UNIVERSITY OF ENGINEERING AND TECHNOLOGY TAXILA

TOPIC:

SPACE INVADERS

SUBMITTED BY:

UMAR(24-CS-130)

EMAN (24-CS-105)

ALIHA (24-CS-39)

SARA(24-CS-37)

SUBMITTED TO:

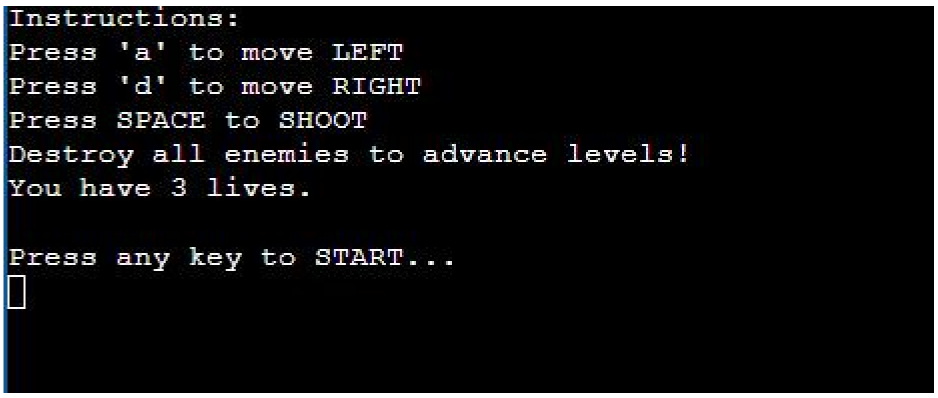
DR.FAHEEM

CODE

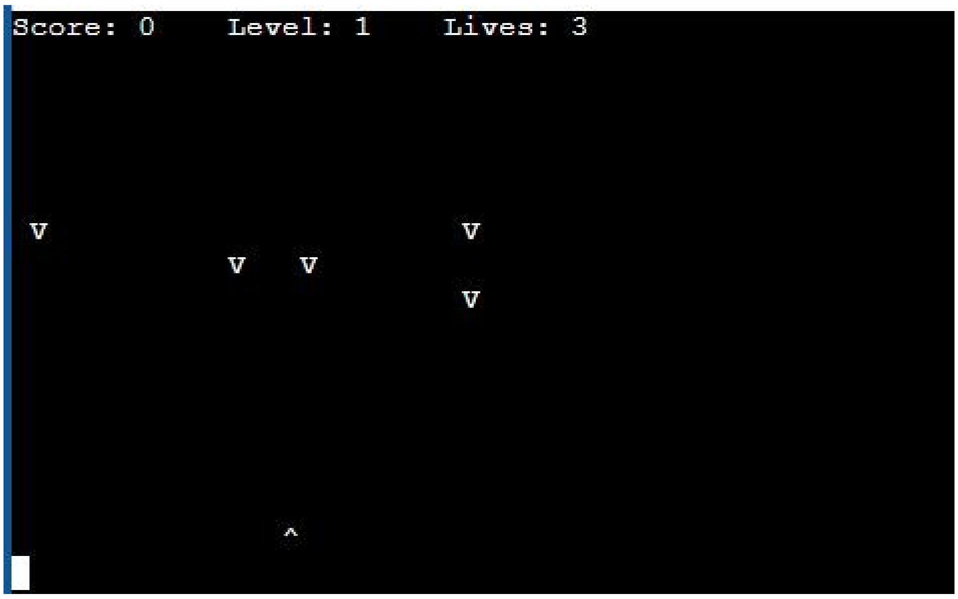
```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
#include <cstdlib>

#ifdef _WIN32
#include <conio.h>
#else
#include <termios.h>
#include <unistd.h>
#include <fcntl.h>
#endif

using namespace std;

// Constants for grid size
const int GRID_WIDTH = 30;
const int GRID_HEIGHT = 15;
const int MAX_LIVES = 3;

bool kbhit() {
#ifdef _WIN32
    return _kbhit();
#else
    struct termios oldt, newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if (ch != EOF) {
        ungetc(ch, stdin);
        return true;
    }
    return false;
#endif
}

char getch() {
#ifdef _WIN32
    return _getch();
#else
    struct termios oldt, newt;
    char ch;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
#endif
}

class GameObject {
protected:
    int x, y;
    char symbol;
public:
    GameObject(int x, int y, char symbol) : x(x), y(y), symbol(symbol) {}
    virtual void move() = 0;
    int getX() { return x; }
    int getY() { return y; }
    char getSymbol() { return symbol; }
    void setPosition(int newX, int newY) { x = newX; y = newY; }
};
```

```
class Player : public GameObject {
public:
    Player(int x, int y) : GameObject(x, y, '^') {}

    void move() override {
        if (kbhit()) {
            char key = getch();
            if (key == 'a' && x > 0) x--;
            if (key == 'd' && x < GRID_WIDTH - 1) x++;
        }
    }
};

class Bullet : public GameObject {
public:
    Bullet(int x, int y) : GameObject(x, y, '|') {}

    void move() override {
        if (y > 0) y--;
    }
};

class Enemy : public GameObject {
public:
    Enemy(int x, int y) : GameObject(x, y, 'V') {}

    void move() override {
        if (y < GRID_HEIGHT - 1) y++;
    }
};

class Game {
private:
    Player player;
    vector<Bullet> bullets;
    vector<Enemy> enemies;
    int enemyMoveCounter = 0;
    int score = 0;
    int level = 1;
    int lives = MAX_LIVES;
    int enemySpeed = 10;

public:
    Game() : player(GRID_WIDTH / 2, GRID_HEIGHT - 1) {
        spawnEnemies(5);
    }

    void showInstructions() {
        cout << "Instructions:\n";
        cout << "Press 'a' to move LEFT\n";
        cout << "Press 'd' to move RIGHT\n";
        cout << "Press SPACE to SHOOT\n";
        cout << "Destroy all enemies to advance levels!\n";
        cout << "You have " << MAX_LIVES << " lives.\n\n";
        cout << "Press any key to START...\n";
        getch();
    }

    void spawnEnemies(int count) {
        enemies.clear();
        for (int i = 0; i < count; i++) {
            enemies.push_back(Enemy(rand() % GRID_WIDTH, rand() % 3));
        }
    }

    void update() {
        player.move();

        for (size_t i = 0; i < bullets.size(); i++) {
            bullets[i].move();
            if (bullets[i].getY() <= 0) {
                bullets.erase(bullets.begin() + i);
                i--;
            }
        }

        enemyMoveCounter++;
        if (enemyMoveCounter % enemySpeed == 0) {
            for (auto &enemy : enemies) {
                enemy.move();
            }
        }

        for (size_t i = 0; i < bullets.size(); i++) {
            for (size_t j = 0; j < enemies.size(); j++) {
                if (bullets[i].getX() == enemies[j].getX() && bullets[i].getY() == enemies[j].getY()) {
                    bullets.erase(bullets.begin() + i);
                    enemies.erase(enemies.begin() + j);
                    score += 10;
                    i--;
                    break;
                }
            }
        }
    }

    void render() {
#ifdef _WIN32
        system("cls");
#else
        system("clear");
#endif

        cout << "Score: " << score << "    Level: " << level << "    Lives: " << lives << "\n";

        for (int y = 0; y < GRID_HEIGHT; y++) {
            for (int x = 0; x < GRID_WIDTH; x++) {
                bool drawn = false;

                if (player.getX() == x && player.getY() == y) {
                    cout << player.getSymbol();
                    drawn = true;
                }

                for (auto &bullet : bullets) {
                    if (bullet.getX() == x && bullet.getY() == y) {
                        cout << bullet.getSymbol();
                        drawn = true;
                    }
            }

            for (auto &enemy : enemies) {
                if (enemy.getX() == x && enemy.getY() == y) {
                    cout << enemy.getSymbol();
                    drawn = true;
                }
            }

            if (!drawn) cout << " ";
        }
        cout << endl;
    }
}

void shoot() {
    bullets.push_back(Bullet(player.getX(), player.getY() - 1));
}

bool isGameOver() {
    for (auto &enemy : enemies) {
        if (enemy.getY() == GRID_HEIGHT - 1) {
            lives--;
            if (lives <= 0) return true;
            spawnEnemies(5 + (level - 1) * 3);
            return false;
        }
    }
    return false;
}

void nextLevel() {
    level++;
    if (level == 2) {
        enemySpeed = 7;
        spawnEnemies(8);
    } else if (level == 3) {
        enemySpeed = 5;
        spawnEnemies(12);
    } else {
        cout << "Congratulations! You completed all levels!\n";
        exit(0);
    }
}

bool restartPrompt() {
    cout << "Game Over! You lost all your lives.\n";
    cout << "Press 'r' to restart or any other key to exit...\n";
    char choice = getch();
    return (choice == 'r' || choice == 'R');
}

void resetGame() {
    bullets.clear();
    enemies.clear();
    score = 0;
    level = 1;
    lives = MAX_LIVES;
    enemySpeed = 10;
    player.setPosition(GRID_WIDTH / 2, GRID_HEIGHT - 1);
    spawnEnemies(5);
}

void run() {
    showInstructions();

    while (true) {
        update();
        render();

        if (kbhit()) {
            char key = getch();
            if (key == ' ') {
                shoot();
            }
        }

        if (isGameOver()) {
            if (restartPrompt()) {
                resetGame();
                continue;
            } else {
                break;
            }
        }

        if (enemies.empty()) {
            nextLevel();
        }

        this_thread::sleep_for(chrono::milliseconds(300));
    }
  }
};

int main() {
    Game game;
    game.run();
    return 0;
}
```

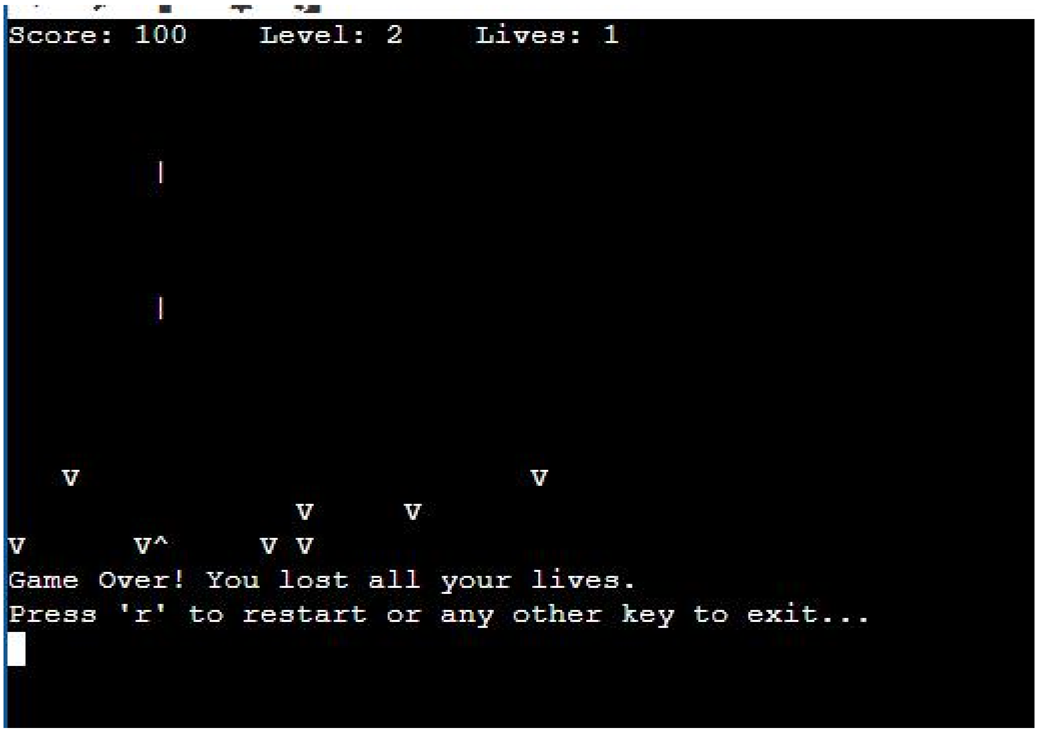
OUTPUT:



```
Instructions:
Press 'a' to move LEFT
Press 'd' to move RIGHT
Press SPACE to SHOOT
Destroy all enemies to advance levels!
You have 3 lives.

Press any key to START...
```



```
Score: 0    Level: 1    Lives: 3





 V                       V
            V   V
                         V






               ^
```



```
Score: 100    Level: 2    Lives: 1



        |



        |




   V                         V
                V     V
V      V^     V V
Game Over! You lost all your lives.
Press 'r' to restart or any other key to exit...
```

EXPLANATION :

Basic Concept

A player controls a shooter (^) at the bottom of a grid. Enemies (V) fall from the top, and the player shoots bullets (|) to destroy them. The goal is to destroy all enemies before they reach the bottom. The game progresses through levels, gets harder, and ends when all lives are lost.

Key Components

1. Grid Settings

- The game grid is 30 x 15.
- Player starts with 3 lives.

2. Platform-Independent Input

kbhit() and getch() allow reading keyboard input instantly (cross-platform: Windows/Linux).

3. Classes

GAME OBJECT

Base class for all moving objects (Player, Bullet, Enemy). Holds position and symbol

PLAYER

Inherits from GameObject. Moves left (a) or right (d).

BULLET

Inherits from GameObject. Moves upward to hit enemies.

ENEMY

Inherits from GameObject. Falls down from the top.

GAME

Controls all logic: movement, collisions, levels, scoring, rendering, game loop.

3.Main Game Loop

The game loop runs like this:

Show instructions once at the start.

Every cycle:

- Update player, bullets, enemies.
- Check for shooting (space key).
- Check for bullet-enemy collisions (adds points).
- If any enemy reaches the bottom:
- Lose a life.
- If all lives are lost → Game Over.
- If all enemies are gone → Next Level.
- Redraw the screen (console-based graphics).
- Add delay (300ms) to slow down the loop.

4.Level & Difficulty System

- Level 1: 5 enemies, normal speed.
- Level 2: 8 enemies, faster.
- Level 3: 12 enemies, fastest.
- After Level 3: Congratulatory message and game ends.

5.Restart Option

If the player loses all lives, they can press 'r' to restart the game.

6. Program Entry

main() creates a Game object and calls run() to start the game.

7. Conclusion

This code is a clean, object-oriented terminal game with cross-platform support, smooth controls, basic level system, and user-friendly interface. It teaches core programming concepts like:

- Inheritance & polymorphism
- Real-time input handling
- Dynamic arrays (std::vector)
- Game loops and rendering logic