Fight. It is known for its firing techniques. This makes Blaze a tricky opponent for the player's truck, Spike, and the player must use strategy and quick reflexes to defeat it. Blaze's making spike target with bullets adds an extra layer of challenge to the game, making it a must-defeat opponent for those seeking to become the ultimate truck fighting champion.

# **Game object description**

Following are objects in the game:

### **Walls:**

Walls are the barriers in the game which the Spike and the enemy trucks cannot cross.

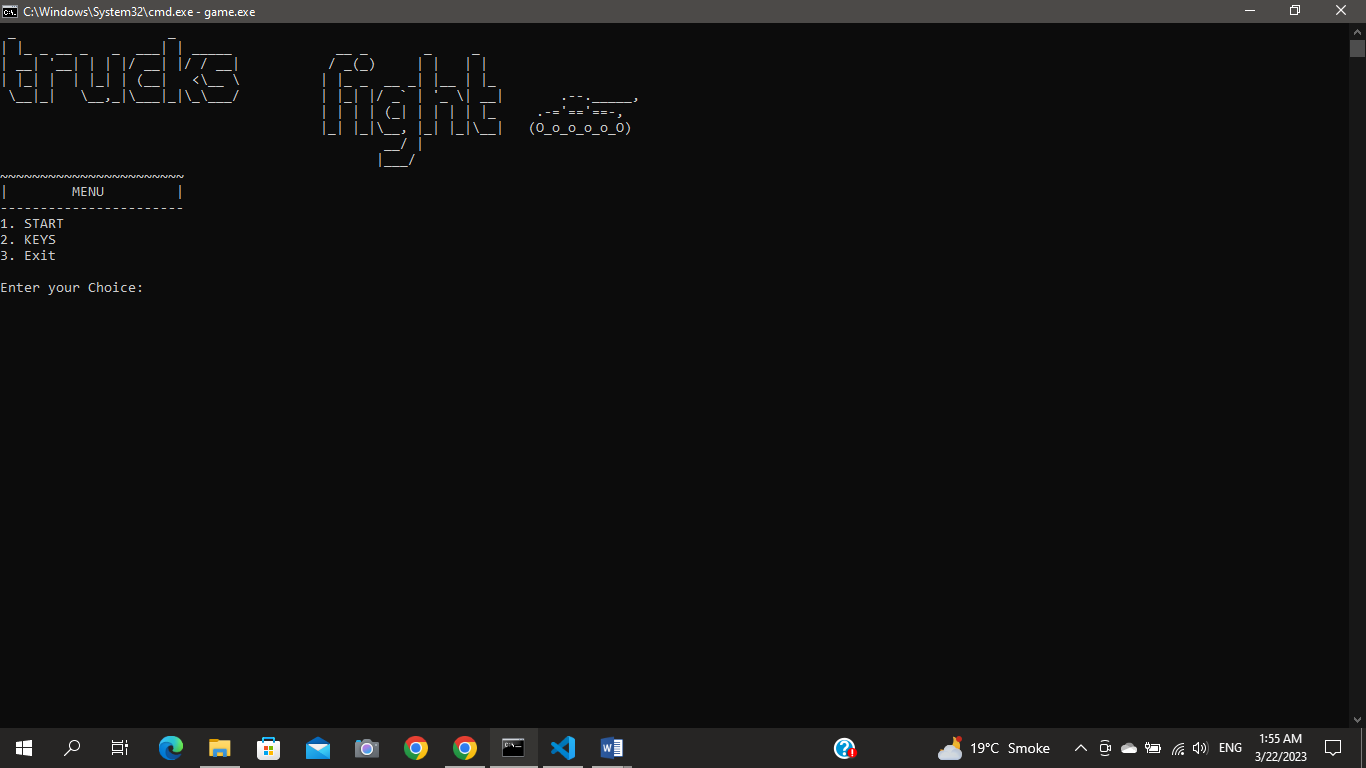
# **Rules and Interaction:**

Spike can eat food palette that have been put across the maze. The gameplay involves using Spike's bullets to defeat enemies, collecting Food Palette to increase his health, and avoiding enemy attacks. Spike loses a life if he collides with any of the bullets coming from blaze. If spike eats Power Palettes, then there is increase in its number of bullets. Health increases when the spike eats food palettes.

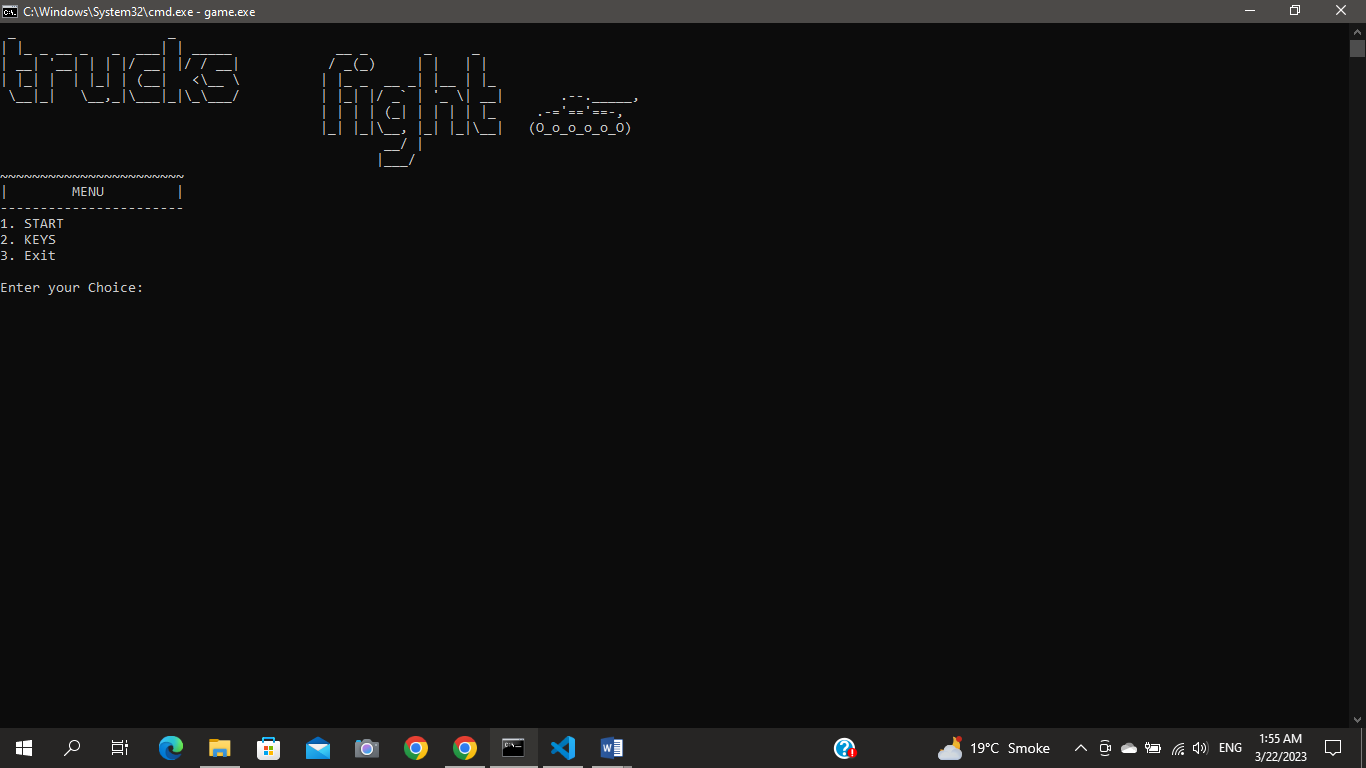
# **Goal of the game:**

The goal of the game Truck Fight is for the player to defeat all three enemy trucks, Crusher, Steel Thunder, and Blaze, and become the champion truck fighter. The player must use Spike's bullets to defeat the enemies, collect Food Palette to increase their health, and avoid enemy attacks. The game's difficulty increases as the player progress and the player's score are tracked throughout the game. The ultimate objective is to destroy all the three enemy trucks and become the ultimate truck fighting champion.

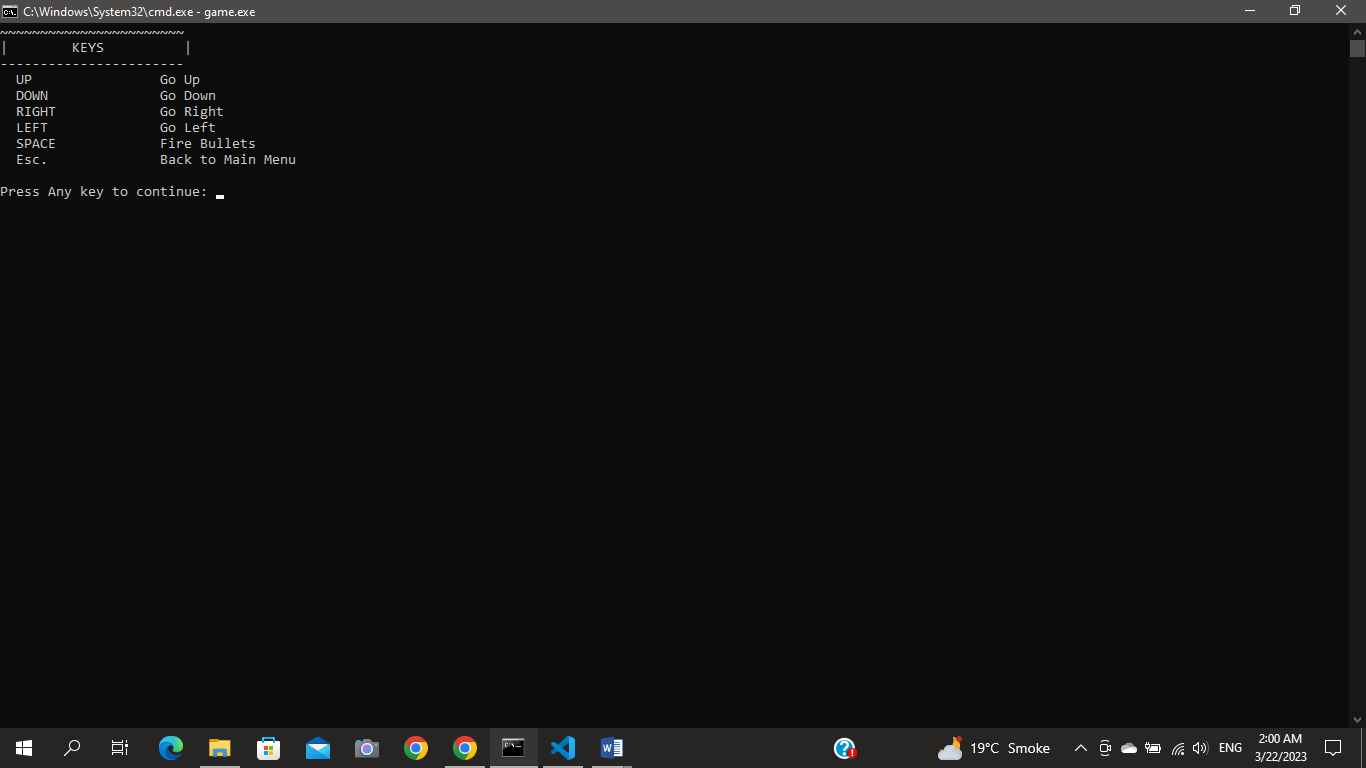
# **Wireframes of the game:**

****

**Figure 1: WelcomeScreen**



**Figure 2: MENU**

****

**Figure 3: KEYS**

# **Data Structures:**

The 2D arrays used are:

* char spike[3][11]
* char steelThunder[3][7]
* char crusher[3][9]
* char blaze[3][9]

The variable used are:

* int spikeX = 18;
* int spikeY = 4;
* int steelThunderX = 4;
* int steelThunderY = 4;
* int crusherX = 60;
* int crusherY = 12;
* int blazeX = 15;
* int blazeY = 24;
* string steelThunderDirection = "Down";
* string crusherDirection = "Down";
* string blazeDirection = "Right";
* int bulletX[1000];
* int bulletY[1000];
* int bulletCount = 0;
* int blazeBulletX[1000];
* int blazeBulletY[1000];
* int blazeBulletCount = 0;
* int crusherBulletX[1000];
* int crusherBulletY[1000];
* int crusherBulletCount = 0;
* int timer1 = 0;
* int timer2 = 0;
* int timer3 = 0;
* char check = 'Y';
* int score = 0;
* int crusher\_Health = 7;
* int steelThunder\_Health = 5;
* int blaze\_Health = 10;
* int spike\_Health = 10;
* int spikeLive = 3;

# **Function Prototypes:**

* void welcomeScreen();
* void menu();
* void keys();
* void showMaze();
* void showSpike();
* void eraseSpike();
* void moveSpikeLeft();
* void moveSpikeRight();
* void moveSpikeUp();
* void moveSpikeDown();
* void showBlaze();
* void eraseBlaze();
* void moveBlaze();
* void showSteelThunder();
* void eraseSteelThunder();
* void moveSteelThunder();
* void showCrusher();
* void eraseCrusher();
* void moveCrusher();
* void generateBullet();
* void moveBullet();
* void displayBullet(int x, int y);
* void displayBullet2(int x, int y);
* void eraseBullet(int x, int y);
* void removeBulletFromArray(int idx);
* void generateBlazeBullet();
* void moveBlazeBullet();
* void removeBlazeBulletFromArray(int idx);
* void eraseBlazeBullet();
* void generateCrusherBullet();
* void moveCrusherBulletLeft();
* void moveCrusherBulletRight();
* void removeCrusherBulletFromArray(int idx);
* void eraseCrusherBullet();
* void bulletCollisionWithBlaze();
* void bulletCollisionWithSteelThunder();
* void bulletCollisionWithCrusher();
* void crusherCollisionWithSpike();
* void steelThunderCollisionWithSpike();
* void blazeCollisionWithSpike();
* void blazeBulletCollisionWithSpike();
* void crusherBulletCollisionWithSpike();
* void addScore();
* void crusherHealth();
* void steelThunderHealth();
* void blazeHealth();
* void spikeHealth();
* void displayScoreAndHealth();
* void winning();
* void losing();
* void clearScreen();
* void gotoxy(int x, int y);
* char getCharAtxy(short int x, short int y);

# **Complete Code:**

#include <iostream>

#include <conio.h>

#include <windows.h>

using namespace std;

int spikeX = 18;

int spikeY = 4;

char spike[3][11] = {

{' ', ' ', ' ', '.', '-', '`', '-', '.', ' ', ' ', ' '},

{'.', '-', '=', '`', '=', ' ', '=', '`', '=', '-', '.'},

{'(', 'O', '\_', 'o', '\_', '-', '\_', 'o', '\_', 'O', ')'}};

int steelThunderX = 4;

int steelThunderY = 4;

char steelThunder[3][7] = {

{' ', '\_', '[', '`', ']', '\_', ' '},

{'|', '\_', '\_', '\_', '\_', '\_', '|'},

{'(', 'O', '\_', 'o', '\_', 'O', ')'}};

string steelThunderDirection = "Down";

int crusherX = 60;

int crusherY = 12;

char crusher[3][9] = {

{'\_', '\_', '/', '"', '"', '"', '"', '\\', '\_'},

{'/', '"', '"', '"', '"', '"', '"', '"', '\\'},

{'\\', '\_', '@', '\_', '@', '\_', '@', '\_', '/'}};

string crusherDirection = "Down";

int blazeX = 15;

int blazeY = 24;

char blaze[3][9] = {

{' ', '\_', '(', '`', '`', ')', '\_', '\_', ' '},

{'/', '~', '~', '~', '~', '~', '~', '~', '\\'},

{'\\', 'O', '.', 'O', '.', 'O', '.', 'O', '/'}};

string blazeDirection = "Right";

int bulletX[1000];

int bulletY[1000];

int bulletCount = 0;

int timer1 = 0;

int timer2 = 0;

int timer3 = 0;

char check = 'Y';

int score = 0;

int crusher\_Health = 7;

int steelThunder\_Health = 5;

int blaze\_Health = 10;

int spike\_Health = 10;

int spikeLive = 3;

int blazeBulletX[1000];

int blazeBulletY[1000];

int blazeBulletCount = 0;

int crusherBulletX[1000];

int crusherBulletY[1000];

int crusherBulletCount = 0;

void welcomeScreen();

void menu();

void keys();

void showMaze();

void showSpike();

void eraseSpike();

void moveSpikeLeft();

void moveSpikeRight();

void moveSpikeUp();

void moveSpikeDown();

void showBlaze();

void eraseBlaze();

void moveBlaze();

void showSteelThunder();

void eraseSteelThunder();

void moveSteelThunder();

void showCrusher();

void eraseCrusher();

void moveCrusher();

void generateBullet();

void moveBullet();

void displayBullet(int x, int y);

void displayBullet2(int x, int y);

void eraseBullet(int x, int y);

void removeBulletFromArray(int idx);

void generateBlazeBullet();

void moveBlazeBullet();

void removeBlazeBulletFromArray(int idx);

void eraseBlazeBullet();

void generateCrusherBullet();

void moveCrusherBulletLeft();

void moveCrusherBulletRight();

void removeCrusherBulletFromArray(int idx);

void eraseCrusherBullet();

void bulletCollisionWithBlaze();

void bulletCollisionWithSteelThunder();

void bulletCollisionWithCrusher();

void crusherCollisionWithSpike();

void steelThunderCollisionWithSpike();

void blazeCollisionWithSpike();

void blazeBulletCollisionWithSpike();

void crusherBulletCollisionWithSpike();

void addScore();

void crusherHealth();

void steelThunderHealth();

void blazeHealth();

void spikeHealth();

void displayScoreAndHealth();

void winning();

void losing();

void clearScreen();

void gotoxy(int x, int y);

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

main()

{

while (check == 'Y')

{

system("cls");

welcomeScreen();

menu();

}

}

void welcomeScreen()

{

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

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

cout << "| \_\_| '\_\_| | | |/ \_\_| |/ / \_\_| / \_(\_) | | | | " << endl;

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

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

cout << " | | | | (\_| | | | | |\_ .-='=='==-, " "<< endl;

cout << " |\_| |\_|\\\_\_, |\_| |\_|\\\_\_| (O\_o\_o\_o\_o\_O) " << endl;

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

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

}

void winning()

{

cout << endl

<< endl

<< endl

<< endl

<< endl

<< endl;

cout << " ~~~~~~~~~~~~~~~~~~~~" << endl;

cout << " | YOU WON!! |" << endl;

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

cout << endl

<< endl;

}

void losing()

{

cout << endl

<< endl

<< endl

<< endl

<< endl

<< endl;

cout << " ~~~~~~~~~~~~~~~~~~~~ " << endl;

cout << " | YOU LOSE!! |" << endl;

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

cout << endl

<< endl;

}

void menu()

{

int option;

cout << "~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| MENU |" << endl;

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

cout << "1. START" << endl;

cout << "2. KEYS" << endl;

cout << "3. Exit" << endl;

cout << endl;

cout << "Enter your Choice: " << endl;

cin >> option;

if (option == 1)

{

bool game = true;

system("cls");

showMaze();

showSpike();

while (game == true)

{

displayScoreAndHealth();

if (GetAsyncKeyState(VK\_LEFT))

{

moveSpikeLeft();

}

else if (GetAsyncKeyState(VK\_RIGHT))

{

moveSpikeRight();

}

else if (GetAsyncKeyState(VK\_UP))

{

moveSpikeUp();

}

else if (GetAsyncKeyState(VK\_DOWN))

{

moveSpikeDown();

}

else if (GetAsyncKeyState(VK\_ESCAPE))

{

game = false;

}

else if (GetAsyncKeyState(VK\_SPACE))

{

generateBullet();

}

if (crusher\_Health != 0 && timer1 == 3)

{

showCrusher();

moveCrusher();

timer1 = 0;

}

if (crusher\_Health != 0 && spikeX < 60 )

{

eraseCrusherBullet();

generateCrusherBullet();

moveCrusherBulletLeft();

}

if (crusher\_Health != 0 && spikeX > 60)

{

eraseCrusherBullet();

}

if (steelThunder\_Health != 0)

{

showSteelThunder();

moveSteelThunder();

}

if (blaze\_Health != 0 && timer3 == 2)

{

showBlaze();

moveBlaze();

timer3 = 0;

}

if (blaze\_Health != 0)

{

generateBlazeBullet();

moveBlazeBullet();

}

if (crusher\_Health == 0)

{

eraseCrusher();

eraseCrusherBullet();

crusherX = 0;

crusherY = 35;

}

if (steelThunder\_Health == 0)

{

eraseSteelThunder();

steelThunderX = 0;

steelThunderY = 35;

}

if (blaze\_Health == 0)

{

eraseBlaze();

eraseBlazeBullet();

blazeX = 0;

blazeY = 35;

}

if (crusher\_Health == 0 && blaze\_Health == 0 && steelThunder\_Health == 0 && spike\_Health > 0 && spikeLive > 0)

{

system("cls");

winning();

cin.ignore();

game = false;

cout << " Do You want to Continue(Y/N)? ";

cin >> check;

}

if (spikeLive == 0)

{

system("cls");

losing();

cin.ignore();

game = false;

cout << " Do You want to Continue(Y/N)? ";

cin >> check;

}

moveBullet();

bulletCollisionWithBlaze();

bulletCollisionWithSteelThunder();

bulletCollisionWithCrusher();

crusherCollisionWithSpike();

blazeCollisionWithSpike();

steelThunderCollisionWithSpike();

blazeBulletCollisionWithSpike();

crusherBulletCollisionWithSpike();

timer1++;

timer2++;

timer3++;

Sleep(90);

}

}

else if (option == 2)

{

clearScreen();

keys();

clearScreen();

}

else if (option == 3)

{

clearScreen();

cout << endl

<< endl

<< endl

<< endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| THANK YOU FOR PLAYING THIS GAME!!! |" << endl;

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

check = 'N';

}

}

void keys()

{

cout << "~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "| KEYS |" << endl;

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

cout << " UP Go Up" << endl;

cout << " DOWN Go Down" << endl;

cout << " RIGHT Go Right" << endl;

cout << " LEFT Go Left" << endl;

cout << " SPACE Fire Bullets" << endl;

cout << " Esc. Back to Main Menu" << endl;

}

void showMaze()

{

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;

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;

}

void showSpike()

{

gotoxy(spikeX, spikeY);

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

{

for (int j = 0; j < 11; j++)

{

cout << spike[i][j];

}

gotoxy(spikeX, spikeY + 1);

}

}

void eraseSpike()

{

gotoxy(spikeX, spikeY);

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

{

for (int j = 0; j < 11; j++)

{

cout << " ";

}

gotoxy(spikeX, spikeY + 1);

}

}

void moveSpikeLeft()

{

char nextlocation1 = getCharAtxy(spikeX - 1, spikeY);

char nextlocation2 = getCharAtxy(spikeX - 1, spikeY + 1);

char nextlocation3 = getCharAtxy(spikeX - 1, spikeY + 2);

if (nextlocation1 == ' ' && nextlocation2 == ' ' && nextlocation3 == ' ')

{

eraseSpike();

spikeX = spikeX - 1;

showSpike();

}

}

void moveSpikeRight()

{

char nextlocation1 = getCharAtxy(spikeX + 11, spikeY);

char nextlocation2 = getCharAtxy(spikeX + 11, spikeY + 1);

char nextlocation3 = getCharAtxy(spikeX + 11, spikeY + 2);

if (nextlocation1 == ' ' && nextlocation2 == ' ' && nextlocation3 == ' ')

{

eraseSpike();

spikeX = spikeX + 1;

showSpike();

}

}

void moveSpikeUp()

{

char nextlocation1 = getCharAtxy(spikeX, spikeY - 1);

char nextlocation2 = getCharAtxy(spikeX + 1, spikeY - 1);

char nextlocation3 = getCharAtxy(spikeX + 2, spikeY - 1);

char nextlocation4 = getCharAtxy(spikeX + 3, spikeY - 1);

char nextlocation5 = getCharAtxy(spikeX + 4, spikeY - 1);

char nextlocation6 = getCharAtxy(spikeX + 5, spikeY - 1);

char nextlocation7 = getCharAtxy(spikeX + 6, spikeY - 1);

char nextlocation8 = getCharAtxy(spikeX + 7, spikeY - 1);

char nextlocation9 = getCharAtxy(spikeX + 8, spikeY - 1);

char nextlocation10 = getCharAtxy(spikeX + 9, spikeY - 1);

char nextlocation11 = getCharAtxy(spikeX + 10, spikeY - 1);

if (nextlocation1 == ' ' && nextlocation2 == ' ' && nextlocation3 == ' ' && nextlocation4 == ' ' && nextlocation5 == ' ' && nextlocation6 == ' ' && nextlocation7 == ' ' && nextlocation8 == ' ' && nextlocation9 == ' ' && nextlocation10 == ' ' && nextlocation11 == ' ')

{

eraseSpike();

spikeY = spikeY - 1;

showSpike();

}

}

void moveSpikeDown()

{

char nextlocation1 = getCharAtxy(spikeX, spikeY + 3);

char nextlocation2 = getCharAtxy(spikeX + 1, spikeY + 3);

char nextlocation3 = getCharAtxy(spikeX + 2, spikeY + 3);

char nextlocation4 = getCharAtxy(spikeX + 3, spikeY + 3);

char nextlocation5 = getCharAtxy(spikeX + 4, spikeY + 3);

char nextlocation6 = getCharAtxy(spikeX + 5, spikeY + 3);

char nextlocation7 = getCharAtxy(spikeX + 6, spikeY + 3);

char nextlocation8 = getCharAtxy(spikeX + 7, spikeY + 3);

char nextlocation9 = getCharAtxy(spikeX + 8, spikeY + 3);

char nextlocation10 = getCharAtxy(spikeX + 9, spikeY + 3);

char nextlocation11 = getCharAtxy(spikeX + 10, spikeY + 3);

if (nextlocation1 == ' ' && nextlocation2 == ' ' && nextlocation3 == ' ' && nextlocation4 == ' ' && nextlocation5 == ' ' && nextlocation6 == ' ' && nextlocation7 == ' ' && nextlocation8 == ' ' && nextlocation9 == ' ' && nextlocation10 == ' ' && nextlocation11 == ' ')

{

eraseSpike();

spikeY = spikeY + 1;

showSpike();

}

}

void generateBullet()

{

bulletX[bulletCount] = spikeX + 12;

bulletY[bulletCount] = spikeY;

gotoxy(spikeX + 14, spikeY);

cout << "\*";

bulletCount++;

}

void moveBullet()

{

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

{

char nextLocation = getCharAtxy(bulletX[j] + 1, bulletY[j] + 1);

if (nextLocation != ' ' || nextLocation == '|' || nextLocation == '?')

{

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

removeBulletFromArray(j);

}

else if (nextLocation == ' ')

{

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

bulletX[j] = bulletX[j] + 1;

displayBullet2(bulletX[j], bulletY[j]);

}

}

}

void displayBullet(int x, int y)

{

gotoxy(x, y);

cout << ".";

}

void displayBullet2(int x, int y)

{

gotoxy(x, y);

cout << "\*";

}

void eraseBullet(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void removeBulletFromArray(int idx)

{

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

{

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

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

}

bulletCount--;

}

void generateBlazeBullet()

{

blazeBulletX[blazeBulletCount] = blazeX + 4;

blazeBulletY[blazeBulletCount] = blazeY - 1;

gotoxy(blazeX + 4, blazeY - 1);

cout << ".";

blazeBulletCount++;

}

void moveBlazeBullet()

{

for (int j = 0; j < blazeBulletCount; j++)

{

char nextLocation = getCharAtxy(blazeBulletX[j], blazeBulletY[j] - 1);

if (nextLocation != ' ')

{

eraseBullet(blazeBulletX[j], blazeBulletY[j]);

removeBlazeBulletFromArray(j);

}

else if (nextLocation == ' ')

{

eraseBullet(blazeBulletX[j], blazeBulletY[j]);

blazeBulletY[j] = blazeBulletY[j] - 1;

displayBullet(blazeBulletX[j], blazeBulletY[j]);

}

}

}

void removeBlazeBulletFromArray(int idx)

{

for (int x = idx; x < blazeBulletCount - 1; x++)

{

blazeBulletX[x] = blazeBulletX[x + 1];

blazeBulletY[x] = blazeBulletY[x + 1];

}

blazeBulletCount--;

}

void generateCrusherBullet()

{

crusherBulletX[crusherBulletCount] = crusherX ;

crusherBulletY[crusherBulletCount] = crusherY ;

gotoxy(crusherX , crusherY);

cout << ".";

crusherBulletCount++;

}

void moveCrusherBulletLeft()

{

for (int j = 0; j < crusherBulletCount; j++)

{

char nextLocation = getCharAtxy(crusherBulletX[j] - 1, crusherBulletY[j] + 1);

if (nextLocation != ' ' || nextLocation == '|' || nextLocation == '?')

{

eraseBullet(crusherBulletX[j], crusherBulletY[j]);

removeCrusherBulletFromArray(j);

}

else if (nextLocation == ' ')

{

eraseBullet(crusherBulletX[j], crusherBulletY[j]);

crusherBulletX[j] = crusherBulletX[j] - 1;

displayBullet(crusherBulletX[j], crusherBulletY[j]);

}

}

}

void moveCrusherBulletRight()

{

for (int j = 0; j < crusherBulletCount; j++)

{

char nextLocation = getCharAtxy(crusherBulletX[j] + 9, crusherBulletY[j] + 1);

if (nextLocation != ' ' || nextLocation == '|' || nextLocation == '?')

{

eraseBullet(crusherBulletX[j], crusherBulletY[j]);

removeCrusherBulletFromArray(j);

}

else if (nextLocation == ' ')

{

eraseBullet(crusherBulletX[j], crusherBulletY[j]);

crusherBulletX[j] = crusherBulletX[j] + 1;

displayBullet(crusherBulletX[j], crusherBulletY[j]);

}

}

}

void removeCrusherBulletFromArray(int idx)

{

for (int x = idx; x < crusherBulletCount - 1; x++)

{

crusherBulletX[x] = crusherBulletX[x + 1];

crusherBulletY[x] = crusherBulletY[x + 1];

}

crusherBulletCount--;

}

void eraseBlazeBullet()

{

for (int i = 0; i < blazeBulletCount; i++)

{

eraseBullet(blazeBulletX[i], blazeBulletY[i]);

}

}

void eraseCrusherBullet()

{

for (int i = 0; i < crusherBulletCount; i++)

{

eraseBullet(crusherBulletX[i], crusherBulletY[i]);

}

}

void showCrusher()

{

gotoxy(crusherX, crusherY);

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

{

for (int j = 0; j < 9; j++)

{

cout << crusher[i][j];

}

gotoxy(crusherX, crusherY + 1);

}

}

void eraseCrusher()

{

gotoxy(crusherX, crusherY);

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

{

for (int j = 0; j < 9; j++)

{

cout << " ";

}

gotoxy(crusherX, crusherY + 1);

}

}

void moveCrusher()

{

if (crusherDirection == "Down")

{

char next = getCharAtxy(crusherX, crusherY + 3);

if (next == ' ')

{

eraseCrusher();

crusherY = crusherY + 1;

showCrusher();

}

else

{

crusherDirection = "Up";

}

}

if (crusherDirection == "Up")

{

char next = getCharAtxy(crusherX, crusherY - 1);

if (next == ' ')

{

eraseCrusher();

crusherY = crusherY - 1;

showCrusher();

}

else

{

crusherDirection = "Down";

}

}

}

void showSteelThunder()

{

gotoxy(steelThunderX, steelThunderY);

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

{

for (int j = 0; j < 7; j++)

{

cout << steelThunder[i][j];

}

gotoxy(steelThunderX, steelThunderY + 1);

}

}

void eraseSteelThunder()

{

gotoxy(steelThunderX, steelThunderY);

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

{

for (int j = 0; j < 7; j++)

{

cout << " ";

}

gotoxy(steelThunderX, steelThunderY + 1);

}

}

void moveSteelThunder()

{

if (steelThunderDirection == "Down")

{

char next = getCharAtxy(steelThunderX, steelThunderY + 3);

if (next == ' ')

{

eraseSteelThunder();

steelThunderY = steelThunderY + 1;

showSteelThunder();

}

if (next == '|')

{

eraseSteelThunder();

steelThunderX = 121;

steelThunderY = 21;

steelThunderDirection = "Up";

}

}

if (steelThunderDirection == "Up")

{

char next = getCharAtxy(steelThunderX, steelThunderY - 1);

if (next == ' ')

{

eraseSteelThunder();

steelThunderY = steelThunderY - 1;

showSteelThunder();

}

if (next == '|')

{

eraseSteelThunder();

steelThunderX = 4;

steelThunderY = 4;

steelThunderDirection = "Down";

}

}

}

void showBlaze()

{

gotoxy(blazeX, blazeY);

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

{

for (int j = 0; j < 9; j++)

{

cout << blaze[i][j];

}

gotoxy(blazeX, blazeY + 1);

}

}

void eraseBlaze()

{

gotoxy(blazeX, blazeY);

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

{

for (int j = 0; j < 9; j++)

{

cout << " ";

}

gotoxy(blazeX, blazeY + 1);

}

}

void moveBlaze()

{

if (blazeDirection == "Right")

{

char next = getCharAtxy(blazeX + 9, blazeY);

if (next == ' ')

{

eraseBlaze();

blazeX = blazeX + 1;

showBlaze();

}

else

{

blazeDirection = "Left";

}

}

if (blazeDirection == "Left")

{

char next = getCharAtxy(blazeX - 1, blazeY);

if (next == ' ')

{

eraseBlaze();

blazeX = blazeX - 1;

showBlaze();

}

else

{

blazeDirection = "Right";

}

}

}

void bulletCollisionWithBlaze()

{

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

{

if (bulletX[x] + 1 == blazeX && (bulletY[x] == blazeY || bulletY[x] == blazeY + 1 || bulletY[x] == blazeY + 2))

{

if (blaze\_Health > 0)

{

addScore();

blazeHealth();

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

removeBulletFromArray(x);

}

}

}

}

void bulletCollisionWithSteelThunder()

{

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

{

if (bulletX[x] + 1 == steelThunderX && (bulletY[x] == steelThunderY || bulletY[x] == steelThunderY + 1 || bulletY[x] == steelThunderY + 2))

{

addScore();

steelThunderHealth();

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

removeBulletFromArray(x);

}

}

}

void bulletCollisionWithCrusher()

{

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

{

if (bulletX[x] + 1 == crusherX && (bulletY[x] == crusherY || bulletY[x] == crusherY + 1 || bulletY[x] == crusherY + 2))

{

addScore();

crusherHealth();

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

removeBulletFromArray(x);

}

}

}

void crusherCollisionWithSpike()

{

if ((spikeX == crusherX || spikeX + 11 == crusherX || crusherX + 9 == spikeX || crusherX + 9 == spikeX + 11) && (spikeY == crusherY || spikeY == crusherY + 1 || spikeY == crusherY + 2 || spikeY + 1 == crusherY || spikeY + 2 == crusherY))

{

spikeHealth();

}

}

void steelThunderCollisionWithSpike()

{

if ((spikeX == steelThunderX || spikeX + 11 == steelThunderX || steelThunderX + 7 == spikeX || spikeX + 11 == steelThunderX + 7) && (spikeY == steelThunderY || spikeY == steelThunderY + 1 || spikeY == steelThunderY + 2 || spikeY + 1 == steelThunderY || spikeY + 2 == steelThunderY))

{

spikeHealth();

}

}

void blazeCollisionWithSpike()

{

if ((spikeX == blazeX || spikeX + 11 == blazeX || blazeX + 9 == spikeX || spikeX + 11 == blazeX + 9) && (spikeY == blazeY || spikeY + 1 == blazeY || spikeY + 2 == blazeY || spikeY == blazeY + 1 || spikeY == blazeY + 2))

{

spikeHealth();

}

}

void blazeBulletCollisionWithSpike()

{

for (int i = 0; i < blazeBulletCount; i++)

{

if ((blazeBulletX[i] == spikeX || blazeBulletX[i] == spikeX + 1 || blazeBulletX[i] == spikeX + 2 || blazeBulletX[i] == spikeX + 3 || blazeBulletX[i] == spikeX + 4 || blazeBulletX[i] == spikeX + 5 || blazeBulletX[i] == spikeX + 6 || blazeBulletX[i] == spikeX + 7 || blazeBulletX[i] == spikeX + 8 || blazeBulletX[i] == spikeX + 9 || blazeBulletX[i] == spikeX + 10) && blazeBulletY[i] == spikeY + 3)

{

eraseBullet(blazeBulletX[i], blazeBulletY[i]);

spikeHealth();

}

}

}

void crusherBulletCollisionWithSpike()

{

for (int i = 0; i < crusherBulletCount; i++)

{

if ((crusherBulletX[i] == spikeX || crusherBulletX[i] == spikeX + 1 || crusherBulletX[i] == spikeX + 2 || crusherBulletX[i] == spikeX + 3 || crusherBulletX[i] == spikeX + 4 || crusherBulletX[i] == spikeX + 5 || crusherBulletX[i] == spikeX + 6 || crusherBulletX[i] == spikeX + 7 || crusherBulletX[i] == spikeX + 8 || crusherBulletX[i] == spikeX + 9 || crusherBulletX[i] == spikeX + 10) && (crusherBulletY[i] == spikeY + 3 || crusherBulletY[i] == spikeY + 2 || crusherBulletY[i] == spikeY))

{

eraseBullet(crusherBulletX[i], crusherBulletY[i]);

spikeHealth();

}

}

}

void addScore()

{

score = score + 1;

}

void crusherHealth()

{

if (crusher\_Health > 0)

{

crusher\_Health--;

}

}

void steelThunderHealth()

{

if (steelThunder\_Health > 0)

{

steelThunder\_Health = steelThunder\_Health - 1;

}

}

void blazeHealth()

{

if (blaze\_Health > 0)

{

blaze\_Health = blaze\_Health - 1;

}

}

void spikeHealth()

{

if (spike\_Health > 0)

{

spike\_Health--;

}

else if (spike\_Health == 0 && spikeLive > 0)

{

spikeLive--;

spike\_Health = 10;

}

else if (spikeLive == 0)

{

spike\_Health = 0;

}

}

void displayScoreAndHealth()

{

gotoxy(10, 29);

cout << "SCORE = " << score << " ";

gotoxy(10, 30);

cout << "LIVES = " << spikeLive << " ";

gotoxy(10, 31);

cout << "SPIKE HEALTH = " << spike\_Health << " ";

gotoxy(10, 32);

cout << "CRUSHER HEALTH = " << crusher\_Health << " ";

gotoxy(10, 33);

cout << "STEEL THUNDER HEALTH = " << steelThunder\_Health << " ";

gotoxy(10, 34);

cout << "BLAZE HEALTH = " << blaze\_Health << " ";

}

void clearScreen()

{

cout << endl;

cout << "Press Any key to continue: ";

getch();

system("cls");

}

void gotoxy(int x, int y)

{

COORD coordinates;

coordinates.X = x;

coordinates.Y = y;

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

}

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 : ' ';

}