**Chopper Command**



Session 2022 - 2025

**Submitted By:**

Bilal Ahmad, 2022-CS-170

**Supervised by:**

Dr. Awais Hassan

Department of Computer Science

**University of Engineering and Technology**



Table of Contents

[Short Description and Story 0f Game 2](#_Toc130296713)

[Game Characters Description 2](#_Toc130296714)

[Player 2](#_Toc130296715)

[Gunship-Pilot: 2](#_Toc130296716)

[Enemies 2](#_Toc130296717)

[Helicopter: 2](#_Toc130296718)

[Plane: 2](#_Toc130296719)

[Jet: 2](#_Toc130296720)

[Game Objects Description 3](#_Toc130296721)

[Bullets: 3](#_Toc130296722)

[Walls: 3](#_Toc130296723)

[Rules & Interactions 3](#_Toc130296724)

[Goal of the Game 3](#_Toc130296725)

[Wireframes 3](#_Toc130296726)

[Figure 1: Logo 4](#_Toc130296727)

[Figure 2: Main Menu 4](#_Toc130296728)

[Figure 3: Gameplay View 4](#_Toc130296729)

[Figure 4: Options Menu 5](#_Toc130296730)

[Figure 5: Keys Information 5](#_Toc130296731)

[Figure 6: Characters Information 5](#_Toc130296732)

[Data Structures: 5](#_Toc130296733)

[Function Prototypes 9](#_Toc130296734)

[Complete Code 12](#_Toc130296735)

# Short Description and Story 0f Game

The game takes place in a war where you are a military officer and your work to maintain peace by defeating enemies of justice.

Chopper Command is a side shooter game. As the pilot of a gunship, your mission is to defeat all the enemies to maintain peace. Each wave consists of some enemy choppers and jets which are trying to destroy you. When all enemies in the current wave are destroyed, you can move on to the next wave.

You are given three lives. If you are hit by any of the explosive material emitted by the enemies you will lose a life. Once all three of your lives has been lost the game will be ended. The enemies will attack you in a specific pattern.

# Game Characters Description

## Player

There is one human player in the game.

Gunship-Pilot: Gunship Pilot is the main character in the game and is known for saving the convoy of trucks in the war. He will be using a helicopter to battle with invading enemies. He will try his best to protect the convoy from these invading jets and helicopters.

Enemies There are 3 enemies in the game whom will attack you in a wave.

Helicopter:  
Helicopter is one of the three enemies in the game and is known for being aggressive and difficult to shake. It is always trying to shoot us down through the entire maze. It is fast and relentless, making it one of the most dangerous foes that Gunship-Pilot must face.

Plane:  
Plane is one of the enemies in the game and is known for its unpredictable movements and moves vertically in the Game.

Jet:Jet is one of the enemies in the game and is known for his horizontal movement.

# Game Objects Description

Following are the Objects in the Game:

## Bullets:

Bullets are emitted by plane and jet and the Gunship-Pilot should be aware to not to get hit by them.

## Walls:

Walls are the barriers in the game which the Gunship-Pilot and the enemies cannot cross.

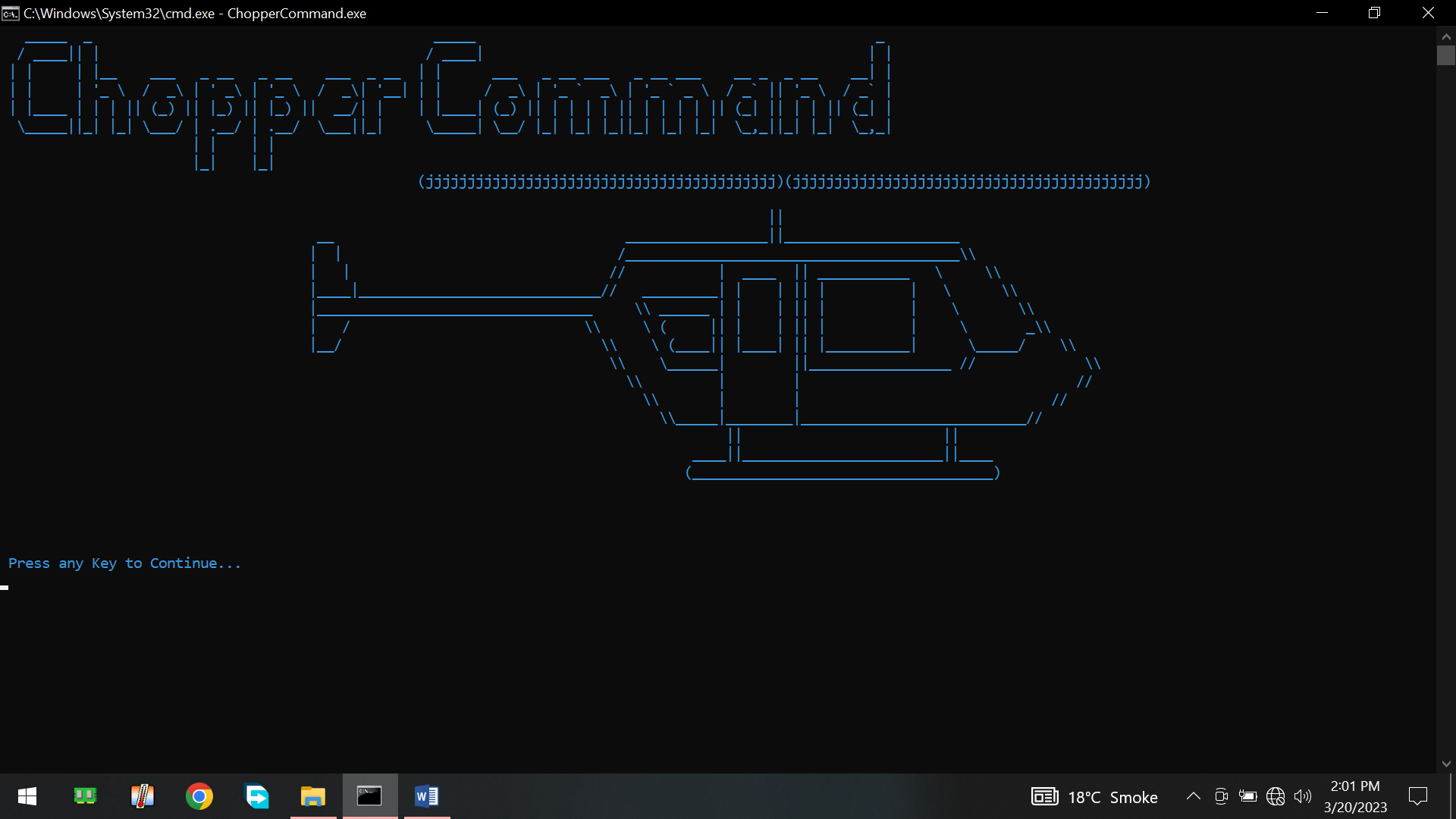
# Rules & Interactions

The Gunship-Pilot needs to protect himself from the bullets. If he got hit by them 1 life will be deducted. If main player collides with any of the enemy its health will start reducing. Gunship-Pilot needs to defeat all the enemies. Gunship-Pilot will be given 3 lives and the game will be finished when all 3 lives have been deducted. Score will increase when you fire at enemies.

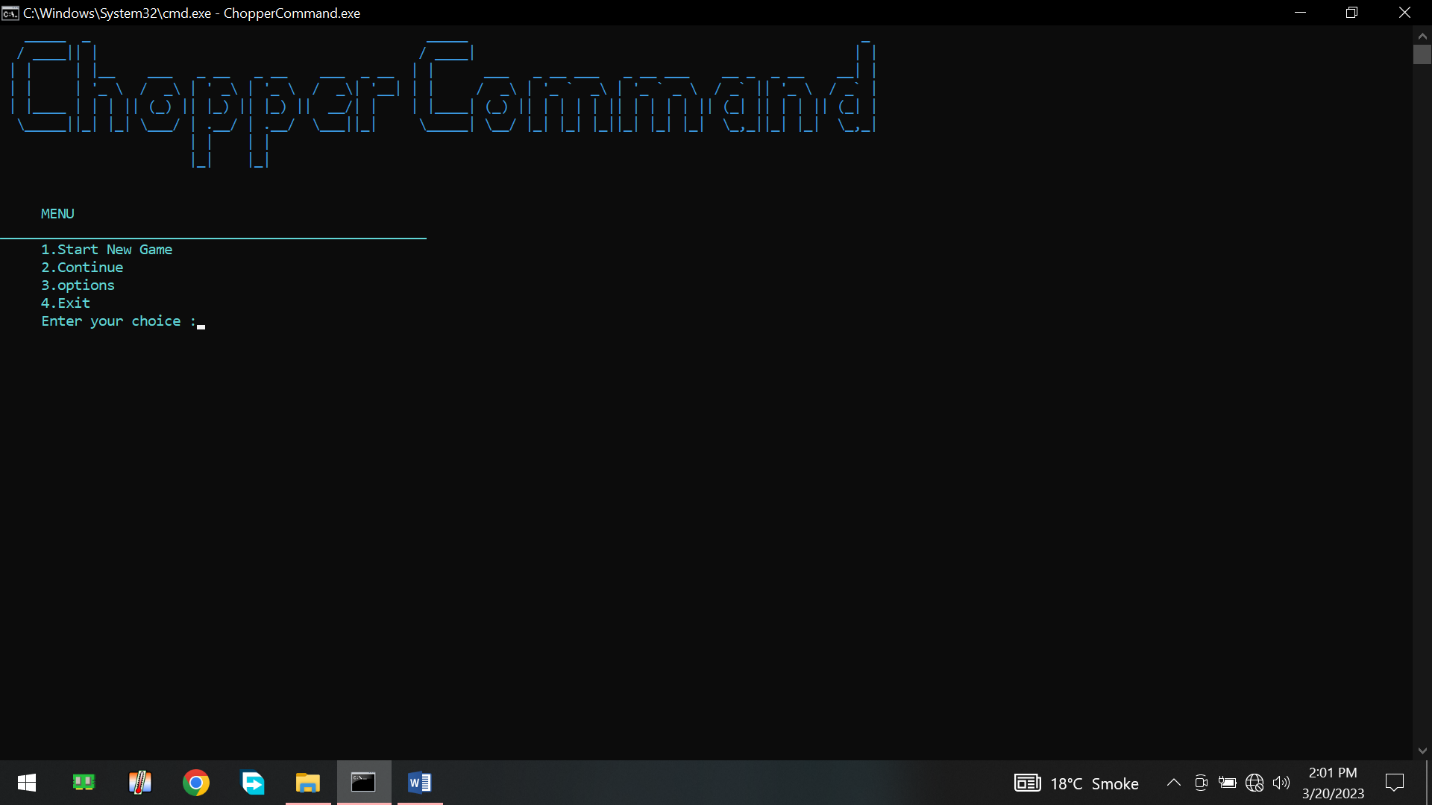
# Goal of the Game

The goal of the game is to defeat all the enemies and end the war.

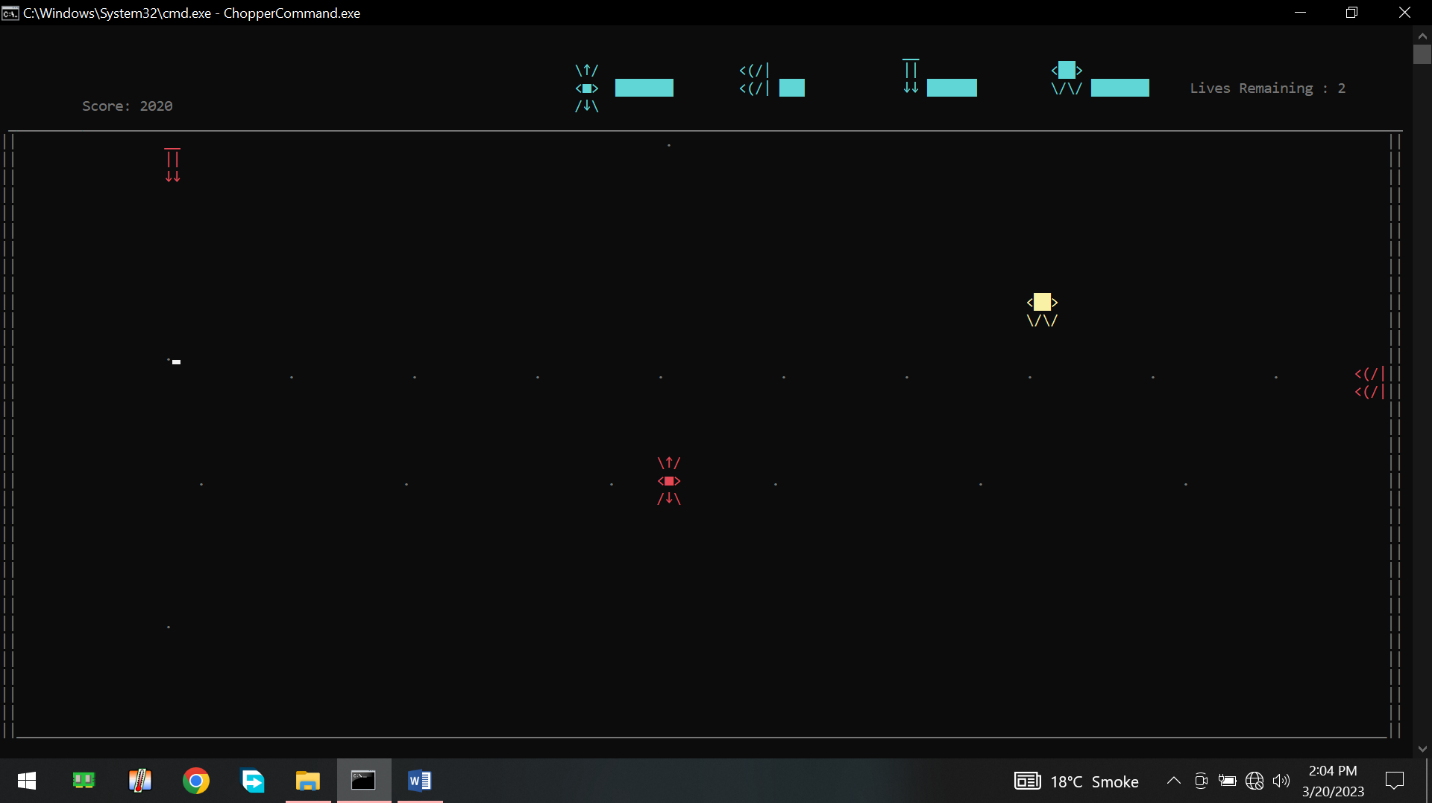
# Wireframes



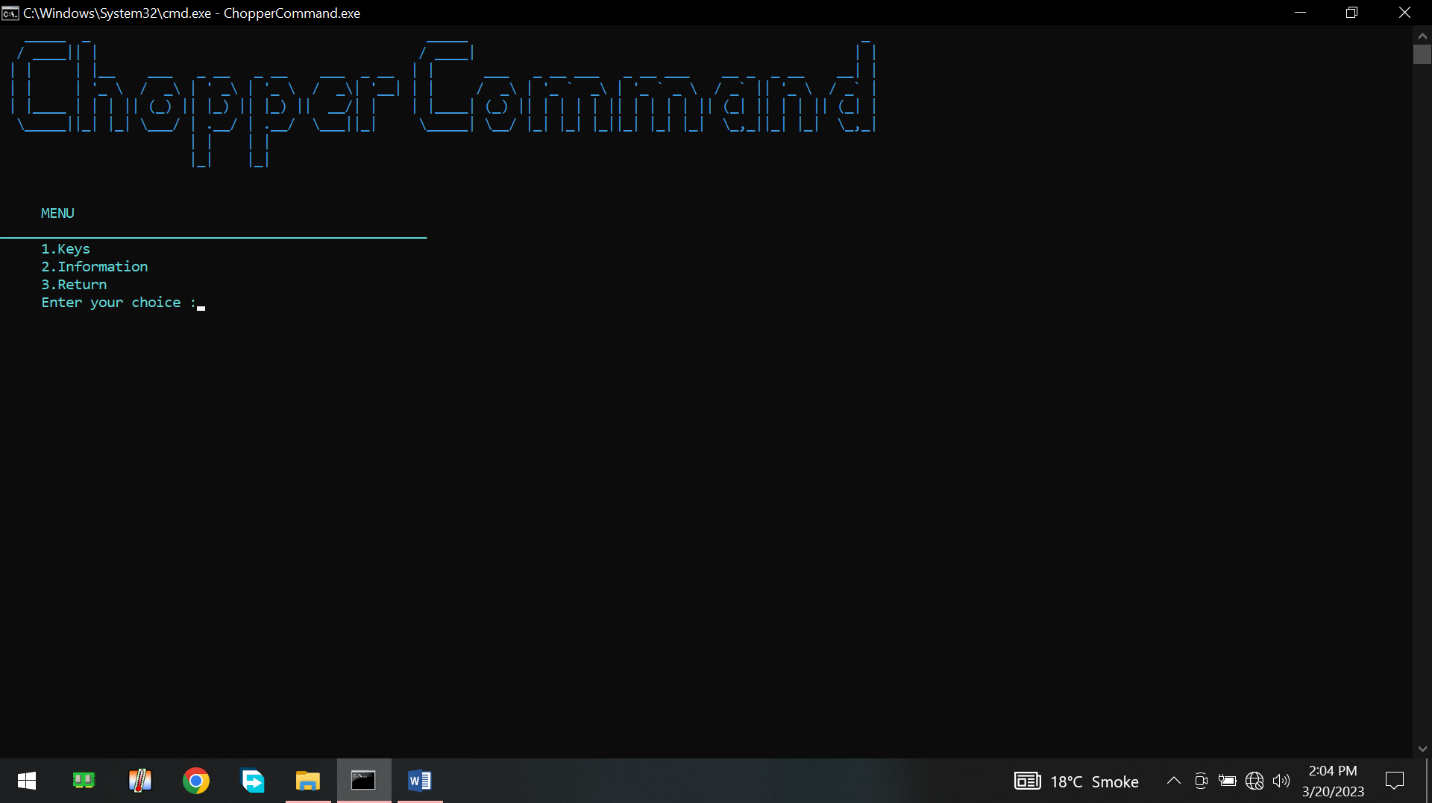
## Figure 1: Logo



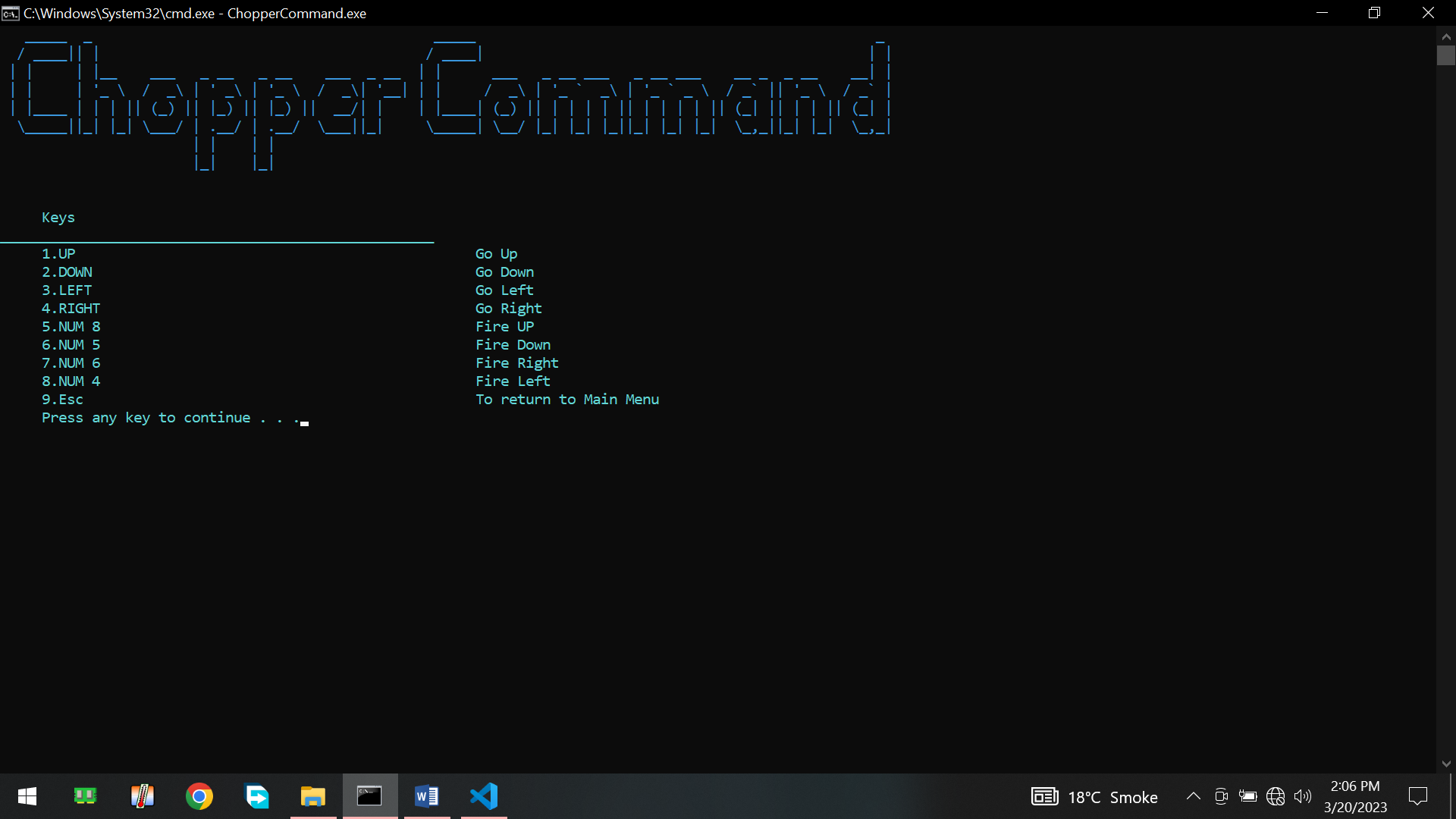
|  |  |
| --- | --- |
| |  | | --- | | Figure 2: Main Menu | |



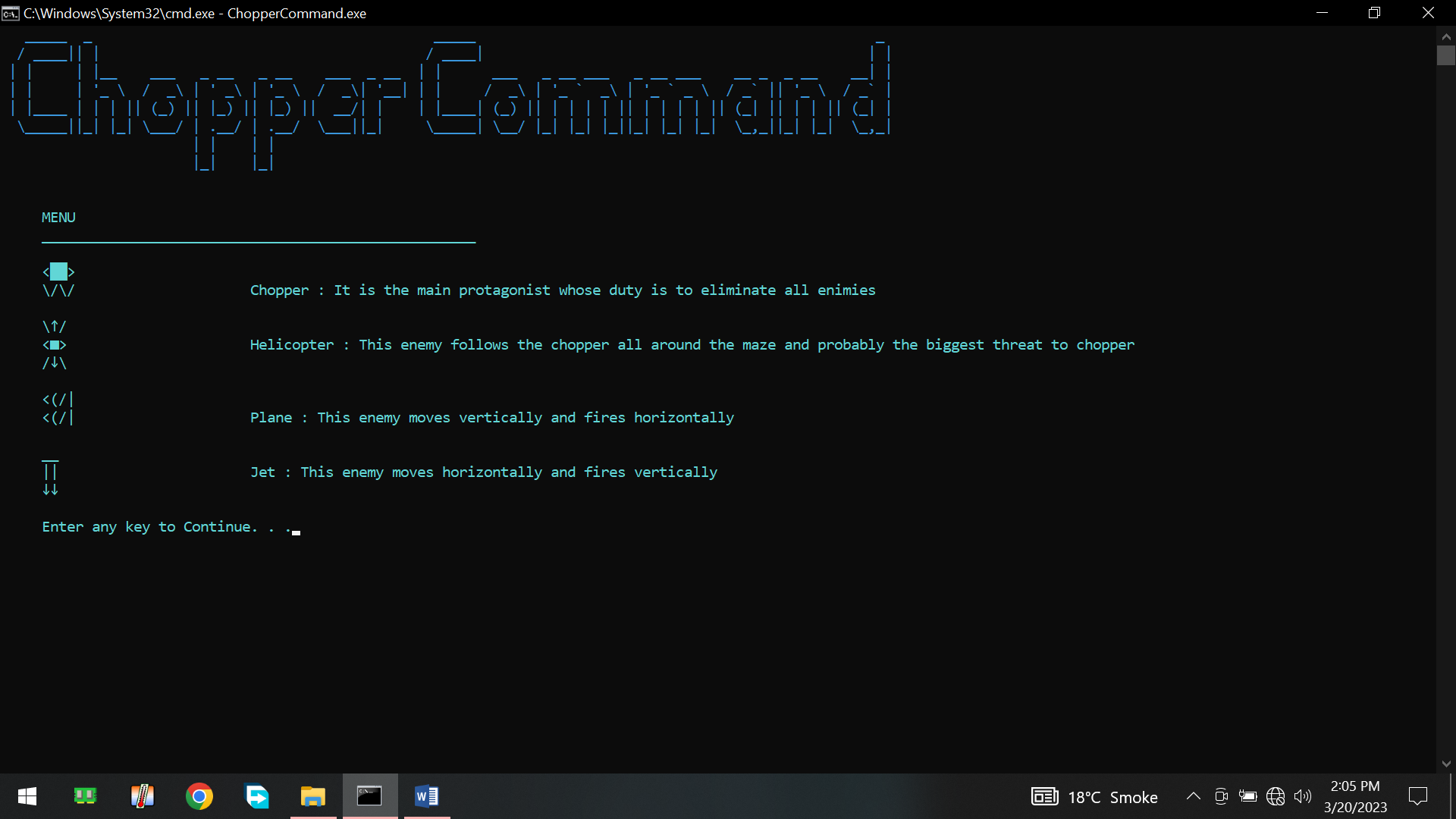
## Figure 3: Gameplay View



## Figure 4: Options Menu



## Figure 5: Keys Information



## Figure 6: Characters Information

# Data Structures:

// Chopper structure

char chopper[2][4] = {

{'<', box, box, '>'},

{'\\', '/', '\\', '/'}};

// horizontal Enemy structure

char enemyH[3][2] = {

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

{'|', '|'},

{head, head}};

// vertical enemy structure

char enemyV[2][4] = {

{'<', '(', '/', '|'},

{'<', '(', '/', '|'}};

// chasing enemy structure

char enemy[3][3] = {

{'\\', tail, '/'},

{'<', body, '>'},

{'/', head, '\\'}};

string grid[35][1] = {

{" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||"},

};

# Function Prototypes

void gotoxy(int x, int y);

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

void mainScreen();

void header();

void maze();

void menu();

bool menuValidity(string option);

void Options();

void keys();

void instructions();

void printChopper();

void eraseChopper();

void moveChopperLeft();

void moveChopperRight();

void moveChopperUp();

void moveChopperDown();

void printEnemyV();

void eraseEnemyV();

void moveEnemyV();

void printEnemyH();

void eraseEnemyH();

void moveEnemyH();

void printEnemy();

void eraseEnemy();

void moveEnemy(int x, int y);

void generateBulletRight();

void generateBulletLeft();

void generateBulletUp();

void generateBulletDown();

void generateEnemyBulletRight();

void generateEnemyBulletLeft();

void generateEnemyBulletUp();

void generateEnemyBulletDown();

void moveEnemyBulletRight();

void moveEnemyBulletLeft();

void moveEnemyBulletUp();

void moveEnemyBulletDown();

void generateEnemyVBullet();

void moveBulletV();

void generateEnemyHBullet();

void moveBulletH();

void moveBulletRight();

void moveBulletLeft();

void moveBulletUp();

void moveBulletDown();

void bulletCollisionWithEnemyRight();

void bulletCollisionWithEnemyLeft();

void bulletCollisionWithEnemyUp();

void bulletCollisionWithEnemyDown();

void bulletCollisionWithEnemyVRight();

void bulletCollisionWithEnemyVUp();

void bulletCollisionWithEnemyVDown();

void bulletCollisionWithEnemyHRight();

void bulletCollisionWithEnemyHLeft();

void bulletCollisionWithEnemyHUp();

void printBullet(int x, int y);

void eraseBullet(int x, int y);

void addScore();

void printScore();

void printLives();

void changeChopperHealth();

void printChopperHealth();

void changeEnemyHealth();

void printEnemyHealth();

void changeEnemyVHealth();

void printEnemyVHealth();

void changeEnemyHHealth();

void printEnemyHHealth();

bool ChopperCollisionWithEnemy(int CX, int CY, int EX, int EY);

bool ChopperCollisionWithEnemyH(int CX, int CY, int EX, int EY);

bool ChopperCollisionWithEnemyV(int CX, int CY, int EX, int EY);

void bulletVCollisionWithChopper();

void bulletHCollisionWithChopper();

void EnemyBulletsCollisionWithChopper();

void EnemyBulletsCollisionWithChopperUp();

void EnemyBulletsCollisionWithChopperDowwn();

void EnemyBulletsCollisionWithChopperLeft();

void EnemyBulletsCollisionWithChopperRight();

void saving();

void loading();

void loadingLives();

string parsing(string word, int location);

# Complete Code

#include <iostream>

#include <windows.h>

#include <conio.h>

#include <fstream>

using namespace std;

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

//////////////////////////////////////////////Prototypes//////////////////////////////////////

void gotoxy(int x, int y);

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

void mainScreen();

void header();

void maze();

void menu();

bool menuValidity(string option);

void Options();

void keys();

void instructions();

void printChopper();

void eraseChopper();

void moveChopperLeft();

void moveChopperRight();

void moveChopperUp();

void moveChopperDown();

void printEnemyV();

void eraseEnemyV();

void moveEnemyV();

void printEnemyH();

void eraseEnemyH();

void moveEnemyH();

void printEnemy();

void eraseEnemy();

void moveEnemy(int x, int y);

void generateBulletRight();

void generateBulletLeft();

void generateBulletUp();

void generateBulletDown();

void generateEnemyBulletRight();

void generateEnemyBulletLeft();

void generateEnemyBulletUp();

void generateEnemyBulletDown();

void moveEnemyBulletRight();

void moveEnemyBulletLeft();

void moveEnemyBulletUp();

void moveEnemyBulletDown();

void generateEnemyVBullet();

void moveBulletV();

void generateEnemyHBullet();

void moveBulletH();

void moveBulletRight();

void moveBulletLeft();

void moveBulletUp();

void moveBulletDown();

void bulletCollisionWithEnemyRight();

void bulletCollisionWithEnemyLeft();

void bulletCollisionWithEnemyUp();

void bulletCollisionWithEnemyDown();

void bulletCollisionWithEnemyVRight();

void bulletCollisionWithEnemyVUp();

void bulletCollisionWithEnemyVDown();

void bulletCollisionWithEnemyHRight();

void bulletCollisionWithEnemyHLeft();

void bulletCollisionWithEnemyHUp();

void printBullet(int x, int y);

void eraseBullet(int x, int y);

void addScore();

void printScore();

void printLives();

void changeChopperHealth();

void printChopperHealth();

void changeEnemyHealth();

void printEnemyHealth();

void changeEnemyVHealth();

void printEnemyVHealth();

void changeEnemyHHealth();

void printEnemyHHealth();

bool ChopperCollisionWithEnemy(int CX, int CY, int EX, int EY);

bool ChopperCollisionWithEnemyH(int CX, int CY, int EX, int EY);

bool ChopperCollisionWithEnemyV(int CX, int CY, int EX, int EY);

void bulletVCollisionWithChopper();

void bulletHCollisionWithChopper();

void EnemyBulletsCollisionWithChopper();

void EnemyBulletsCollisionWithChopperUp();

void EnemyBulletsCollisionWithChopperDowwn();

void EnemyBulletsCollisionWithChopperLeft();

void EnemyBulletsCollisionWithChopperRight();

void saving();

void loading();

void loadingLives();

string parsing(string word, int location);

////////////////////////////////////////Global Variables/////////////////////////////////////

// chopper coordinates

int chopperX = 10;

int chopperY = 10;

// chopper structure

char box = 219;

char chopper[2][4] = {

{'<', box, box, '>'},

{'\\', '/', '\\', '/'}};

string enemyHDirection = "Right";

// horizonatal Enemy coordinates

int enemyHX = 20;

int enemyHY = 7;

// horizontal Enemy structure

char head = 25;

char enemyH[3][2] = {

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

{'|', '|'},

{head, head}};

string enemyVDirection = "Up";

// Vertical Enemy coordinates

int enemyVX = 165;

int enemyVY = 20;

// vertical enemy structure

char enemyV[2][4] = {

{'<', '(', '/', '|'},

{'<', '(', '/', '|'}};

// chasing enemy coordinates

int enemyX = 80;

int enemyY = 25;

// chasing enemy structure

char tail = 24;

char body = 254;

char enemy[3][3] = {

{'\\', tail, '/'},

{'<', body, '>'},

{'/', head, '\\'}};

// chopper bullets variables

int bulletXLeft[1000];

int bulletYLeft[1000];

bool isbulletActiveLeft[1000];

int bulletCountLeft = 0;

int bulletXRight[1000];

int bulletYRight[1000];

bool isbulletActiveRight[1000];

int bulletCountRight = 0;

int bulletXUp[1000];

int bulletYUp[1000];

bool isbulletActiveUp[1000];

int bulletCountUp = 0;

int bulletXDown[1000];

int bulletYDown[1000];

bool isbulletActiveDown[1000];

int bulletCountDown = 0;

// chasing enemys bullet variables

int EnemybulletXLeft[1000];

int EnemybulletYLeft[1000];

bool isEnemybulletActiveLeft[1000];

int EnemybulletCountLeft = 0;

int EnemybulletXRight[1000];

int EnemybulletYRight[1000];

bool isEnemybulletActiveRight[1000];

int EnemybulletCountRight = 0;

int EnemybulletXUp[1000];

int EnemybulletYUp[1000];

bool isEnemybulletActiveUp[1000];

int EnemybulletCountUp = 0;

int EnemybulletXDown[1000];

int EnemybulletYDown[1000];

bool isEnemybulletActiveDown[1000];

int EnemybulletCountDown = 0;

int score = 0;

char health = 219;

char HenemyHealth[10] = {health, health, health, health, health, health, health, health, health, health};

char VenemyHealth[10] = {health, health, health, health, health, health, health, health, health, health};

char enemyHealth[10] = {health, health, health, health, health, health, health, health, health, health};

char ChopperHealth[10] = {health, health, health, health, health, health, health, health, health, health};

int hCountChopper = 20;

int hCountEnemy = 50;

int hCountEnemyV = 30;

int hCountEnemyH = 30;

int EH = 9;

int EHH = 9;

int EVH = 9;

int CH = 9;

int lives = 3;

int bulletVX[1000];

int bulletVY[1000];

bool isVbulletActive[1000];

int bulletCountV = 0;

int bulletHX[1000];

int bulletHY[1000];

bool isHbulletActive[1000];

int bulletCountH = 0;

string grid[35][1] = {

{" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"|| ||"},

{"||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||"},

};

///////////////////////////////////////Main function////////////////////////////////////////

int main()

{

int timer = 0;

int enemyBulletTimer = 0;

int BulletTimer = 0;

int loopE = 0;

int loopEV = 0;

int loopEH = 0;

bool flag;

bool returnFlag;

bool flag1;

string choice;

mainScreen();

getch();

while (true)

{

header();

menu();

gotoxy(5, 16);

cout << "Enter your choice :";

cin >> choice;

flag1 = menuValidity(choice);

while (flag1 == false)

{

gotoxy(5, 17);

cout << "Enter a valid choice !!!";

gotoxy(24, 16);

cout << " ";

gotoxy(24, 16);

cin >> choice;

flag1 = menuValidity(choice);

}

if (choice == "1" || choice == "2")

{

score = 0;

lives = 3;

while (true)

{

flag = false;

returnFlag = false;

bulletCountLeft = 0;

bulletCountRight = 0;

bulletCountUp = 0;

bulletCountDown = 0;

EnemybulletCountLeft = 0;

EnemybulletCountRight = 0;

EnemybulletCountUp = 0;

EnemybulletCountDown = 0;

loopE = 0;

loopEV = 0;

loopEH = 0;

// chopper coordinates

chopperX = 10;

chopperY = 10;

hCountChopper = 40;

CH = 9;

if (choice == "1" || choice == "2")

{

hCountEnemy = 100;

hCountEnemyV = 60;

hCountEnemyH = 60;

EH = 9;

EHH = 9;

EVH = 9;

// horizonatal Enemy coordinates

enemyHX = 20;

enemyHY = 7;

// Vertical Enemy coordinates

enemyVX = 165;

enemyVY = 20;

// chasing enemy coordinates

enemyX = 80;

enemyY = 25;

}

if (choice == "2")

{

loadingLives();

if (lives != 0)

loading();

else

lives = 3;

}

choice = "-1";

for (int i = 0; i < 10; i++)

{

if (i < EHH)

HenemyHealth[i] = health;

else

HenemyHealth[i] = ' ';

}

for (int i = 0; i < 10; i++)

{

if (i < EVH)

VenemyHealth[i] = health;

else

VenemyHealth[i] = ' ';

}

for (int i = 0; i < 10; i++)

{

if (i < EH)

enemyHealth[i] = health;

else

enemyHealth[i] = ' ';

}

for (int i = 0; i < 10; i++)

{

if (i < CH)

ChopperHealth[i] = health;

else

ChopperHealth[i] = ' ';

}

maze();

printEnemyV();

printEnemyH();

printEnemy();

printChopper();

printEnemyVHealth();

printEnemyHHealth();

printEnemyHealth();

printChopperHealth();

while (true)

{

printScore();

printLives();

if (GetAsyncKeyState(VK\_LEFT))

{

moveChopperLeft();

}

if (GetAsyncKeyState(VK\_RIGHT))

{

moveChopperRight();

}

if (GetAsyncKeyState(VK\_UP))

{

moveChopperUp();

}

if (GetAsyncKeyState(VK\_DOWN))

{

moveChopperDown();

}

if (GetAsyncKeyState(VK\_NUMPAD6))

{

generateBulletRight();

}

if (GetAsyncKeyState(VK\_NUMPAD4))

{

generateBulletLeft();

}

if (GetAsyncKeyState(VK\_NUMPAD8))

{

generateBulletUp();

}

if (GetAsyncKeyState(VK\_NUMPAD5))

{

generateBulletDown();

}

if (GetAsyncKeyState(VK\_ESCAPE))

{

returnFlag = true;

break;

}

if (BulletTimer == 15 && hCountEnemyH != 0)

generateEnemyVBullet();

if (BulletTimer == 15 && hCountEnemyH != 0)

generateEnemyHBullet();

flag = ChopperCollisionWithEnemy(chopperX, chopperY, enemyX, enemyY);

if (flag == true)

{

changeChopperHealth();

}

flag = ChopperCollisionWithEnemyH(chopperX, chopperY, enemyHX, enemyHY);

if (flag == true)

{

changeChopperHealth();

}

flag = ChopperCollisionWithEnemyV(chopperX, chopperY, enemyVX, enemyVY);

if (flag == true)

{

changeChopperHealth();

}

if (enemyBulletTimer == 6 && hCountEnemy != 0)

generateEnemyBulletDown();

else if (enemyBulletTimer == 12 && hCountEnemy != 0)

generateEnemyBulletUp();

else if (enemyBulletTimer == 18 && hCountEnemy != 0)

generateEnemyBulletRight();

else

{

if (enemyBulletTimer == 24 && hCountEnemy != 0)

generateEnemyBulletLeft();

}

if (hCountChopper != 0)

{

bulletVCollisionWithChopper();

bulletHCollisionWithChopper();

EnemyBulletsCollisionWithChopper();

}

moveBulletRight();

moveBulletLeft();

moveBulletUp();

moveBulletDown();

moveEnemyBulletRight();

moveEnemyBulletLeft();

moveEnemyBulletUp();

moveEnemyBulletDown();

moveBulletV();

moveBulletH();

if (hCountEnemy != 0)

{

if (timer == 3)

moveEnemy(chopperX, chopperY);

bulletCollisionWithEnemyRight();

bulletCollisionWithEnemyLeft();

bulletCollisionWithEnemyUp();

bulletCollisionWithEnemyDown();

}

else

{

if (loopE == 0)

{

eraseEnemy();

enemyX = 0;

enemyY = 0;

loopE++;

}

}

if (hCountEnemyV != 0)

{

if (timer == 3)

moveEnemyV();

bulletCollisionWithEnemyVRight();

bulletCollisionWithEnemyVUp();

bulletCollisionWithEnemyVDown();

}

else

{

if (loopEV == 0)

{

eraseEnemyV();

enemyVX = 0;

enemyVY = 0;

loopEV++;

}

}

if (hCountEnemyH != 0)

{

if (timer == 3)

moveEnemyH();

bulletCollisionWithEnemyHRight();

bulletCollisionWithEnemyHLeft();

bulletCollisionWithEnemyHUp();

}

else

{

if (loopEH == 0)

{

eraseEnemyH();

enemyHX = 0;

enemyHY = 0;

loopEH++;

}

}

if (lives == 1 && (hCountChopper == 0))

{

lives--;

system("cls");

gotoxy(75, 25);

cout << "Game Overrrr!!";

Sleep(1000);

break;

}

else

{

if (hCountChopper == 0)

{

lives--;

gotoxy(75, 25);

cout << "One life has been reduced!";

Sleep(1000);

break;

}

}

if (timer == 3)

timer = 0;

if (BulletTimer == 15)

BulletTimer = 0;

if (enemyBulletTimer == 25)

enemyBulletTimer = 0;

BulletTimer++;

timer++;

enemyBulletTimer++;

Sleep(60);

if ((hCountEnemyV == 0) && (hCountEnemy == 0) && (hCountEnemyH == 0))

{

system("cls");

gotoxy(75, 25);

cout << "You Won!!";

Sleep(1000);

getch();

break;

}

saving();

}

saving();

if (returnFlag == true)

break;

if (((hCountEnemyV == 0) && (hCountEnemy == 0) && (hCountEnemyH == 0)) || ((lives == 0)))

{

Sleep(1000);

break;

}

else

{

Sleep(1000);

}

}

}

else if (choice == "3")

{

header();

Options();

gotoxy(5, 15);

cout << "Enter your choice :";

cin >> choice;

flag1 = menuValidity(choice);

while (flag1 == false)

{

gotoxy(5, 16);

cout << "Enter a valid choice !!!";

gotoxy(24, 15);

cout << " ";

gotoxy(24, 15);

cin >> choice;

flag1 = menuValidity(choice);

}

if (choice == "1")

{

keys();

getch();

}

else if (choice == "2")

{

instructions();

getch();

}

else

{

}

}

else

{

return 0;

}

getch();

}

}

//////////////////////////////////////////other functions//////////////////////////////////////////////////////

void printChopper()

{

SetConsoleTextAttribute(hConsole, 14);

gotoxy(chopperX, chopperY);

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 4; col++)

{

cout << chopper[row][col];

}

gotoxy(chopperX, chopperY + 1);

}

}

void eraseChopper()

{

gotoxy(chopperX, chopperY);

for (int i = 0; i < 4; i++)

{

cout << " ";

}

gotoxy(chopperX, chopperY + 1);

for (int i = 0; i < 4; i++)

{

cout << " ";

}

}

void moveChopperLeft()

{

char next = getCharAtxy(chopperX - 1, chopperY);

char next1 = getCharAtxy(chopperX - 1, chopperY + 1);

if (next == ' ' && next1 == ' ')

{

eraseChopper();

chopperX = chopperX - 1;

printChopper();

}

}

void moveChopperRight()

{

char next = getCharAtxy(chopperX + 4, chopperY);

char next1 = getCharAtxy(chopperX + 4, chopperY + 1);

if (next == ' ' && next1 == ' ')

{

eraseChopper();

chopperX = chopperX + 1;

printChopper();

}

}

void moveChopperUp()

{

char next = getCharAtxy(chopperX, chopperY - 1);

char next1 = getCharAtxy(chopperX + 1, chopperY - 1);

char next2 = getCharAtxy(chopperX + 2, chopperY - 1);

char next3 = getCharAtxy(chopperX + 3, chopperY - 1);

if (next == ' ' && next1 == ' ' && next2 == ' ' && next3 == ' ')

{

eraseChopper();

chopperY = chopperY - 1;

printChopper();

}

}

void moveChopperDown()

{

char next = getCharAtxy(chopperX, chopperY + 2);

char next1 = getCharAtxy(chopperX + 1, chopperY + 2);

char next2 = getCharAtxy(chopperX + 2, chopperY + 2);

char next3 = getCharAtxy(chopperX + 3, chopperY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ' && next3 == ' ')

{

eraseChopper();

chopperY = chopperY + 1;

printChopper();

}

}

void printEnemyV()

{

SetConsoleTextAttribute(hConsole, 12);

gotoxy(enemyVX, enemyVY);

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 4; col++)

{

cout << enemyV[row][col];

}

gotoxy(enemyVX, enemyVY + 1);

}

}

void eraseEnemyV()

{

gotoxy(enemyVX, enemyVY);

for (int i = 0; i < 4; i++)

{

cout << " ";

}

gotoxy(enemyVX, enemyVY + 1);

for (int i = 0; i < 4; i++)

{

cout << " ";

}

}

void moveEnemyV()

{

if (enemyVDirection == "Up")

{

char next = getCharAtxy(enemyVX, enemyVY - 1);

char next1 = getCharAtxy(enemyVX + 1, enemyVY - 1);

char next2 = getCharAtxy(enemyVX + 2, enemyVY - 1);

char next3 = getCharAtxy(enemyVX + 3, enemyVY - 1);

if (next == ' ' && next1 == ' ' && next2 == ' ' && next3 == ' ')

{

eraseEnemyV();

enemyVY--;

printEnemyV();

}

if (next == '\_')

{

enemyVDirection = "Down";

}

}

if (enemyVDirection == "Down")

{

char next = getCharAtxy(enemyVX, enemyVY + 2);

char next1 = getCharAtxy(enemyVX + 1, enemyVY + 2);

char next2 = getCharAtxy(enemyVX + 2, enemyVY + 2);

char next3 = getCharAtxy(enemyVX + 3, enemyVY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ' && next3 == ' ')

{

eraseEnemyV();

enemyVY++;

printEnemyV();

}

if (next == '\_')

{

enemyVDirection = "Up";

}

}

}

void printEnemyH()

{

SetConsoleTextAttribute(hConsole, 12);

gotoxy(enemyHX, enemyHY);

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

{

for (int col = 0; col < 2; col++)

{

cout << enemyH[row][col];

}

gotoxy(enemyHX, enemyHY + row + 1);

}

}

void eraseEnemyH()

{

gotoxy(enemyHX, enemyHY);

for (int i = 0; i < 2; i++)

{

cout << " ";

}

gotoxy(enemyHX, enemyHY + 1);

for (int i = 0; i < 2; i++)

{

cout << " ";

}

gotoxy(enemyHX, enemyHY + 2);

for (int i = 0; i < 2; i++)

{

cout << " ";

}

}

void moveEnemyH()

{

if (enemyHDirection == "Left")

{

char next = getCharAtxy(enemyHX - 1, enemyHY);

char next1 = getCharAtxy(enemyHX - 1, enemyHY + 1);

char next2 = getCharAtxy(enemyHX - 1, enemyHY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemyH();

enemyHX--;

printEnemyH();

}

if (next == '|')

{

enemyHDirection = "Right";

}

}

if (enemyHDirection == "Right")

{

char next = getCharAtxy(enemyHX + 2, enemyHY);

char next1 = getCharAtxy(enemyHX + 2, enemyHY + 1);

char next2 = getCharAtxy(enemyHX + 2, enemyHY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemyH();

enemyHX++;

printEnemyH();

}

if (next == '|')

{

enemyHDirection = "Left";

}

}

}

void printEnemy()

{

SetConsoleTextAttribute(hConsole, 12);

gotoxy(enemyX, enemyY);

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

{

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

{

cout << enemy[row][col];

}

gotoxy(enemyX, enemyY + row + 1);

}

}

void eraseEnemy()

{

gotoxy(enemyX, enemyY);

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

{

cout << " ";

}

gotoxy(enemyX, enemyY + 1);

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

{

cout << " ";

}

gotoxy(enemyX, enemyY + 2);

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

{

cout << " ";

}

}

void moveEnemy(int x, int y)

{

char next;

char next1;

char next2;

if (x > enemyX && y > enemyY)

{

next = getCharAtxy(enemyX, enemyY + 3);

next1 = getCharAtxy(enemyX + 1, enemyY + 3);

next2 = getCharAtxy(enemyX + 2, enemyY + 3);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyY++;

printEnemy();

}

next = getCharAtxy(enemyX + 3, enemyY);

next1 = getCharAtxy(enemyX + 3, enemyY + 1);

next2 = getCharAtxy(enemyX + 3, enemyY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyX++;

printEnemy();

}

}

else if (x == enemyX && y > enemyY)

{

next = getCharAtxy(enemyX, enemyY + 3);

next1 = getCharAtxy(enemyX + 1, enemyY + 3);

next2 = getCharAtxy(enemyX + 2, enemyY + 3);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyY++;

printEnemy();

}

}

else if (x < enemyX && y > enemyY)

{

next = getCharAtxy(enemyX, enemyY + 3);

next1 = getCharAtxy(enemyX + 1, enemyY + 3);

next2 = getCharAtxy(enemyX + 2, enemyY + 3);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyY++;

printEnemy();

}

next = getCharAtxy(enemyX - 1, enemyY);

next1 = getCharAtxy(enemyX - 1, enemyY + 1);

next2 = getCharAtxy(enemyX - 1, enemyY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyX--;

printEnemy();

}

}

else if (x > enemyX && y == enemyY)

{

next = getCharAtxy(enemyX + 3, enemyY);

next1 = getCharAtxy(enemyX + 3, enemyY + 1);

next2 = getCharAtxy(enemyX + 3, enemyY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyX++;

printEnemy();

}

}

else if (x < enemyX && y == enemyY)

{

next = getCharAtxy(enemyX - 1, enemyY);

next1 = getCharAtxy(enemyX - 1, enemyY + 1);

next2 = getCharAtxy(enemyX - 1, enemyY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyX--;

printEnemy();

}

}

else if (x < enemyX && y < enemyY)

{

next = getCharAtxy(enemyX, enemyY - 1);

next1 = getCharAtxy(enemyX + 1, enemyY - 1);

next2 = getCharAtxy(enemyX + 2, enemyY - 1);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyY--;

printEnemy();

}

next = getCharAtxy(enemyX - 1, enemyY);

next1 = getCharAtxy(enemyX - 1, enemyY + 1);

next2 = getCharAtxy(enemyX - 1, enemyY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyX--;

printEnemy();

}

}

else if (x > enemyX && y < enemyY)

{

next = getCharAtxy(enemyX, enemyY - 1);

next1 = getCharAtxy(enemyX + 1, enemyY - 1);

next2 = getCharAtxy(enemyX + 2, enemyY - 1);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyY--;

printEnemy();

}

next = getCharAtxy(enemyX + 3, enemyY);

next1 = getCharAtxy(enemyX + 3, enemyY + 1);

next2 = getCharAtxy(enemyX + 3, enemyY + 2);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyX++;

printEnemy();

}

}

else

{

next = getCharAtxy(enemyX, enemyY - 1);

next1 = getCharAtxy(enemyX + 1, enemyY - 1);

next2 = getCharAtxy(enemyX + 2, enemyY - 1);

if (next == ' ' && next1 == ' ' && next2 == ' ')

{

eraseEnemy();

enemyY--;

printEnemy();

}

}

}

void generateBulletRight()

{

if (getCharAtxy(chopperX + 4, chopperY) == ' ')

{

bulletXRight[bulletCountRight] = chopperX + 4;

bulletYRight[bulletCountRight] = chopperY;

isbulletActiveRight[bulletCountRight] = true;

gotoxy(chopperX + 4, chopperY);

cout << ".";

bulletCountRight++;

}

}

void generateBulletLeft()

{

if (getCharAtxy(chopperX - 1, chopperY) == ' ')

{

bulletXLeft[bulletCountLeft] = chopperX - 1;

bulletYLeft[bulletCountLeft] = chopperY;

isbulletActiveLeft[bulletCountLeft] = true;

gotoxy(chopperX - 1, chopperY);

cout << ".";

bulletCountLeft++;

}

}

void generateBulletUp()

{

if (getCharAtxy(chopperX + 1, chopperY - 1) == ' ')

{

bulletXUp[bulletCountUp] = chopperX + 1;

bulletYUp[bulletCountUp] = chopperY - 1;

isbulletActiveUp[bulletCountUp] = true;

gotoxy(chopperX + 1, chopperY - 1);

cout << ".";

bulletCountUp++;

}

}

void generateBulletDown()

{

if (getCharAtxy(chopperX + 1, chopperY + 2) == ' ')

{

bulletXDown[bulletCountDown] = chopperX + 1;

bulletYDown[bulletCountDown] = chopperY + 2;

isbulletActiveDown[bulletCountDown] = true;

gotoxy(chopperX + 1, chopperY + 2);

cout << ".";

bulletCountDown++;

}

}

void generateEnemyBulletRight()

{

if (getCharAtxy(enemyX + 3, enemyY + 1) == ' ')

{

EnemybulletXRight[EnemybulletCountRight] = enemyX + 3;

EnemybulletYRight[EnemybulletCountRight] = enemyY + 1;

isEnemybulletActiveRight[EnemybulletCountRight] = true;

gotoxy(enemyX + 3, enemyY + 1);

cout << ".";

EnemybulletCountRight++;

}

}

void generateEnemyBulletLeft()

{

if (getCharAtxy(enemyX - 1, enemyY + 1) == ' ')

{

EnemybulletXLeft[EnemybulletCountLeft] = enemyX - 1;

EnemybulletYLeft[EnemybulletCountLeft] = enemyY + 1;

isEnemybulletActiveLeft[EnemybulletCountLeft] = true;

gotoxy(enemyX - 1, enemyY + 1);

cout << ".";

EnemybulletCountLeft++;

}

}

void generateEnemyBulletUp()

{

if (getCharAtxy(enemyX + 1, enemyY - 1) == ' ')

{

EnemybulletXUp[EnemybulletCountUp] = enemyX + 1;

EnemybulletYUp[EnemybulletCountUp] = enemyY - 1;

isEnemybulletActiveUp[EnemybulletCountUp] = true;

gotoxy(enemyX + 1, enemyY - 1);

cout << ".";

EnemybulletCountUp++;

}

}

void generateEnemyBulletDown()

{

if (getCharAtxy(enemyX + 1, enemyY + 3) == ' ')

{

EnemybulletXDown[EnemybulletCountDown] = enemyX + 1;

EnemybulletYDown[EnemybulletCountDown] = enemyY + 3;

isEnemybulletActiveDown[EnemybulletCountDown] = true;

gotoxy(enemyX + 1, enemyY + 3);

cout << ".";

EnemybulletCountDown++;

}

}

void moveBulletRight()

{

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

{

if (isbulletActiveRight[i] == true)

{

char next = getCharAtxy(bulletXRight[i] + 1, bulletYRight[i]);

if (next != ' ')

{

eraseBullet(bulletXRight[i], bulletYRight[i]);

isbulletActiveRight[i] = false;

;

}

else

{

eraseBullet(bulletXRight[i], bulletYRight[i]);

bulletXRight[i] = bulletXRight[i] + 1;

printBullet(bulletXRight[i], bulletYRight[i]);

}

}

}

}

void moveBulletLeft()

{

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

{

if (isbulletActiveLeft[i] == true)

{

char next = getCharAtxy(bulletXLeft[i] - 1, bulletYLeft[i]);

if (next != ' ')

{

eraseBullet(bulletXLeft[i], bulletYLeft[i]);

isbulletActiveLeft[i] = false;

;

}

else

{

eraseBullet(bulletXLeft[i], bulletYLeft[i]);

bulletXLeft[i] = bulletXLeft[i] - 1;

printBullet(bulletXLeft[i], bulletYLeft[i]);

}

}

}

}

void moveBulletUp()

{

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

{

if (isbulletActiveUp[i] == true)

{

char next = getCharAtxy(bulletXUp[i], bulletYUp[i] - 1);

if (next != ' ')

{

eraseBullet(bulletXUp[i], bulletYUp[i]);

isbulletActiveUp[i] = false;

;

}

else

{

eraseBullet(bulletXUp[i], bulletYUp[i]);

bulletYUp[i] = bulletYUp[i] - 1;

printBullet(bulletXUp[i], bulletYUp[i]);

}

}

}

}

void moveBulletDown()

{

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

{

if (isbulletActiveDown[i] == true)

{

char next = getCharAtxy(bulletXDown[i], bulletYDown[i] + 1);

if (next != ' ')

{

eraseBullet(bulletXDown[i], bulletYDown[i]);

isbulletActiveDown[i] = false;

}

else

{

eraseBullet(bulletXDown[i], bulletYDown[i]);

bulletYDown[i] = bulletYDown[i] + 1;

printBullet(bulletXDown[i], bulletYDown[i]);

}

}

}

}

void moveEnemyBulletRight()

{

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

{

if (isEnemybulletActiveRight[i] == true)

{

char next = getCharAtxy(EnemybulletXRight[i] + 1, EnemybulletYRight[i]);

if (next != ' ')

{

eraseBullet(EnemybulletXRight[i], EnemybulletYRight[i]);

isEnemybulletActiveRight[i] = false;

;

}

else

{

eraseBullet(EnemybulletXRight[i], EnemybulletYRight[i]);

EnemybulletXRight[i] = EnemybulletXRight[i] + 1;

printBullet(EnemybulletXRight[i], EnemybulletYRight[i]);

}

}

}

}

void moveEnemyBulletLeft()

{

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

{

if (isEnemybulletActiveLeft[i] == true)

{

char next = getCharAtxy(EnemybulletXLeft[i] - 1, EnemybulletYLeft[i]);

if (next != ' ')

{

eraseBullet(EnemybulletXLeft[i], EnemybulletYLeft[i]);

isEnemybulletActiveLeft[i] = false;

;

}

else

{

eraseBullet(EnemybulletXLeft[i], EnemybulletYLeft[i]);

EnemybulletXLeft[i] = EnemybulletXLeft[i] - 1;

printBullet(EnemybulletXLeft[i], EnemybulletYLeft[i]);

}

}

}

}

void moveEnemyBulletUp()

{

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

{

if (isEnemybulletActiveUp[i] == true)

{

char next = getCharAtxy(EnemybulletXUp[i], EnemybulletYUp[i] - 1);

if (next != ' ')

{

eraseBullet(EnemybulletXUp[i], EnemybulletYUp[i]);

isEnemybulletActiveUp[i] = false;

;

}

else

{

eraseBullet(EnemybulletXUp[i], EnemybulletYUp[i]);

EnemybulletYUp[i] = EnemybulletYUp[i] - 1;

printBullet(EnemybulletXUp[i], EnemybulletYUp[i]);

}

}

}

}

void moveEnemyBulletDown()

{

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

{

if (isEnemybulletActiveDown[i] == true)

{

char next = getCharAtxy(EnemybulletXDown[i], EnemybulletYDown[i] + 1);

if (next != ' ')

{

eraseBullet(EnemybulletXDown[i], EnemybulletYDown[i]);

isEnemybulletActiveDown[i] = false;

}

else

{

eraseBullet(EnemybulletXDown[i], EnemybulletYDown[i]);

EnemybulletYDown[i] = EnemybulletYDown[i] + 1;

printBullet(EnemybulletXDown[i], EnemybulletYDown[i]);

}

}

}

}

void printBullet(int x, int y)

{

gotoxy(x, y);

cout << '.';

}

void eraseBullet(int x, int y)

{

gotoxy(x, y);

cout << " ";

}

void generateEnemyVBullet()

{

if (getCharAtxy(enemyVX - 1, enemyVY) == ' ')

{

bulletVX[bulletCountV] = enemyVX - 1;

bulletVY[bulletCountV] = enemyVY;

isVbulletActive[bulletCountV] = true;

gotoxy(enemyVX - 1, enemyVY);

cout << ".";

bulletCountV++;

}

}

void moveBulletV()

{

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

{

if (isVbulletActive[i] == true)

{

char next = getCharAtxy(bulletVX[i] - 1, bulletVY[i]);

if (next != ' ')

{

eraseBullet(bulletVX[i], bulletVY[i]);

isVbulletActive[i] = false;

;

}

else

{

eraseBullet(bulletVX[i], bulletVY[i]);

bulletVX[i] = bulletVX[i] - 1;

printBullet(bulletVX[i], bulletVY[i]);

}

}

}

}

void generateEnemyHBullet()

{

if (getCharAtxy(enemyHX, enemyHY + 3) == ' ')

{

bulletHX[bulletCountH] = enemyHX;

bulletHY[bulletCountH] = enemyHY + 3;

isHbulletActive[bulletCountH] = true;

gotoxy(enemyHX, enemyHY + 3);

cout << ".";

bulletCountH++;

}

}

void moveBulletH()

{

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

{

if (isHbulletActive[i] == true)

{

char next = getCharAtxy(bulletHX[i], bulletHY[i] + 1);

if (next != ' ')

{

eraseBullet(bulletHX[i], bulletHY[i]);

isHbulletActive[i] = false;

;

}

else

{

eraseBullet(bulletHX[i], bulletHY[i]);

bulletHY[i] = bulletHY[i] + 1;

printBullet(bulletHX[i], bulletHY[i]);

}

}

}

}

void bulletCollisionWithEnemyRight()

{

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

{

if (isbulletActiveRight[i] == true)

{

if (bulletXRight[i] == enemyX - 1 && (bulletYRight[i] == enemyY || bulletYRight[i] == enemyY + 1 || bulletYRight[i] == enemyY + 2))

{

addScore();

changeEnemyHealth();

}

}

}

}

void bulletCollisionWithEnemyLeft()

{

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

{

if (isbulletActiveLeft[i] == true)

{

if (bulletXLeft[i] == enemyX + 3 && (bulletYLeft[i] == enemyY || bulletYLeft[i] == enemyY + 1 || bulletYLeft[i] == enemyY + 2))

{

addScore();

changeEnemyHealth();

}

}

}

}

void bulletCollisionWithEnemyUp()

{

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

{

if (isbulletActiveUp[i] == true)

{

if (bulletYUp[i] == enemyY + 3 && (bulletXUp[i] == enemyX || bulletXUp[i] == enemyX + 1 || bulletXUp[i] == enemyX + 2))

{

addScore();

changeEnemyHealth();

}

}

}

}

void bulletCollisionWithEnemyDown()

{

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

{

if (isbulletActiveDown[i] == true)

{

if (bulletYDown[i] == enemyY - 1 && (bulletXDown[i] == enemyX || bulletXDown[i] == enemyX + 1 || bulletXDown[i] == enemyX + 2))

{

addScore();

changeEnemyHealth();

}

}

}

}

void bulletCollisionWithEnemyVRight()

{

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

{

if (isbulletActiveRight[i] == true)

{

if (bulletXRight[i] == enemyVX - 1 && (bulletYRight[i] == enemyVY || bulletYRight[i] == enemyVY + 1))

{

addScore();

changeEnemyVHealth();

}

}

}

}

void bulletCollisionWithEnemyVUp()

{

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

{

if (isbulletActiveUp[i] == true)

{

if (bulletYUp[i] == enemyVY + 2 && (bulletXUp[i] == enemyVX || bulletXUp[i] == enemyVX + 1 || bulletXUp[i] == enemyVX + 2 || bulletXUp[i] == enemyVX + 3))

{

addScore();

changeEnemyVHealth();

}

}

}

}

void bulletCollisionWithEnemyVDown()

{

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

{

if (isbulletActiveDown[i] == true)

{

if (bulletYDown[i] == enemyVY - 1 && (bulletXDown[i] == enemyVX || bulletXDown[i] == enemyVX + 1 || bulletXDown[i] == enemyVX + 2 || bulletXDown[i] == enemyVX + 3))

{

addScore();

changeEnemyVHealth();

}

}

}

}

void bulletCollisionWithEnemyHRight()

{

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

{

if (isbulletActiveRight[i] == true)

{

if (bulletXRight[i] == enemyHX - 1 && (bulletYRight[i] == enemyHY || bulletYRight[i] == enemyHY + 1 || bulletYRight[i] == enemyHY + 2))

{

addScore();

changeEnemyHHealth();

}

}

}

}

void bulletCollisionWithEnemyHLeft()

{

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

{

if (isbulletActiveLeft[i] == true)

{

if (bulletXLeft[i] == enemyHX + 2 && (bulletYLeft[i] == enemyHY || bulletYLeft[i] == enemyHY + 1 || bulletYLeft[i] == enemyHY + 2))

{

addScore();

changeEnemyHHealth();

}

}

}

}

void bulletCollisionWithEnemyHUp()

{

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

{

if (isbulletActiveUp[i] == true)

{

if (bulletYUp[i] == enemyHY + 3 && (bulletXUp[i] == enemyHX || bulletXUp[i] == enemyHX + 1))

{

addScore();

changeEnemyHHealth();

}

}

}

}

void addScore()

{

score = score + 10;

}

void printScore()

{

gotoxy(10, 5);

SetConsoleTextAttribute(hConsole, 8);

cout << "Score: " << score;

}

void changeEnemyHealth()

{

hCountEnemy--;

if (hCountEnemy % 10 == 0)

{

enemyHealth[EH] = ' ';

EH--;

}

gotoxy(75, 4);

SetConsoleTextAttribute(hConsole, 11);

cout << enemyHealth;

}

void printEnemyHealth()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(70, 3);

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

{

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

{

cout << enemy[row][col];

}

gotoxy(70, 4 + row);

}

gotoxy(74, 4);

cout << " " << enemyHealth;

}

void changeEnemyVHealth()

{

hCountEnemyV--;

if (hCountEnemyV % 6 == 0)

{

VenemyHealth[EVH] = ' ';

EVH--;

}

gotoxy(95, 4);

SetConsoleTextAttribute(hConsole, 11);

cout << VenemyHealth;

}

void printEnemyVHealth()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(90, 3);

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 4; col++)

{

cout << enemyV[row][col];

}

gotoxy(90, 4);

}

gotoxy(95, 4);

cout << VenemyHealth;

}

void changeEnemyHHealth()

{

hCountEnemyH--;

if (hCountEnemyH % 6 == 0)

{

HenemyHealth[EHH] = ' ';

EHH--;

}

gotoxy(113, 4);

SetConsoleTextAttribute(hConsole, 11);

cout << HenemyHealth;

}

void printEnemyHHealth()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(110, 2);

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

{

for (int col = 0; col < 2; col++)

{

cout << enemyH[row][col];

}

gotoxy(110, 2 + row + 1);

}

gotoxy(112, 4);

cout << " " << HenemyHealth;

}

void changeChopperHealth()

{

hCountChopper--;

if (hCountChopper % 4 == 0)

{

ChopperHealth[CH] = ' ';

CH--;

}

gotoxy(133, 4);

SetConsoleTextAttribute(hConsole, 11);

cout << ChopperHealth;

}

void printChopperHealth()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(128, 3);

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 4; col++)

{

cout << chopper[row][col];

}

gotoxy(128, 4);

}

gotoxy(132, 4);

cout << " " << ChopperHealth;

}

bool ChopperCollisionWithEnemy(int CX, int CY, int EX, int EY)

{

bool flag;

// when chopper is at the right of enemy

if (EX + 3 == CX && (EY == CY || EY == CY + 1 || EY == CY - 1 || EY == CY - 2))

flag = true;

// when chopper is at the left of enemy

else if (EX - 4 == CX && (EY == CY || EY == CY + 1 || EY == CY - 1 || EY == CY - 2))

flag = true;

// when chopper is up of enemy

else if (EY - 2 == CY && (EX == CX || EX == CX + 1 || EX == CX + 2 || EX == CX + 3 || EX == CX - 1 || EX == CX - 2))

flag = true;

// when chopper is down of enemy

else if (EY + 3 == CY && (EX == CX || EX == CX + 1 || EX == CX + 2 || EX == CX + 3 || EX == CX - 1 || EX == CX - 2))

flag = true;

else

flag = false;

return flag;

}

bool ChopperCollisionWithEnemyH(int CX, int CY, int EX, int EY)

{

bool flag;

// when chopper is at the right of enemyH

if (EX + 2 == CX && (EY == CY || EY == CY + 1 || EY == CY - 1 || EY == CY - 2))

flag = true;

// when chopper is at the left of enemyH

else if (EX - 4 == CX && (EY == CY || EY == CY + 1 || EY == CY - 1 || EY == CY - 2))

flag = true;

// when chopper is down of enemyH

else if (EY + 3 == CY && (EX == CX || EX == CX + 1 || EX == CX + 2 || EX == CX + 3 || EX == CX - 1))

flag = true;

else

flag = false;

return flag;

}

bool ChopperCollisionWithEnemyV(int CX, int CY, int EX, int EY)

{

bool flag;

// when chopper is at the left of enemyV

if (EX - 4 == CX && (EY == CY || EY == CY + 1 || EY == CY - 1))

flag = true;

// when chopper is up of enemyV

else if (EY - 2 == CY && (EX == CX || EX == CX + 1 || EX == CX + 2 || EX == CX + 3 || EX == CX - 1 || EX == CX - 2))

flag = true;

// when chopper is down of enemyV

else if (EY + 2 == CY && (EX == CX || EX == CX + 1 || EX == CX + 2 || EX == CX + 3 || EX == CX - 1 || EX == CX - 2))

flag = true;

else

flag = false;

return flag;

}

void bulletVCollisionWithChopper()

{

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

{

if (isVbulletActive[i] == true)

{

if (bulletVX[i] == chopperX + 4 && (bulletVY[i] == chopperY || bulletVY[i] == chopperY + 1))

{

changeChopperHealth();

}

}

}

}

void bulletHCollisionWithChopper()

{

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

{

if (isHbulletActive[i] == true)

{

if (bulletHY[i] == chopperY - 1 && (bulletHX[i] == chopperX || bulletHX[i] == chopperX + 1 || bulletHX[i] == chopperX + 2 || bulletHX[i] == chopperX + 3))

{

changeChopperHealth();

}

}

}

}

void EnemyBulletsCollisionWithChopper()

{

EnemyBulletsCollisionWithChopperUp();

EnemyBulletsCollisionWithChopperDowwn();

EnemyBulletsCollisionWithChopperLeft();

EnemyBulletsCollisionWithChopperRight();

}

void EnemyBulletsCollisionWithChopperUp()

{

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

{

if (isEnemybulletActiveUp[i] == true)

{

if (EnemybulletYUp[i] == chopperY + 2 && (EnemybulletXUp[i] == chopperX || EnemybulletXUp[i] == chopperX + 1 || EnemybulletXUp[i] == chopperX + 2 || EnemybulletXUp[i] == chopperX + 3))

{

changeChopperHealth();

}

}

}

}

void EnemyBulletsCollisionWithChopperDowwn()

{

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

{

if (isEnemybulletActiveDown[i] == true)

{

if (EnemybulletYDown[i] == chopperY - 1 && (EnemybulletXDown[i] == chopperX || EnemybulletXDown[i] == chopperX + 1 || EnemybulletXDown[i] == chopperX + 2 || EnemybulletXDown[i] == chopperX + 3))

{

changeChopperHealth();

}

}

}

}

void EnemyBulletsCollisionWithChopperLeft()

{

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

{

if (isEnemybulletActiveLeft[i] == true)

{

if (EnemybulletXLeft[i] == chopperX + 4 && (EnemybulletYLeft[i] == chopperY || EnemybulletYLeft[i] == chopperY + 1))

{

changeChopperHealth();

}

}

}

}

void EnemyBulletsCollisionWithChopperRight()

{

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

{

if (isEnemybulletActiveRight[i] == true)

{

if (EnemybulletXRight[i] == chopperX - 1 && (EnemybulletYRight[i] == chopperY || EnemybulletYRight[i] == chopperY + 1))

{

changeChopperHealth();

}

}

}

}

void printLives()

{

gotoxy(145, 4);

cout << "Lives Remaining : " << lives;

}

bool menuValidity(string option)

{

bool flag;

if (option == "1" || option == "2" || option == "3" || option == "4")

flag = true;

return flag;

}

void menu()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(5, 10);

cout << "MENU";

gotoxy(0, 11);

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

gotoxy(5, 12);

cout << "1.Start New Game";

gotoxy(5, 13);

cout << "2.Continue";

gotoxy(5, 14);

cout << "3.options";

gotoxy(5, 15);

cout << "4.Exit";

}

void Options()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(5, 10);

cout << "MENU";

gotoxy(0, 11);

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

gotoxy(5, 12);

cout << "1.Keys";

gotoxy(5, 13);

cout << "2.Information";

gotoxy(5, 14);

cout << "3.Return";

}

void keys()

{

SetConsoleTextAttribute(hConsole, 11);

gotoxy(5, 10);

cout << "Keys";

gotoxy(0, 11);

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

gotoxy(5, 12);

cout << "1.UP Go Up";

gotoxy(5, 13);

cout << "2.DOWN Go Down";

gotoxy(5, 14);

cout << "3.LEFT Go Left";

gotoxy(5, 15);

cout << "4.RIGHT Go Right";

gotoxy(5, 16);

cout << "5.NUM 8 Fire UP";

gotoxy(5, 17);

cout << "6.NUM 5 Fire Down";

gotoxy(5, 18);

cout << "7.NUM 6 Fire Right";

gotoxy(5, 19);

cout << "8.NUM 4 Fire Left";

gotoxy(5, 20);

cout << "9.Esc To return to Main Menu";

gotoxy(5, 21);

cout << "Press any key to continue . . .";

}

void instructions()

{

system("cls");

header();

SetConsoleTextAttribute(hConsole, 11);

gotoxy(5, 10);

cout << "MENU";

gotoxy(5, 11);

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

// Chopper

gotoxy(5, 13);

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 4; col++)

{

cout << chopper[row][col];

}

gotoxy(5, 14);

}

gotoxy(30, 14);

cout << "Chopper : It is the main protagonist whose duty is to eliminate all enimies";

// Enemy

gotoxy(5, 16);

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

{

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

{

cout << enemy[row][col];

}

gotoxy(5, 17 + row);

}

gotoxy(30, 17);

cout << "Helicopter : This enemy follows the chopper all around the maze and probably the biggest threat to chopper";

// EnemyV

gotoxy(5, 20);

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 4; col++)

{

cout << enemyV[row][col];

}

gotoxy(5, 21);

}

gotoxy(30, 21);

cout << "Plane : This enemy moves vertically and fires horizontally";

// EnemyH

gotoxy(5, 23);

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

{

for (int col = 0; col < 2; col++)

{

cout << enemyH[row][col];

}

gotoxy(5, 23 + row + 1);

}

gotoxy(30, 24);

cout << "Jet : This enemy moves horizontally and fires vertically";

gotoxy(5, 27);

cout << "Enter any key to Continue. . .";

}

void maze()

{

system("cls");

gotoxy(0, 6);

SetConsoleTextAttribute(hConsole, 8);

for (int row = 0; row < 35; row++)

{

cout << grid[row][0] << endl;

}

}

void mainScreen()

{

header();

cout << " (jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj)(jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj) " << 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 << " Press any Key to Continue... " << endl;

}

void header()

{

system("cls");

SetConsoleTextAttribute(hConsole, 3);

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

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

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

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

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

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

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

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

}

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 : ' ';

}

void saving()

{

fstream file;

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

file << chopperX << "," << chopperY << "," << enemyX << "," << enemyY << "," << enemyHX << "," << enemyHY

<< "," << enemyVX << "," << enemyVY << "," << score << "," << lives << "," << hCountEnemy << ","

<< hCountEnemyV << "," << hCountEnemyH << "," << hCountChopper << "," << EH << "," << EHH << ","

<< EVH << "," << CH << ",";

file.close();

}

void loading()

{

fstream file;

string word;

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

while (!file.eof())

{

getline(file, word);

chopperX = atoi(parsing(word, 1).c\_str());

chopperY = atoi(parsing(word, 2).c\_str());

enemyX = atoi(parsing(word, 3).c\_str());

enemyY = atoi(parsing(word, 4).c\_str());

enemyHX = atoi(parsing(word, 5).c\_str());

enemyHY = atoi(parsing(word, 6).c\_str());

enemyVX = atoi(parsing(word, 7).c\_str());

enemyVY = atoi(parsing(word, 8).c\_str());

score = atoi(parsing(word, 9).c\_str());

hCountEnemy = atoi(parsing(word, 11).c\_str());

hCountEnemyV = atoi(parsing(word, 12).c\_str());

hCountEnemyH = atoi(parsing(word, 13).c\_str());

hCountChopper = atoi(parsing(word, 14).c\_str());

EH = atoi(parsing(word, 15).c\_str());

EHH = atoi(parsing(word, 16).c\_str());

EVH = atoi(parsing(word, 17).c\_str());

CH = atoi(parsing(word, 18).c\_str());

}

file.close();

}

void loadingLives()

{

fstream file;

string word;

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

while (!file.eof())

{

getline(file, word);

lives = atoi(parsing(word, 10).c\_str());

}

file.close();

}

string parsing(string word, int location)

{

int commaCount = 1;

string item = "";

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

{

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

{

commaCount = commaCount + 1;

}

else if (commaCount == location)

{

item = item + word[x];

}

}

return item;

}