PF PROJECT

By

ABDUL HASEEB (CS2142)

Instructor Name: SIR HAMMAD ABBASI
Subject: PF (LAB)
Class: BSCS-1
DATE: 29/01/2024

DEPARTMENT OF COMPUTER SCIENCE

SHIFA TAMEER-E-MILLAT UNIVERSITY

PARK ROAD CAMPUS.

SHOOTING GAME

Shooting game having file handling, after 5 hits its shows a bonus enemy and show the score, name and at which time game is player of previous player.

CODE & EXPLATION:

1. Headers & Constants:

```
#include <iostream>       // Basic I/O operations
#include <conio.h>        // Console input (getch)
#include <windows.h>      // Console manipulation
#include <ctime>          // Time functions
#include <fstream>        // File handling
#include <iomanip>        // Output formatting
#include <chrono>         // Precise timing

const int SCREEN_WIDTH = 90;     // Total console width
const int SCREEN_HEIGHT = 26;    // Total console height
const int PLAY_AREA_WIDTH = 70; // Gameplay zone width
```

- **Purpose:** Essential libraries for console control, timing, and data persistence.

2.Game State Variables:

```
HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
COORD cursorPosition;            // Cursor coordinates
int playerX = PLAY_AREA_WIDTH/2;// Player position
int enemyX, enemyY;              // Regular enemy position
bool enemyActive = true;         // Enemy status
int bonusX, bonusY;              // Bonus enemy position
bool bonusActive = false;        // Bonus enemy status
int bulletsX[20], bulletsY[20]; // Bullet positions
bool bullets[20] = {false};      // Bullet status
int score = 0;                   // Current score
int hitCounter = 0;              // Successful hits counter
```

- **Purpose:** Track all game elements' positions and states.

3.Core Utility Functions:

```
void moveCursor(int x, int y) {
    cursorPosition.X = x;
    cursorPosition.Y = y;
    SetConsoleCursorPosition(console, cursorPosition);
}

void hideCursor() {
    CONSOLE_CURSOR_INFO cursorInfo;
    cursorInfo.bVisible = false;
    SetConsoleCursorInfo(console, &cursorInfo);
}
```

- **Key Features:**
    - Precise cursor positioning
    - Hidden cursor for cleaner visuals

4. Visual Elements:

```
void drawGameBorder() {
    for(int x=0; x<SCREEN_WIDTH; x++) {
        moveCursor(x, 0); cout << "█"; // Top border
        moveCursor(x, SCREEN_HEIGHT-1); cout << "█"; // Bottom border
    }
}

void drawPlayer() {
    moveCursor(playerX, SCREEN_HEIGHT-2);
    cout << "▲"; // Player character
}
```

- **Visual Design:**
    - Uses Unicode block characters (█) for borders
    - Arrow symbols (▲/▼) for player/enemies

5. Enemy System:

```
void spawnEnemy(bool isBonus) {
    int& x = isBonus ? bonusX : enemyX;
    x = 5 + rand()%(PLAY_AREA_WIDTH-10); // Random X position
    (isBonus ? bonusActive : enemyActive) = true;
}

void updateEnemyPosition(bool isBonus) {
    int& y = isBonus ? bonusY : enemyY;
    if(++y > SCREEN_HEIGHT-3) { // Move down
        (isBonus ? bonusActive : enemyActive) = false;
        if(!isBonus) score = max(0, score-2); // Penalty
```

- **Mechanics:**
  - Regular enemies deduct 2 points if reach bottom
  - Bonus enemies spawn every 5 hits

6. Combat System:

```
void fireBullet() {
    for(int i=0; i<20; i++) {
        if(!bullets[i]) {
            bullets[i] = true;
            bulletsX[i] = playerX; // Start from player
            bulletsY[i] = SCREEN_HEIGHT-3;
            break;
        }
    }
}
```

- **Features:**
  - 20 bullet capacities
  - Vertical trajectory
  - Automatic bullet cleanup

7. Collision Detection:

```
bool checkCollisions(bool isBonus) {
    int ex = isBonus ? bonusX : enemyX;
    int ey = isBonus ? bonusY : enemyY;

    for(int i=0; i<20; i++) {
        if(bullets[i] && bulletsY[i]==ey &&
           abs(bulletsX[i] - ex) <= 2) {
            bullets[i] = false;
            return true;
        }
    }
```

- **Logic:**
  - Checks bullet-enemy proximity
  - 3-character wide hitbox

8.Score Management

```
void saveScore(const string& name) {
    time_t now = time(0);
    tm* lt = localtime(&now);
    char timestamp[20];
    strftime(timestamp, 20, "%Y-%m-%d %H:%M:%S", lt);

    ofstream records("scores.dat", ios::app);
    if(records) {
        records << left << setw(25) << name
                << setw(10) << score
                << timestamp << endl;
```

- **Data Format:**
  - Player name (25 char width)
  - Score (10 char width)
  - Precise timestamp

9. Game Flow Control:

```
void runGame() {
    // ... initialization ...
    while(true) {
        if(_kbhit()) { // Input handling
            switch(tolower(_getch())) {
                case 'a': playerX -= 3; break; // Move left
                case 'd': playerX += 3; break; // Move right
                case ' ': fireBullet(); break; // Shoot
                case 27: return; // Exit
            }
        }
```

- **Key Controls:**
  - A/D: 3-character lateral movement
  - Space: Fire bullet
  - ESC: Return to menu

10. Menu System:

```
int main() {
    hideCursor();
    srand(time(0));

    while(true) {
        system("cls");
        cout << "\n\t=== MAIN MENU ===";
        cout << "\n\t1. New Game\n\t2. High Scores\n\t3. Exit";

        switch(_getch()) { // Instant key response
            case '1': runGame(); break;
            case '2': displayHighscores(); break;
            case '3': return 0;
        }
```

- **Features:**
  - Persistent menu
  - Instant key input (no Enter required)
  - High score viewing

Key Technical Features:

1. Console Optimization:

- Cursor position control
- Flicker-free updates
- Hidden cursor

2. Game Balance:

- Regular enemies: +10 points
- Bonus enemies: +25 points
- Miss penalty: -2 points

3. Data Persistence:

- Scores saved in scores.dat
- Timestamp precision to seconds

4. Visual Design:

- Unicode characters for better graphics
- Clean HUD layout
- Formatted score display

5. Performance:

- Fixed array for bullets
- Efficient collision checks
- 120ms frame delay (~8 FPS)

FAQ’s:

1. Player Movement

How does the player move left and right using keyboard input?

- The player moves left with A and right with D. Each key press adjusts playerPos by 2 units and redraws the player character (<^>) at the new position.

2. Enemy Generation

What triggers the creation of new enemies when old ones are destroyed?

- Enemies respawn automatically when they reach the bottom (enemyY > SCREEN_HEIGHT - 2) or are destroyed. generateEnemy() sets a new random X position.

3. Bullet Mechanics

How many bullets can be active at once, and how are they fired?

- Maximum 20 bullets can be active. Pressing the spacebar fires one bullet at a time, which travels upward until it exits the screen or hits an enemy

4. Scoring System

How many points are awarded for hitting regular vs. bonus enemies?

- Regular enemies: +10 points
- Bonus enemies: +20 points
- Missing enemies deducts 1 point.

5. Collision Detection

How does the game detect when a bullet hits an enemy?

- Checks if a bullet's (X, Y) coordinates overlap with an enemy's position range (enemyX to enemyX+2). On overlap, the enemy is destroyed.

6. High Score Handling

Where are high scores stored, and what information is saved with them?
Scores are saved to highscores.txt with:

- Player name
- Score value
- Timestamp (YYYY-MM-DD HH:MM:SS)

- Displayed in a formatted table via showHighScores().

7. Game Over Condition

What happens when the player collides with an enemy?

- Player-enemy collision triggers gameover(), which saves the score, displays a "Game Over" screen, and returns to the main menu.

8. Bonus Enemies

How does the game decide when to spawn a bonus enemy?

- Spawns every 5 consecutive hits (tracked by hitCount). Controlled by generateBonusEnemy().

9. Console Display

What is the purpose of the gotoxy() function in the game?

- gotoxy(x,y) moves the cursor to specific coordinates for precise text placement (e.g., borders, score display, player/enemy positions).

10. Menu System

How does the main menu allow navigation between game options?

- Uses _getch() to detect key presses:
- 1 starts the game
- 2 shows high scores
- 3 exits
- Runs in a loop until the user quits.