TANK DEFENDER

GROUP 13

IM/2023/110 IM/2023/111 IM/2023/031 IM/2023/032

Department of Industrial Management

INTE 11223 Programming Concepts

Prof. Janaka Wijayanayaka

14.07.2025

Abstract

This report outlines the development of *Tank Defender*, a 2D shooting game in which the player controls a tank to shoot at enemy suicide tanks, including light and heavy types. Inspired by classic arcade shooter games, the project aims to recreate fast-paced, engaging action within a simple framework. The game features real-time keyboard input, enemy generation, collision detection, and a scoring system, all implemented using C++ and Windows console functions. Through this project, the team gained practical experience in game logic design, user interaction, and managing gameplay flow in a low-graphics environment.

Table of Contents

Introduction

The development of Tank Defender was undertaken as a practical project to explore the fundamentals of game design and real-time user interaction using C++. The game Tank Defender is an adaptation of several games that have similar patterns. Our game resembles a console version of a classic vertical scrolling tank defense shooter, with inspiration from *Space Invaders* and *Battle City.*

In this game, the player's main objective is to survive as long as possible by shooting down enemy tanks while avoiding incoming enemy tanks. The player earns points based on the number of enemy tanks destroyed. The game includes a health point system, starting with 100 points, which decreases each time the player is hit by enemy fire. The game ends when the player's health reaches zero.

The purpose of developing this game is to help us understand the implementation of core programming concepts in C++, such as keyboard input handling, screen rendering, enemy generation, and collision detection. It also gave us hands-on experience with structuring game logic, managing real-time gameplay elements, and improving console-based user interaction.

Through this report, we will discuss the techniques used in developing the game, provide a user guide on how to play, reflect on the challenges we faced during development, and suggest possible future improvements.

Technical documentation

Libraries used

```
#include <iostream>
#include <conio.h>
#include <windows.h>
```

- iostream: Used for input and output operations.
- conio.h: Handles keyboard input with getch() and _kbhit() functions.
- windows.h: Provides access to Windows API for cursor positioning, console manipulation, and sleep control.

Global Constants

```
#define screen_width 95 // Width of the console screen
#define screen_height 30 // Height of the console screen
#define playspace_width 70 // Width of the game play space

Enemy enemies[MAX_ENEMIES]; // Array to hold enemies

Bullet playerBullets[MAX_BULLETS]; // Support multiple bullets
const int MAX_ROAD_ROWS = screen_height - 2;
char roadPattern[MAX_ROAD_ROWS]; // stores line pattern per row
const int patternDurationFrames = 15; // Number of frames a pattern runs (~1.5 seconds if Sleep(100))
const int totalPatterns = 3; // number of patterns you have
```

- screen_width, screen_height: Define dimensions of the game screen.
- playspace_width: Width of the active gameplay area.
- MAX_ENEMIES: Maximum number of enemy vehicles active on screen at once.
- MAX_BULLETS: Maximum number of bullets the player can shoot simultaneously.
- MAX_ROAD_ROWS: Rows used for moving road animation.
- patternDurationFrames, totalPatterns: Constants used to control road line pattern switching.

Global Variables

```
int status =1 ; // Variable to control the game loop
int tankPos=playspace_width/2; // Variable to keep track of the tank position
char tank[4][7] = {
    {' ', ' ', ' ', char(219), ' ', ' ', ' '},
    {' ', ' ', char(219), char(219), char(219), ' ', ' '},     // Top cannon
    {char(219), char(219), char(219), char(219), char(219), char(219), char(219)}, // Body
    {char(219), ' ', char(219), ' ', char(219), ' ', char(219)}  // Wheels
}; // shape of the tank
int score = 0; // Variable to keep track of the score
float Health = 100.0; // Variable to keep track of the number of lives
const int MAX_ENEMIES = 3; // Maximum number of enemies
const int MAX_BULLETS = 5; // Maximum number of bullets
struct Enemy {
    int x;
    int y;
    bool isActive;
    char type; // 'L' = light, 'H' = heavy
    int health; // track how many hits it can take
};


Enemy enemies[MAX_ENEMIES]; // Array to hold enemies
char roadPattern[MAX_ROAD_ROWS]; // stores line pattern per row
// For controlling pattern switching timing
int bulletBuffer =0;
bool isReloading = false;
int reloadCounter = rp; // means that wait for rp/10 seconds to reload
```

- HANDLE console, COORD CursorPosition: Used to control cursor movement and visibility in the console.
- int tankPos: Current X-position of the player's tank.
- float Health: Tracks the player's remaining health.
- int score: Stores the player's score.
- Enemy enemies[MAX_ENEMIES]: Stores enemy attributes like position, type (light or heavy), health, and status.
- Bullet playerBullets[MAX_BULLETS]: Manages the player's active bullets.
- char tank[4][7]: 2D ASCII art representation of the player's tank.
- char roadPattern[MAX_ROAD_ROWS]: Stores patterns for animated road lines.
- int bulletBuffer, isReloading, reloadCounter: Manage bullet firing and gun reload cooldown.

Key Functions

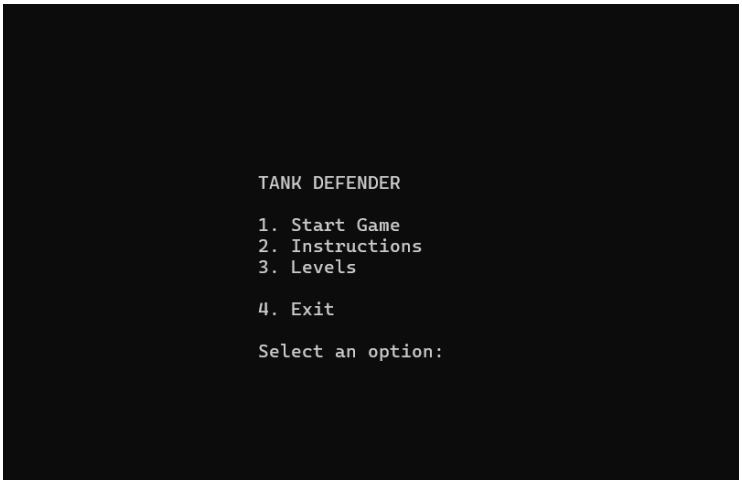
- main() – The entry point of the game. Displays the welcome screen and navigates the main menu.
- welcome() – Displays animated intro message and waits for player input.
- menuDisplay() – Displays main menu and returns user's choice.
- play() – Main game loop handling gameplay mechanics.
- CursorLocation(x, y) – Moves the console cursor to a given X and Y coordinate.
- cursorInvisible() – Hides the blinking console cursor for a cleaner look.
- drawBorder() – Draws the outer game screen boundary and the side panel separator.
- genarateEnemy(e) – Spawns enemy tanks at random positions.
- driveEnemy(n) – Moves enemy tanks downward and draws them based on type.
- shootBullet() – Fires a bullet from the tank position.
- pullBullets() – Updates and renders active bullets on screen.
- bulletHit() – Detects and handles collision between bullets and enemies.
- collision() – Checks whether any enemy has collided with the tank.
- killEnemy(i) – Removes an enemy from screen and deactivates it.
- updateScore() – Updates the score display.
- disSidepannel() – Shows score, health, reload bar, and control instructions.
- drawRoadLines() – Displays moving road lines to simulate motion.
- instructions() – Displays game story, mission brief, enemy types, and controls.
- LevelSelection() – Placeholder function showing upcoming level selection feature.
- gameOver() – Displays game over screen and final score.

User Guide

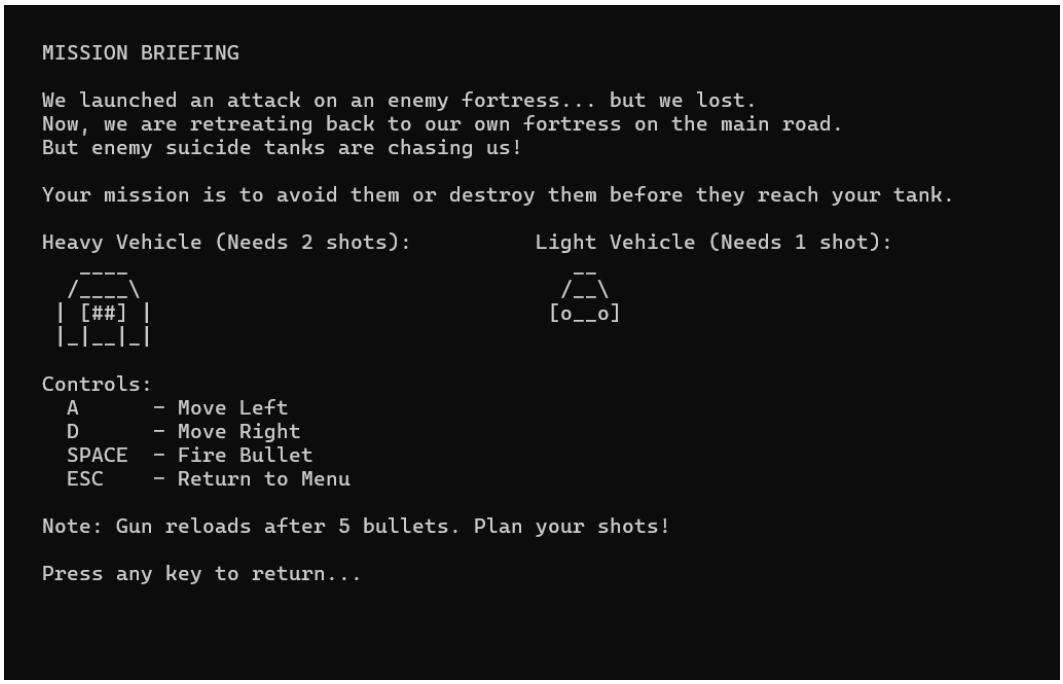
The purpose of this game is to obtain the highest points by destroying the enemy tanks while not getting hit by them.

1) Open screen
    - Once you run the provided executable file, you will be directed to the opening screen.
    - Press any key to continue on to the main menu.

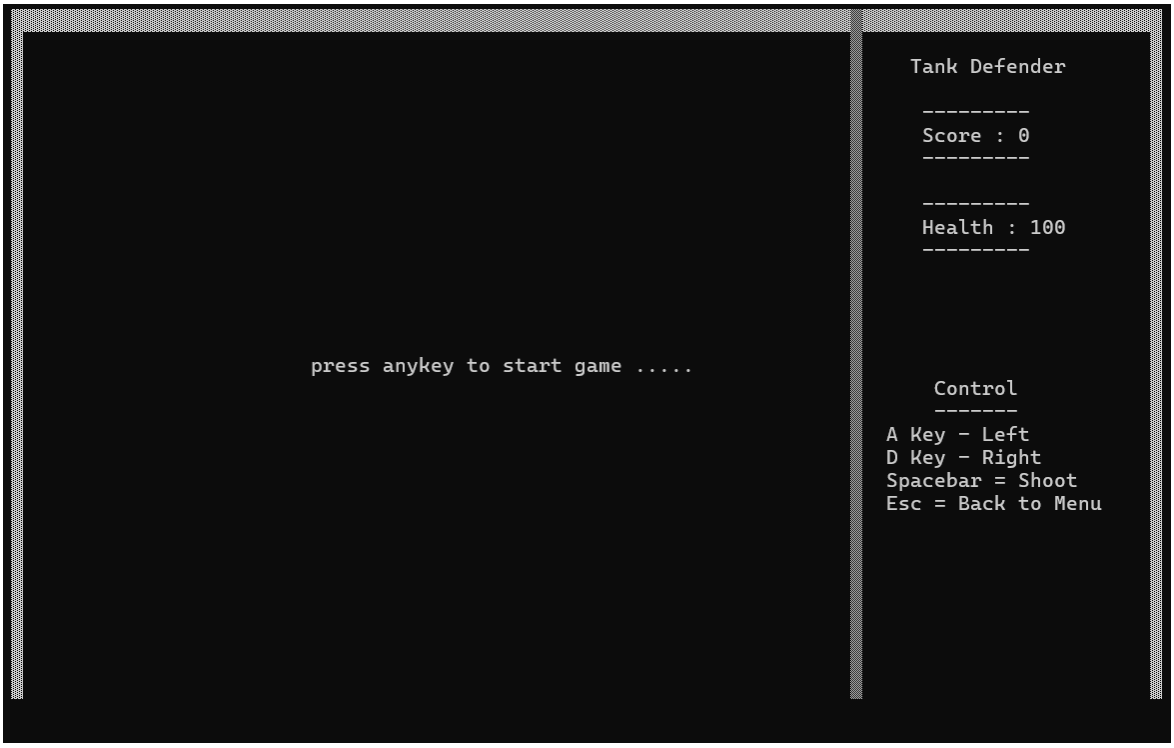
2) Main menu



- Press 2 to see the instructions.

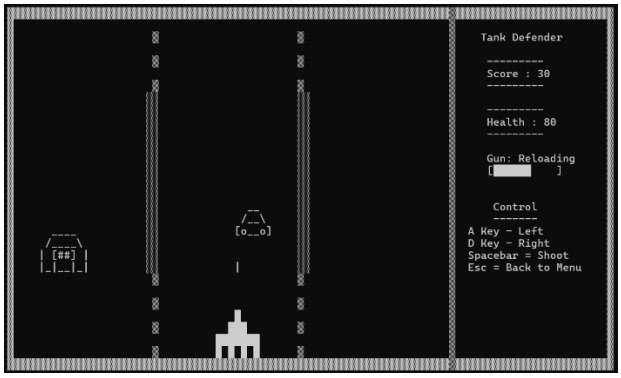


- Press any key to return back to the main menu.
- Press 1 to start the game.



3) How to play

- You can start playing by initially pressing any key.
- Use A and D keys to control your tank and the space bar to shoot.
- You need one shot for light enemy tanks and 2 shots for heavy enemy tanks. Your gun will reload after five shots.
- You will earn 10 points for each enemy tank.
- Your health will decrease by 20 for being hit by a heavy tank and by 30 for being hit by a light tank.
- If the health hits 0, the game is over; otherwise, you can continue scoring.



4) Exiting the game

- Press Esc to go back to main menu.
- Select 4 to exit.

Challenges

- Managing real-time input and rendering in a console environment:
  Handling simultaneous input and display updates in a text-based interface without freezing or blocking user actions.

- Ensuring smooth animation without flickering or lag:
  Redrawing moving objects like tanks and bullets without leaving visual trails or causing screen jitter.

- Handling multiple enemies and bullets simultaneously:
  Tracking and updating the state of several active enemies and bullets in real-time without conflicts or memory issues.

- Implementing collision detection with limited console graphics:
  Designing effective collision logic using only character positions and ASCII art shapes.

- Balancing game difficulty with reload mechanics and enemy health:
  Creating fair gameplay by adjusting how fast players can shoot and how tough enemies are to defeat.

- Synchronizing frame rate and game speed using Sleep():
  Controlling the pace of the game to make it playable and consistent, regardless of system speed.

Future enhancements and improvements

- **Add multiple difficulty levels with different enemy types and speeds:**
  Allow players to choose between easy, medium, or hard modes, each with unique pacing and enemy behavior.
- **Introduce new enemy types: drones, landmines, and rewards:**
  Add more variety and challenge with flying drones, ground hazards, and collectible bonuses to make gameplay richer.
- **Implement sound effects for shooting and explosions:**
  Use basic system beeps or external libraries to give audio feedback for key events like firing or taking damage.
- **Add high score saving and leaderboard:**
  Store player scores in a file and display a leaderboard to encourage competition and replayability.
- **Enhance graphics with color and better ASCII art:**
  Use SetConsoleTextAttribute() to add color to tanks, bullets, and enemies, making the game visually more engaging.
- **Add multiplayer or networked gameplay:**
  Introduce a two-player mode or networked version where players can compete or cooperate.
- **Optimize code for cross-platform compatibility beyond Windows:**
  Replace Windows-specific libraries like windows.h and conio.h with more portable alternatives to support Linux and macOS.

Conclusion

The development process of *Tank Defender* has been a valuable learning experience in applying C++ programming to a real-world project. It allowed us to deepen our understanding of game logic, user interaction, and the structure of console-based games. Throughout the project, we encountered challenges that helped improve our problem-solving skills, from managing real-time input and rendering to handling enemy logic and collision detection. This project not only enhanced our technical knowledge but also taught us the importance of teamwork, organization, and iterative development in software projects.

Reference

C++ Forum. (n.d.). *About Conio.h*. https://cplusplus.com/forum/beginner/284909/

GeeksforGeeks. (2017, January 6). *Basic graphic programming in C++*. https://www.geeksforgeeks.org/cpp/basic-graphic-programming-in-c/

GrantMeStrength. (n.d.). *Winbase.h header - Win32 apps*. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/api/winbase/

Instructables. (2019, May 8). *C++ Snake Game (Simple!)* https://www.instructables.com/C-Snake-Game-Simple/

Stack Overflow. (n.d.). *Update console without flickering C++*. Retrieved July 18, 2025, from https://stackoverflow.com/questions/34842526/update-console-without-flickering-c

Appendix A : Game Access (GitHub QR code)

To view and download the full Tank Defender game project, you can scan the below QR code.

Appendix B : Full Game Source Code

```
#include <iostream>

#include <conio.h>

#include <windows.h>



using namespace std;


#define screen_width 95 // Width of the console screen

#define screen_height 30 // Height of the console screen

#define playspace_width 70 // Width of the game play space



int status =1 ; // Variable to control the game loop

int tankPos=playspace_width/2; // Variable to keep track of the tank position

char tank[4][7] = {

    {' ', ' ', ' ', char(219), ' ', ' ', ' '},

    {' ', ' ', char(219), char(219), char(219), ' ', ' '},     // Top cannon

    {char(219), char(219), char(219), char(219), char(219), char(219), char(219)}, // Body

    {char(219), ' ', char(219), ' ', char(219), ' ', char(219)}  // Wheels

}; // shape of the tank
```

```
int score = 0; // Variable to keep track of the score
float Health = 100.0; // Variable to keep track of the number of lives
const int MAX_ENEMIES = 3; // Maximum number of enemies
const int MAX_BULLETS = 5; // Maximum number of bullets
struct Enemy {
    int x;
    int y;
    bool isActive;
    char type; // 'L' = light, 'H' = heavy
    int health; // track how many hits it can take
};



Enemy enemies[MAX_ENEMIES]; // Array to hold enemies


struct Bullet {
    int x;
    int y;
    bool isActive;
}; // struct for bullets to sve x,y and status of the bullet


Bullet playerBullets[MAX_BULLETS]; // Support multiple bullets
```

```
const int MAX_ROAD_ROWS = screen_height - 2; // Maximum number of rows for road lines

char roadPattern[MAX_ROAD_ROWS]; // stores line pattern per row

// For controlling pattern switching timing

static int roadFrameCounter = 0; // counts frames for movement

static int patternFrameCounter = 0; // counts frames for pattern duration

static int currentPattern = 0; // index of current road pattern


const int patternDurationFrames = 15; // Number of frames a pattern runs (~1.5 seconds if Sleep(100))

const int totalPatterns = 3; // number of patterns you have

int rp = 50; // reload time in frames (100ms each frame, so 5 seconds total)

int bulletBuffer =0;

bool isReloading = false;

int reloadCounter = rp; // means that wait for rp/10 seconds to reload


HANDLE console=GetStdHandle(STD_OUTPUT_HANDLE); // variable used to make changes to the console

COORD CursorPosition; // variable used to change cursor position in the console according to x and y coordinates


void welcome(); // Function for the welcome scree

void CursorLocation(int , int ); // Function for setting cursor position

void cursorInvisible(); // Function for making the cursor invisible
```

```
char menuDisplay(); // Function for displaying the menu
void play(); // Function for starting the game
void drawBorder(); // Function for drawing the border of the game
void worktoInput(int ); // Function for handling user input
void genarateEnemy(int e); // Function for generating enemies
void updateScore(); // Function for updating the score
void disSidepannel(); // Function for displaying the side panel
void driveTank(); // Function for driving the tank
int collision(); // Function for checking collisions
void driveEnemy(int n); // Function for driving the enemy
int bulletHit(); // Function for checking if a bullet has hit an enemy
void shootBullet(); // Function for shooting bullets
void pullBullets(); // Function for pulling bullets
void gameOver(); // Function for handling game over
void killEnemy(int i); // Function for killing an enemy
void drawRoadLines(); // Function for drawing road lines
void instructions(); // Function for displaying instructions
void LevelSelection(); // Function for level selection



int main (){

    system("mode con: cols=96 lines=31"); // Set console size
```

```
cursorInvisible(); // make curser invissible for better expearience
welcome(); // Call the welcome function
Sleep(1000); // Wait for 1 second
system("cls"); // Clear the console

while (status) {

  char option = menuDisplay() ; // Call the menu display function

   if (option == '1') {
      play(); // Call the play function
   }
   else if (option == '2') {
      instructions(); // call the instructions function
   }
   else if (option == '3') {
      LevelSelection(); // comming soon call the level selection function
   }
   else if (option == '4') {
      status = 0; // Exit the loop and end the program
   }
   else {
      system("cls"); // Clear the console
```

```
            CursorLocation(30, 10);
            cout << "Invalid option! Please try again." << endl;
        }

    }

}

void welcome() {

    system("cls");
    CursorLocation(40, 10); // set cursor position to (40, 12)
    cout << "TANK DEFENDER" << endl;
    Sleep(200);
    CursorLocation(35, 15);
    cout<< "Created by : " ; // animation effect for starting the game
    Sleep(200);
    cout << "G" ;
    Sleep(200);
    cout << "R" ;
    Sleep(200);
    cout << "O" ;
    Sleep(200);
```

```
cout << "U" ;
Sleep(200);
cout << "P" ;
Sleep(200);
cout << " " ;
Sleep(200);
cout << "1" ;
Sleep(200);
cout << "3" ;
Sleep(200);
cout << " " ;
Sleep(100);
 cout << "." ;
Sleep(100);
 cout << "." ;
Sleep(100);
 cout << "." ;
Sleep(100);
 cout << "." ;
Sleep(100);
 cout << "." ;
Sleep(100);
 cout << "." << endl;
```

```
    Sleep(500);
    CursorLocation(35, 20);

    cout << "Press any key to continue..." << endl;
    _getch(); // Wait for a key press
    system("cls"); // Clear the console
}

void CursorLocation(int x, int y){
                    CursorPosition.X=x; // Set the X coordinate
                    CursorPosition.Y=y; // Set the Y coordinate
                    SetConsoleCursorPosition(console, CursorPosition); // Set the cursor
position in the console
}

void cursorInvisible(){
                    CONSOLE_CURSOR_INFO lpCursor; // Create a console cursor info
structure
                    lpCursor.bVisible=0; // Set the cursor visibility to false
                    lpCursor.dwSize=20; // Set the cursor size
                    SetConsoleCursorInfo(console, &lpCursor); // Set the cursor info in the
console
}
```

```
char menuDisplay() {


    system("cls");
    CursorLocation(30, 10);
    cout << "TANK DEFENDER" << endl;
    CursorLocation(30, 12);
    cout << "1. Start Game" << endl;
    CursorLocation(30, 13);
    cout << "2. Instructions" << endl;
    CursorLocation(30, 14);
    cout << "3. Levels " << endl;
    CursorLocation(30, 16);
    cout << "4. Exit" << endl;
    CursorLocation(30, 18);
    cout << "Select an option: ";
    char choice; // create a variable to store the user's choice
    CursorLocation(50, 18); // Set cursor position for input
    cin >> choice ;
    return choice; // Return the selected option
}
```

```
void play(){

    system("cls"); // Clear screen at game start
    bulletBuffer =0; // Reset bullet buffer
    isReloading = false; // Reset reloading status
    reloadCounter = rp; // means that wait for rp/10 seconds to reload

    for (int i = 0; i < MAX_BULLETS; i++) { // Initialize player bullets
    playerBullets[i].isActive = false;
}
    tankPos = playspace_width / 2 - 2; // Reset tank position
    // Initialize first two enemies
    enemies[0].x = 10; // Initial X position for first enemy
    enemies[0].y = 4; // Initial Y position for first enemy
    enemies[0].isActive = true;  // Set first enemy as active
    enemies[0].type = 'L'; // Light enemy

    enemies[1].x = 30;
    enemies[1].y = 4;
    enemies[1].isActive = true;
    enemies[1].type = 'H'; // Heavy enemy

    // Optional: deactivate the rest
```

```
    for (int i = 2; i < MAX_ENEMIES; i++) {
        enemies[i].isActive = false;
    }
   for (int i = 0; i < MAX_BULLETS; i++) { // no b ullets at start
    playerBullets[i].isActive = false;
}
    // Initialize road line patterns to empty
    for (int i = 0; i < MAX_ROAD_ROWS; i++) {
        roadPattern[i] = ' '; // Start with no road lines
    }



    drawBorder(); // Draw the game border
    genarateEnemy(0); // Generate first enemy
    genarateEnemy(1); // Generate second enemy
    genarateEnemy(2); // Generate third enemy
    updateScore(); // Update the score display
    disSidepannel(); // Display the side panel
    CursorLocation(25,15);
    cout<<"press anykey to start game ..... ";
    _getch(); // Wait for a key press
    CursorLocation(25,15);
    cout<<"                              "; // Clear the line
```

```
int prevTankPos = tankPos; // Store the previous tank position for clearing


while (true){


    drawRoadLines();
    if (_kbhit()) { // Check if a key is pressed
        char key = _getch(); // Get the pressed key


        if(key == 27) { // If the key is ESC
            if (score > 0) {
             gameOver(); // Call the game over function
            }
            break; // Exit the game loop
        }
        else {
        worktoInput(key); // Call the function to handle input
        }
}


// Clear the tank from the old position
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 7; j++) {
```

```
        CursorLocation(prevTankPos + j, screen_height - 5 + i);
        cout << ' ';
    }
}

driveTank(); // Call the function to drive the tank
prevTankPos = tankPos; // Update previous position tracker
disSidepannel(); // Call the function to display the side panel
driveEnemy(0); // Call the function to drive the first enemy
driveEnemy(1); // Call the function to drive the second enemy
driveEnemy(2); // Call the function to drive the third enemy
pullBullets(); // Call the function to pull bullets

                    if(collision()==1){

                            gameOver(); // ends the game when all 3 lifes are lost
                            return;
                        }

if(bulletHit()==1){ // 1 is returned to the condtion if a bullet has hit any of the enemy crafts
                            updateScore(); // updates the score
                        }
```

```
    for (int i = 0; i < MAX_ENEMIES; i++) {
        if (!enemies[i].isActive) {
            genarateEnemy(i);  // Bring back the enemy
        }
    }

    if (isReloading) {
         reloadCounter--; // Decrease reload counter
    if (reloadCounter <= 0) {
        isReloading = false; // Reset reloading status
        bulletBuffer = 0; // Reset bullet buffer
    }
}

Sleep(100);

}
}

void drawBorder(){

                        char ch=178; // assigns the character with ASCII value 176 to variable
ch
```

```
                    // draws the lower border
                    for(int i=0; i<=screen_width; i++){
                        CursorLocation(i, screen_height);
                        cout << ch;
                    }

// draws the upper border
for(int i=0; i<=screen_width; i++){
   CursorLocation(i, 0);
   cout << ch;
}

                    // draws the right and left border
                    for(int i=0; i<screen_height ;i++){
                        CursorLocation(0, i);
                        cout << ch;
                        CursorLocation(screen_width, i);
                        cout << ch;
                    }

                    ch = 177;
```

```
                        // draws the border which seperates the game-play space and
instructions/score panel
                        for(int i=0; i<screen_height; i++){
                            CursorLocation(playspace_width, i);
                            cout << ch;
                        }
}

void genarateEnemy(int e) {
    bool positionValid = false; // Flag to check if position is valid
    int attempts = 0; // Limit attempts to find a valid position

    while (!positionValid && attempts < 20) {
        int newX = 3 + rand() % (playspace_width - 10); // Random X position
        positionValid = true;

        // Check for overlap with other enemies
        for (int i = 0; i < MAX_ENEMIES; i++) {
            if (i != e && enemies[i].isActive) {
                if (abs(enemies[i].x - newX) < 10) { // If too close
                    positionValid = false; // Mark position as invalid
                    break; // Exit the loop if overlap found
                }
```

```
            }
        }

        if (positionValid) {
            enemies[e].x = newX; // Set new X position
            enemies[e].y = 4; // Reset Y position
            enemies[e].isActive = true; // Activate the enemy

            // Randomly assign type
            enemies[e].type = (rand() % 2 == 0) ? 'L' : 'H'; // Light or Heavy
            enemies[e].health = (enemies[e].type == 'L') ? 1 : 2; // ðŸ’¡ Light = 1 hit, Heavy = 2 hits
        }

        attempts++;
    }

    // If failed to find a valid position after many tries, deactivate
    if (!positionValid) {
        enemies[e].isActive = false;
    }
}
```

```
void updateScore(){
                        CursorLocation(playspace_width+6, 5); // x-
coordinate=playspace_width+6 and y-coordinate=5
                        cout << "Score : " << score << endl; // displays score
}

void disSidepannel() {

                        CursorLocation(playspace_width+5, 2);
                        cout << "Tank Defender";
                        CursorLocation(playspace_width+6, 4);
                        cout << "---------";
                        CursorLocation(playspace_width+6, 6);
                        cout << "---------";
                        CursorLocation(playspace_width+6, 8);
                        cout << "---------";
                        CursorLocation(playspace_width+6, 9);
                        cout << "Health : " << Health << endl; // displays lives left
                        CursorLocation(playspace_width+6, 10);
                        cout << "---------";


   if (isReloading) { // If reloading
```

```
    CursorLocation(playspace_width+6, 12);
    cout << "Gun: Reloading ";

    // Progress bar based on reloadCounter
    int totalSteps = 10;  // Total "fill" slots
    int progress = totalSteps - reloadCounter; // e.g., if reloadCounter = 7 â†’ progress = 3
    if (progress < 0) progress = 0; // safety
    if (progress > totalSteps) progress = totalSteps; //

    string bar = "["; // animation effect for showing the progress of reloading
    for (int i = 0; i < totalSteps; i++) {
        if (i < progress)
            bar += char(219);  // solid block
        else
            bar += ' ';
    }
    bar += "]";

    CursorLocation(playspace_width+6, 13);
    cout << bar << "    ";
}
```

```
                        CursorLocation(playspace_width+7, 16);
                        cout << "Control";
                        CursorLocation(playspace_width+7, 17);
                        cout << "-------";
                        CursorLocation(playspace_width+2, 18);
                        cout << " A Key - Left";
                        CursorLocation(playspace_width+2, 19);
                        cout << " D Key - Right";
                        CursorLocation(playspace_width+2, 20);
                        cout << " Spacebar = Shoot";
    CursorLocation(playspace_width+2, 21);
                        cout << " Esc = Back to Menu";

}

void worktoInput(int w){
    switch (w) {
        case 'a':
        case 'A':
            if (tankPos > 1) {
                tankPos-=2;
            }
```

```
            break;


        case 'd':
        case 'D':
            if (tankPos < playspace_width - 7) {
                tankPos+=2;
            }
            break;


        case ' ': // Shoot bullet
            shootBullet();
            break;


        default:
            break;
    }
}


void driveTank() {
    for (int i =0; i <4 ; i++){
        for (int j = 0; j < 7; j++) {
            CursorLocation(tankPos + j, screen_height - 5 + i);
            cout << tank[i][j]; // Draw the tank
```

```
        }
    }

}

void driveEnemy(int n) {
    if (!enemies[n].isActive) {// If enemy is not active, skip
        return;
    }
    // Clear previous enemy frame (width: 7 or more)
    for (int i = 0; i < 5; i++) {
        CursorLocation(enemies[n].x, enemies[n].y + i);
        cout << "        "; // Clear 8 characters width
    }

    // Move enemy downward
    enemies[n].y++;

    // Stop if out of bounds
    if (enemies[n].y + 4 >= screen_height - 1) {
        enemies[n].isActive = false;
        return;
    }
```

```
// Draw based on type
if (enemies[n].type == 'L') {
    //  Light Vehicle
    CursorLocation(enemies[n].x, enemies[n].y);
    cout << "  __";
    CursorLocation(enemies[n].x, enemies[n].y + 1);
    cout << " /__\\";
    CursorLocation(enemies[n].x, enemies[n].y + 2);
    cout << "[o__o]";
} else {
    //  Heavy Vehicle
    CursorLocation(enemies[n].x, enemies[n].y);
    cout << "  ____";
    CursorLocation(enemies[n].x, enemies[n].y + 1);
    cout << " /____\\";
    CursorLocation(enemies[n].x, enemies[n].y + 2);

    cout << "| ["; // animation effect for showing health of heavy enemeies
    if (enemies[n].health == 2) cout << "##";
    else if (enemies[n].health == 1) cout << "# ";
    else cout << "  ";
    cout << "] |";
```

```
        CursorLocation(enemies[n].x, enemies[n].y + 3);
        cout << "|_|__|_|";
    }
}

int collision() {
    for (int i = 0; i < MAX_ENEMIES; i++) {
        if (!enemies[i].isActive){
            continue;
        }

        // If enemy has reached the tank area (bottom rows)
        if (enemies[i].y + 3 >= screen_height - 4) { // Adjust based on your tank's height
            // Check horizontal overlap with tank position
            if (enemies[i].x + 6 >= tankPos && enemies[i].x <= tankPos + 6) {
                // Apply damage based on enemy type
                if (enemies[i].type == 'L') {
                    Health -= 30.0;
                } else if (enemies[i].type == 'H') {
                    Health -= 20.0;
```

```
            }
             killEnemy(i); // Call the function to kill the enemy

            // Mark this enemy as deactivated after collision
            enemies[i].isActive = false;

            // Update health display
            CursorLocation(playspace_width + 6, 9);
            cout << "Health : " << (Health < 0 ? 0 : Health) << "    ";

            // Check for game over
            if (Health <= 0.0) {
                return 1; // Game over
            }
        }
    }
    return 0; // No game over
}



void gameOver() {
```

```
    system("cls"); // Clear the console
    CursorLocation(30, 10); // Set cursor position for game over message
    cout << "Game Over!" << endl;
    CursorLocation(30, 12);
    cout << "Your final score: " << score << endl;
    CursorLocation(30, 14);
    cout << "Press any key to return to the menu..." << endl;
    _getch(); // Wait for a key press
   // status = 0; // Exit the game loop
}

int bulletHit() {
    for (int b = 0; b < MAX_BULLETS; b++) { // Loop through all bullets
        if (!playerBullets[b].isActive)
            continue;

        int bulletX = playerBullets[b].x;
        int bulletY = playerBullets[b].y;

        for (int i = 0; i < MAX_ENEMIES; i++) {
            if (!enemies[i].isActive)
                continue;
```

```
int ex = enemies[i].x;
int ey = enemies[i].y;
int width = (enemies[i].type == 'L') ? 6 : 8;
int height = (enemies[i].type == 'L') ? 3 : 4;

if (bulletX >= ex && bulletX <= ex + width &&
    bulletY >= ey && bulletY <= ey + height) {

    // Bullet hits enemy
    // Clear the bullet from screen
    CursorLocation(bulletX, bulletY);
    cout << '*'; // Mini explosion animation
    Sleep(30);   // Small delay to show explosion
    CursorLocation(bulletX, bulletY);
    cout << ' ';

    playerBullets[b].isActive = false;

    enemies[i].health--; // Decrease enemy health

    if (enemies[i].health <= 0) {
        killEnemy(i);
        score += 10;
```

```
            }

            return 1; // Hit detected
        }
    }
}

return 0; // No hit
}

void shootBullet() {
    if (isReloading) return; // Stop if reloading

    for (int i = 0; i < MAX_BULLETS; i++) {
        if (!playerBullets[i].isActive) {
            playerBullets[i].x = tankPos + 3;
            playerBullets[i].y = screen_height - 2;
            playerBullets[i].isActive = true;
            bulletBuffer++;

            if (bulletBuffer >= 5) { // Allow 5 shots max
                isReloading = true;
```

```
            reloadCounter = 10; // wait ~1 second (10 frames * 100ms)
        }

        break;
    }
  }
}

void pullBullets() {
  for (int i = 0; i < MAX_BULLETS; i++) {

    if (playerBullets[i].isActive) {
        // Clear previous
        CursorLocation(playerBullets[i].x, playerBullets[i].y);
        cout << ' ';

        playerBullets[i].y--; // this is to move the bullet upward

        if (playerBullets[i].y < 1) {
            playerBullets[i].isActive = false; // this is to check if the bullet is out of bounds ( if yes bullet is deactivated  )
        } else {
            CursorLocation(playerBullets[i].x, playerBullets[i].y);
```

```
                cout << '|'; // Draw the bullet
            }
        }
    }
}


void killEnemy(int i) {

    if (!enemies[i].isActive) // If enemy is already inactive no need to kill it again
        return;

    int ex = enemies[i].x;
    int ey = enemies[i].y;

     // this height annd width variables used to store height and width of the enemy crafts
    // light = 6 width and 3 height
    // heavy = 8 width and 4 height
    int height = (enemies[i].type == 'L') ? 3 : 4;
    int width = (enemies[i].type == 'L') ? 6 : 8;


    for (int j = 0; j < height; j++) {
```

```
        CursorLocation(ex, ey + j);
        for (int k = 0; k < width; k++) {
            cout << ' ';
        }
    }

    enemies[i].isActive = false;

}

void drawRoadLines() {
    static int roadLinePattern = 0; // Current pattern
    static int frameCount = 0; // For timing
    static const int switchPatternFrames = 15; // Change pattern every ~1.5 seconds

    static int linePositions[screen_height] = { 0 }; // Holds line types per row

    // Shift all lines down (simulate movement upward)
    for (int i = screen_height - 2; i > 1; i--) {
        linePositions[i] = linePositions[i - 1];
    }

    // Generate new top line based on current pattern
```

```
switch (roadLinePattern) {
    case 0: // Single dashed line
        linePositions[1] = (frameCount % 2 == 0) ? 1 : 0;
        break;

    case 1: // Double lines
        linePositions[1] = 2;
        break;

}

// Draw lines
for (int i = 1; i < screen_height - 1; i++) {
    // Clear all road lane positions
    for (int offset = -1; offset <= 1; offset++) {
        CursorLocation(playspace_width / 3 + offset, i); cout << ' ';
        CursorLocation(2 * playspace_width / 3 + offset, i); cout << ' ';
        CursorLocation(playspace_width / 2 + offset, i); cout << ' ';
    }

    if (linePositions[i] == 1) {
        // single line
        CursorLocation(playspace_width / 3, i); cout << char(177);
```

```
            CursorLocation(2 * playspace_width / 3, i); cout << char(177);
        } else if (linePositions[i] == 2) {
            // double line
            CursorLocation(playspace_width / 3 - 1, i); cout << char(176);
            CursorLocation(playspace_width / 3, i); cout << char(176);
            CursorLocation(2 * playspace_width / 3, i); cout << char(176);
            CursorLocation(2 * playspace_width / 3 + 1, i); cout << char(176);
        } else if (linePositions[i] == 3) {
            CursorLocation(playspace_width / 2, i); cout << '.';
        }
    }

    frameCount++;

    if (frameCount % switchPatternFrames == 0) {
        roadLinePattern = (roadLinePattern + 1) % 2; // Switch between 0 and 1
    }
}



void instructions() {
    system("cls");
    CursorLocation(5, 2);
```

```
cout << "MISSION BRIEFING";

Sleep(200);

CursorLocation(5, 4);

cout << "We launched an attack on an enemy fortress... but we lost.";

Sleep(200);

CursorLocation(5, 5);

cout << "Now, we are retreating back to our own fortress on the main road.";

Sleep(200);

CursorLocation(5, 6);

cout << "But enemy suicide tanks are chasing us!";

Sleep(200);


CursorLocation(5, 8);

cout << "Your mission is to avoid them or destroy them before they reach your tank.";

Sleep(200);


CursorLocation(5, 10);

cout << "Heavy Vehicle (Needs 2 shots):";

Sleep(200);

CursorLocation(5, 11);

cout << "   ____";

CursorLocation(5, 12);

cout << "  /____\\";
```

```
CursorLocation(5, 13);
cout << " | [##] |";
CursorLocation(5, 14);
cout << " |_|__|_|";
Sleep(200);

CursorLocation(45, 10);
cout << "Light Vehicle (Needs 1 shot):";
Sleep(200);
CursorLocation(45, 11);
cout << "   __";
CursorLocation(45, 12);
cout << "  /__\\";
CursorLocation(45, 13);
cout << " [o__o]";
Sleep(200);

CursorLocation(5, 16);
cout << "Controls:";
CursorLocation(5, 17);
cout << "  A      - Move Left";
CursorLocation(5, 18);
cout << "  D      - Move Right";
```

```
    CursorLocation(5, 19);
    cout << "  SPACE  - Fire Bullet";
    CursorLocation(5, 20);
    cout << "  ESC    - Return to Menu";

    CursorLocation(5, 22);
    cout << "Note: Gun reloads after 5 bullets. Plan your shots!";

    Sleep(200);
    CursorLocation(5, 24);
    cout << "Press any key to return...";
    getch(); // Wait for a key press

}

void LevelSelection(){

    system("cls");
    CursorLocation(25, 8);
    cout << "Level Selection (Coming Soon!)" << endl;
    CursorLocation(25, 12);
```

```
    cout << "This feature will allow you to choose different levels with varying difficulty. " << endl;


    CursorLocation(25, 14);
    cout << "In levels there will be shooting enemies, Drones , landmines ,rewards and lot more .. " << endl;
     CursorLocation(25, 16);
    cout << "Stay Tuned ...    :)  " << endl;
    CursorLocation(25, 22);
    cout << "Press any key to return ..... " << endl;
    _getch(); // Wait for a key press


}



// End of the code
```