Space Shooter Game

- In my space shooter game, players earn points by shooting enemies, and as they score more points, the game's level increases (maximum 5 level).

- As the level increases, the speed of the enemy also increases.

- With each level, shooting an enemy causes another one to respawn, but if an enemy reaches the player or collides with them, the game ends.

Basic Requirements

I made this game in 5 different environment:

1 - On mkeykernel OS (Linux Kernel in console)

https://github.com/arjun024/mkeykernel

2 - On basekernel OS (Linux Kernel in graphics mode)

https://github.com/dthain/basekernel

3 - On Windows at Code Blocks in console mode(with C++)

4 - On Windows at Code Blocks in graphics mode/WinBGI (with C++)

5 - On OpenGL environment in graphics mode (with C++)

Pseudocodes

1 - mkeykernel OS (Linux Kernel in console)

```
// Function to initialize Interrupt Descriptor Table (idt_init)
// Function to initialize keyboard (kb_init)
// Function to clear screen (clear_screen)
// Function to display start screen (start_screen)
// Function to display end screen (end_screen)
// Function to draw game board (gameboard)
// Function to check for collision (checkCollision)
// Function to draw enemy (draw_enemy)
// Function to clear enemy (clear_enemy_up)
// Function to update bullets (update_bullets)
// Function to draw string at specified position (draw_strxy)
// Function to print integer at specified position (print_integer)
// Function to pause execution for a specified duration (sleep)

void game() // Main game loop
{
    gameboard(); // Draw initial game board
    while (true) // Infinite loop for game
    {
        if (checkCollision() == 1) // Check for collision
        {
            gameover = 1; // Set game over flag
        }
        if (gameover == 1) // Check if game over
        {
            break; // Exit loop if game over
        }
        if (didStart == 1) // Check if game has started
        {
            if (enemyY > 25) // Check if enemy passes the player
            {
                gameover = 1; // Set game over flag
```

```
            break; // Exit loop
        }
        if (random_arr_index >= 99) // Check if random array index reached threshold
        {
            random_arr_length += 100; // Increase random array length
        }

        draw_strxy("Score: ", 1, 2); // Draw score label
        print_integer(score, 0x02); // Print score
        kprint_newline(); // Print new line
        kprint_newline(); // Print new line
        draw_strxy("Level: ", 1, 3); // Draw level label
        print_integer(level, 0x02); // Print level

        enemyY = enemyY + 1; // Move enemy down
        draw_enemy(); // Draw enemy
        clear_enemy_up(); // Clear enemy from previous position

        // Set sleep duration based on score to control game speed
        if (score >= 50)
            sleep(65000000);
        else if (score >= 30)
            sleep(70000000);
        else if (score >= 20)
            sleep(75000000);
        else if (score >= 10)
            sleep(80000000);
        else
            sleep(85000000);

        level = (score >= 50) ? 5 : (score >= 30) ? 4 : (score >= 20) ? 3 : (score >= 10) ? 2 : 1; // Set level based on score

        update_bullets(); // Update bullets position
    }
```

```
    }
    clear_screen(); // Clear screen after game ends
}

void kmain()
{
    idt_init(); // Initialize Interrupt Descriptor Table
    kb_init();  // Initialize keyboard
    clear_screen(); // Clear screen
    start_screen(); // Display start screen
    game(); // Start the game
    end_screen(); // Display end screen
}
```

2 - basekernel OS (Linux Kernel in graphics mode)

```
// Function to create root graphics object
graphics *g = graphics_create_root();

// Function to initialize console with provided graphics object
console_init(g);

// Function to add reference to console root
console_addref(&console_root);

// Function to initialize paging
page_init();

// Function to initialize kernel memory allocation
kmalloc_init((char *)KMALLOC_START, KMALLOC_LENGTH);

// Function to initialize interrupts
interrupt_init();
```

```
// Function to initialize real-time clock
rtc_init();
// Function to initialize keyboard
keyboard_init();

// Function to initialize processes
process_init();

// Set initial position and direction of player
player.x = (SCREEN_WIDTH - player_size) / 2;
player.y = SCREEN_HEIGHT - player_size - 1;
player.directions = NONE;

// Set initial position and direction of enemy
enemy.x = simple_rand() % (SCREEN_WIDTH - ENEMY_WIDTH);
enemy.y = 0;
enemy.directions = NONE;

// Display start screen
start_screen(g);

// Wait for any key press to start the game
keyboard_read(0);

// Main game loop
while (true) {
    // Clear screen
    clear_screen(g);

    // Handle keyboard input
    keyboard_handler_main();

    // Move player if there's active key input
    if (activeKey == 1) {
```

```
    move_player();
    activeKey = 0; // Reset activeKey flag
}

// Move enemy
move_enemy(g);

// Draw player, enemy, and bullet
draw_player(g);
draw_enemy(g);
draw_bullet(g);

// Move bullet
move_bullet();

// Check for collisions
check_collision(g);

// Print score and level
print_score(g);
print_score_Integer(g);
print_level(g);
print_level_Integer(g);

// Update enemy speed and level based on score
if (score >= 50) {
    enemy_speed = 10;
    level = 5;
} else if (score >= 30) {
    enemy_speed = 9;
    level = 4;
} else if (score >= 20) {
    enemy_speed = 8;
    level = 3;
```

```
    } else if (score >= 10) {
        enemy_speed = 7;
        level = 2;
    } else {
        enemy_speed = 6;
        level = 1;
    }

    // Pause execution for a short duration
    sleep(15);
}

// End of main function
return 0;
```

3 - Windows console mode

```
// Game is created with Draw(), Input() and Logic() functions
//  Draw() draws the enemy player and bullet with for loops
// Input() takes input from keyboard
// Logic() checks collisions and increases speed of enemy

// Variable declaration
char restartChoice;

// Main loop for restarting the game
do {
    // Setup game
    Setup();

    // Game loop
    while (!gameOver) {
        // Draw game state
        Draw();
```

```
        // Handle player input
        Input();

        // Update game logic
        Logic();

        // Pause for a short duration
        Sleep(50); // Adjust this value as needed
    }

    // Clear screen
    system("cls");

    // Print game over message and final score
    cout << "Game Over!" << endl;
    cout << "Your Final Score: " << score << endl;
    cout << "Press any key to play again . . .";

    // Get restart choice from user
    restartChoice = _getch();

    // Clear screen
    system("cls");

    // Reset game state if player chooses to restart
    gameOver = false;
    score = 0;
    difficulty = 5;
    enemyMoveCounter = 0;

} while (true); // End of main loop

// End of main function
```

```
return 0;
```

4 - Windows graphics mode/WinBGI

```
// Checks keyboard inside of the main game loop
// Function to initialize the game
void initialize() {
    // Initialize graphics window
    initwindow(WIDTH, HEIGHT, "Space Shooter");

    // Initialize player position
    player.x = WIDTH / 2;
    player.y = HEIGHT - 50;

    // Deactivate enemy
    enemy.active = false;

    // Seed the random number generator
    srand(time(NULL));

    // Initialize bullets
    for (int i = 0; i < 10; i++) {
        bullets[i].active = false;
    }
    // Display instructions
    setcolor(WHITE);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2); // Font size for instructions
    outtextxy(50, HEIGHT / 2 - 50, "Press SPACE to shoot.");
    outtextxy(50, HEIGHT / 2, "Press A to move left, D to move right.");
    outtextxy(50, HEIGHT / 2 + 50, "Press any key to start the game.");
    getch(); // Wait for any key press to start the game
    cleardevice(); // Clear the screen before starting the game
}

// Main function
```

```
int main() {
    initialize(); // Initialize the game
    // Main game loop
    while (true) {
        cleardevice(); // Clear the screen


        // Handle keyboard input
        if (kbhit()) {
            char key = getch();
            if (key == 'a' && player.x - ((SHIP_SIZE/2) + 20) > 0) { // Check if moving left won't go out of the screen
                player.x -= 10;
            }
            if (key == 'd' && player.x + ((SHIP_SIZE/2) + 20) < WIDTH) { // Check if moving right won't go out of the screen
                player.x += 10;
            }
            if (key == ' ') {
                shootBullet();
            }
        }


        // Spawn enemy if not active
        if (!enemy.active) {
            spawnEnemy();
        }


        // Move bullets, enemy, check collisions, score, and draw game elements
        moveBullets();
        moveEnemy();
        checkCollisions();
        checkEnemyPassed(); // Check if enemy has passed the screen
        checkScore(); // Check the score for speed increase


        drawShip(player.x, player.y);
```

```
        for (int i = 0; i < 10; i++) {
            if (bullets[i].active) {
                drawBullet(bullets[i].x, bullets[i].y);
            }
        }
        if (enemy.active) {
            drawEnemy(enemy.x, enemy.y);
        }

        drawScore(); // Draw score
        delay(30); // Delay for smooth gameplay
    }

    closegraph(); // Close the graphics window
    return 0; // End of main function
}
```

5 - OpenGL graphics mode

```
// Define global variables
float playerX, playerY, playerSize;
float enemyX, enemyY, enemySize, enemySpeed;
int score, level;
bool gameOver, showInstructions;
vector<Bullet> bullets;

// Function to initialize OpenGL
void init() {
    // Initialize OpenGL settings
    // Set background color
    // Set projection matrix
}

// Function to draw a triangle
```

```
void drawTriangle(float x, float y, float size) {
    // Draw a triangle at given coordinates and size
}
// Function to draw a hexagon
void drawHexagon(float x, float y, float size) {
    // Draw a hexagon at given coordinates and size
}

// Function to draw the player
void drawPlayer() {
    // Draw the player ship
}

// Function to draw the enemy
void drawEnemy() {
    // Draw the enemy ship
}

// Function to draw a bullet
void drawBullet(float x, float y, float size) {
    // Draw a bullet at given coordinates and size
}

// Function to draw the score
void drawScore() {
    // Draw the score on the screen
}

// Function to draw the level
void drawLevel() {
    // Draw the level on the screen
}

// Function to draw the instructions
```

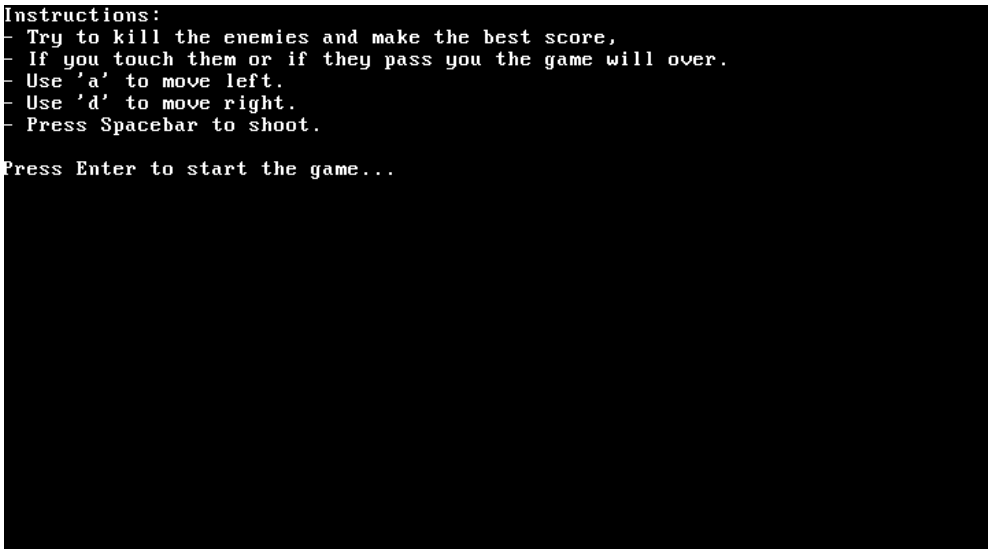
```
void drawInstructions() {
    // Draw game instructions on the screen
}

// Function to display the game
void display() {
    // Clear the screen
    // Check if instructions are to be displayed
    // Draw instructions if required
    // If game is not over:
        // Draw player, enemy, bullets, score, and level
        // Check collision between player and enemy
        // If enemy has passed the player or collided with player:
            // Set game over flag
    // If game is over:
        // Display game over message with final score
    // Swap buffers
}

// Function to update game state
void update(int value) {
    // If game is not over:
        // Update enemy position based on level and speed
        // Update bullet positions
        // Check collision between bullets and enemy
        // Check bounds for player and enemy
    // Post redisplay and set timer for next update
}
// Function to handle keyboard input
void handleKeypress(unsigned char key, int x, int y) {
    // If instructions are shown, hide them on any key press
    // Otherwise, handle key presses for player movement and shooting
}
// Main function
```

```
int main(int argc, char** argv) {
    // Initialize random seed
    // Initialize GLUT
    // Set display mode, window size, and position
    // Create window with title
    // Initialize OpenGL
    // Register display, update, and keyboard callback functions
    // Enter GLUT event processing loop
    // Return 0 to exit progra
```

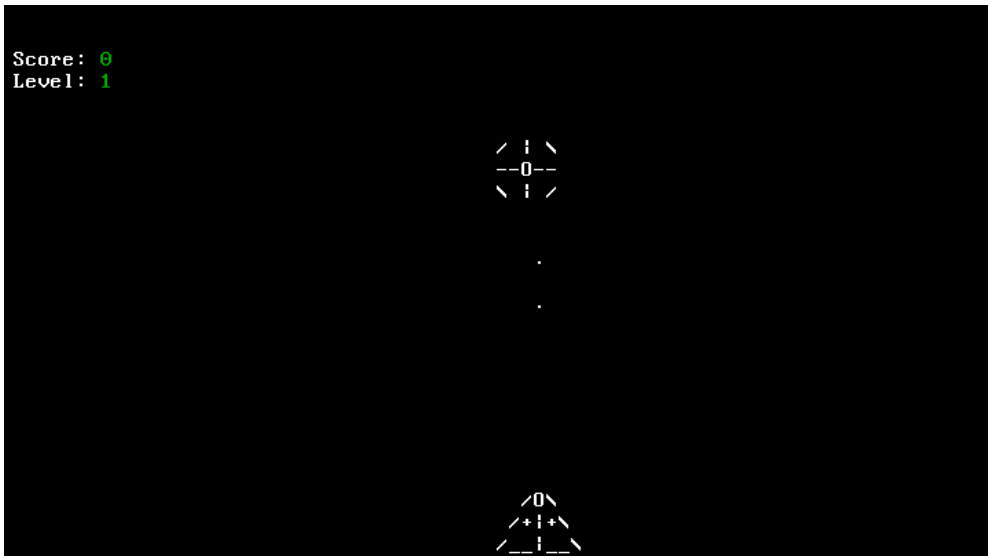
Demonstration

1 - mkeykernel OS (Linux Kernel in console)

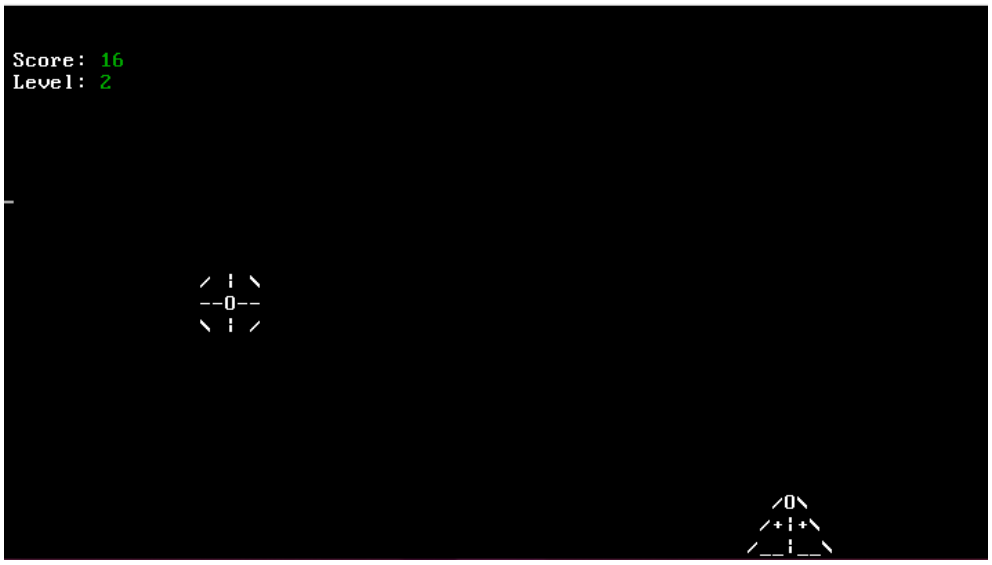
- Instructions



- Gameplay 1



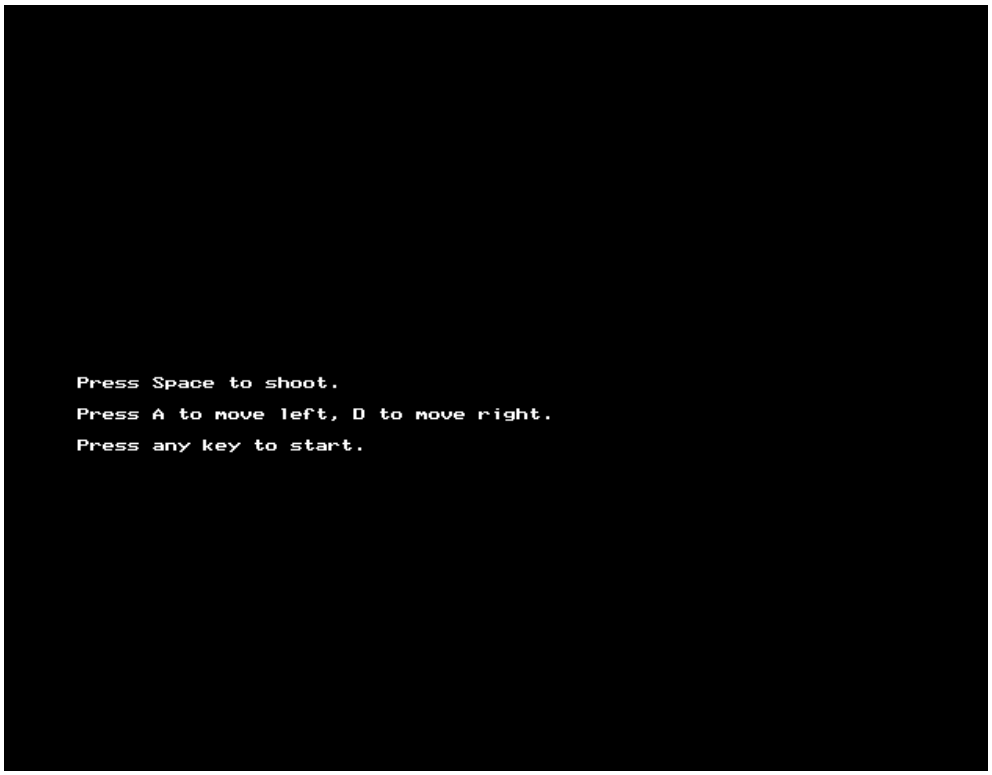
- Gameplay 2



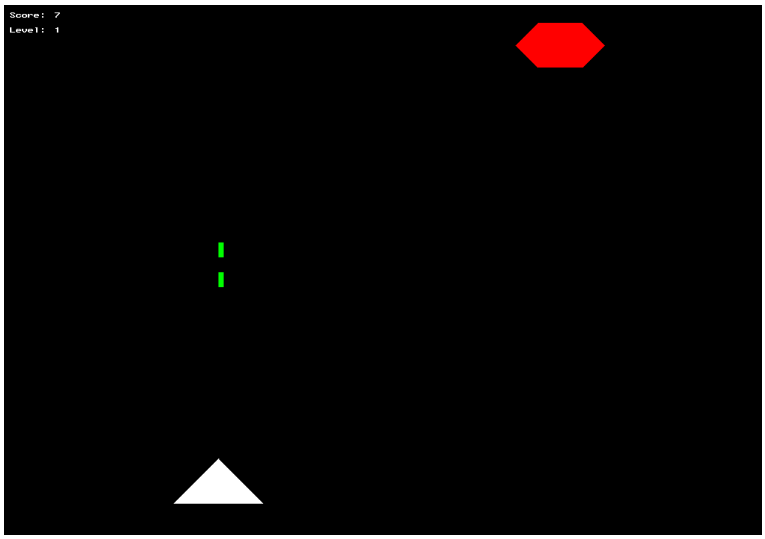
- End

2 - basekernel OS (Linux Kernel in graphics mode)

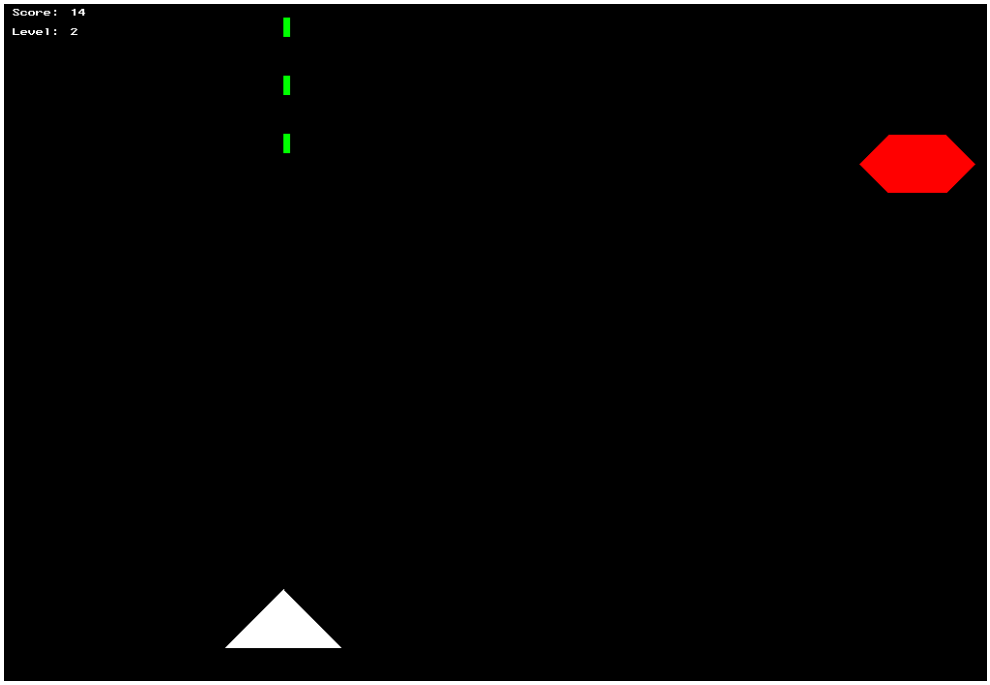
- Instructions



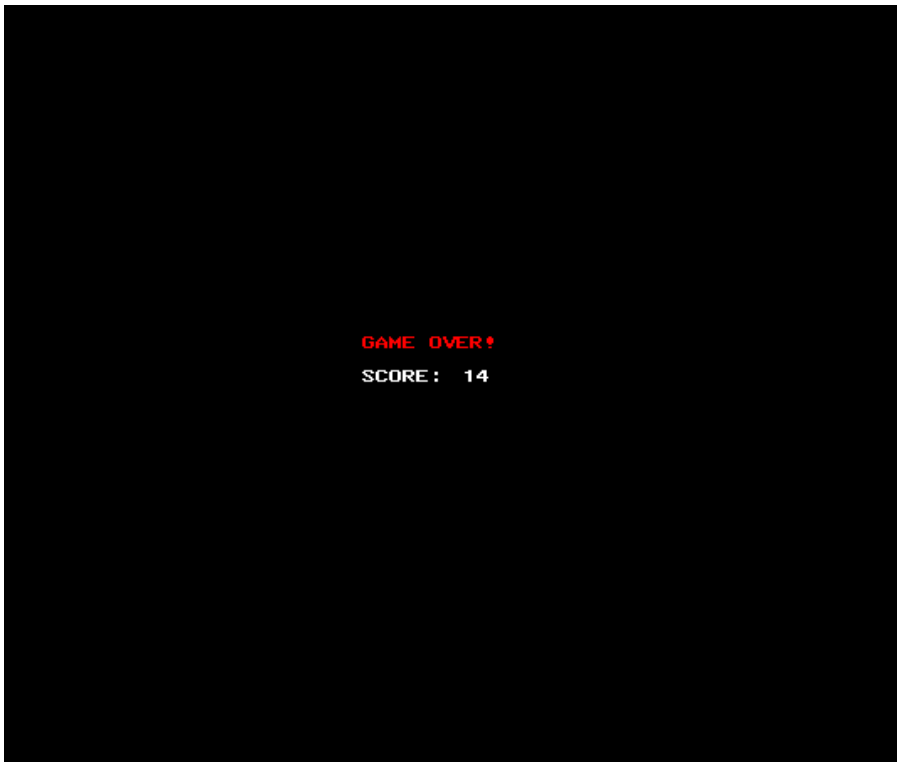
- Gameplay 1



- Gameplay 2

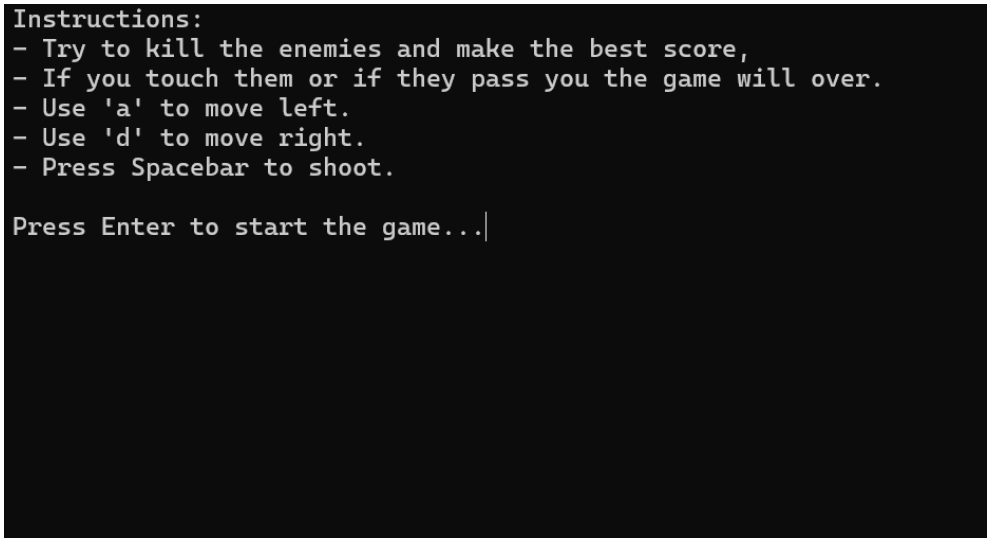


- End



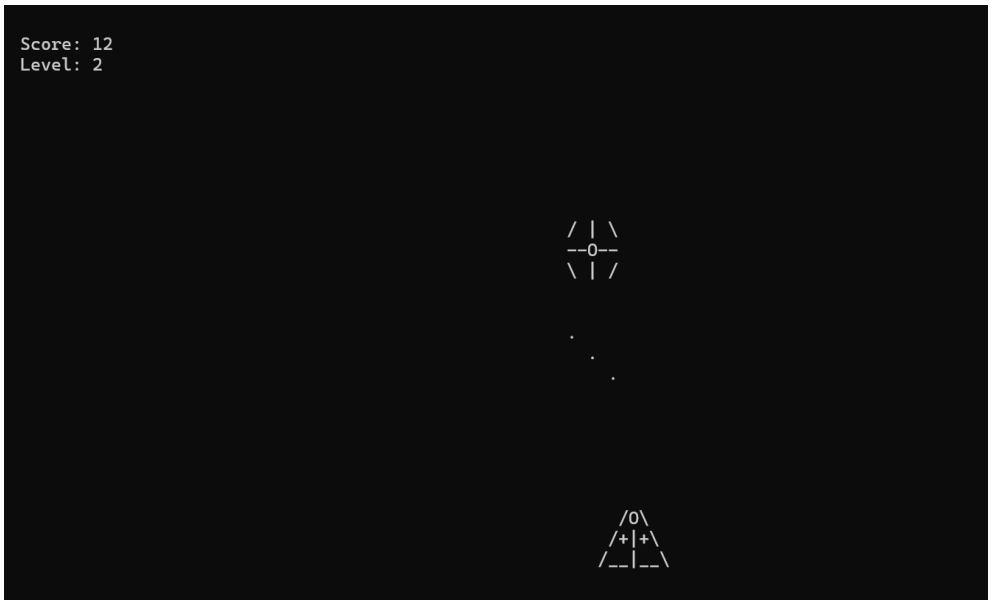
3 - Windows console mode

- Instructions

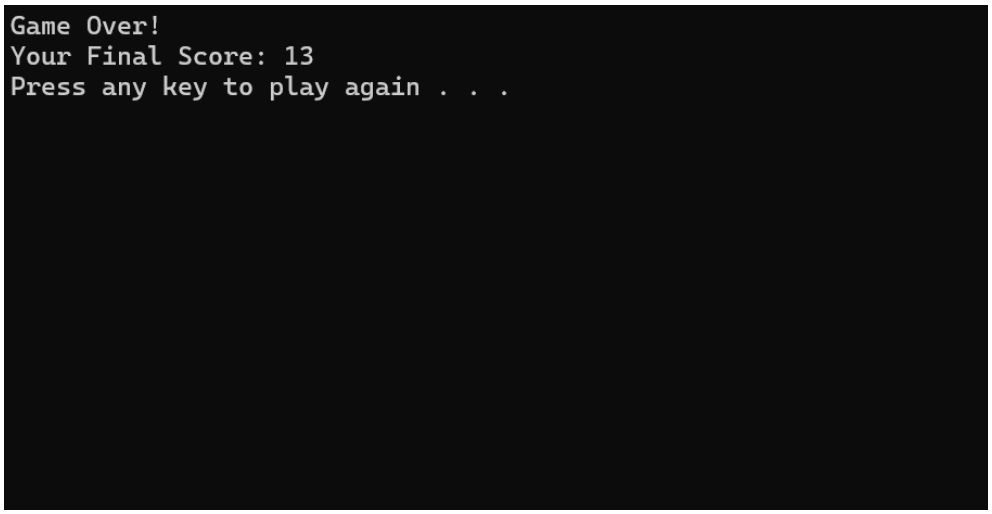


- Gameplay 1

- Gameplay 2

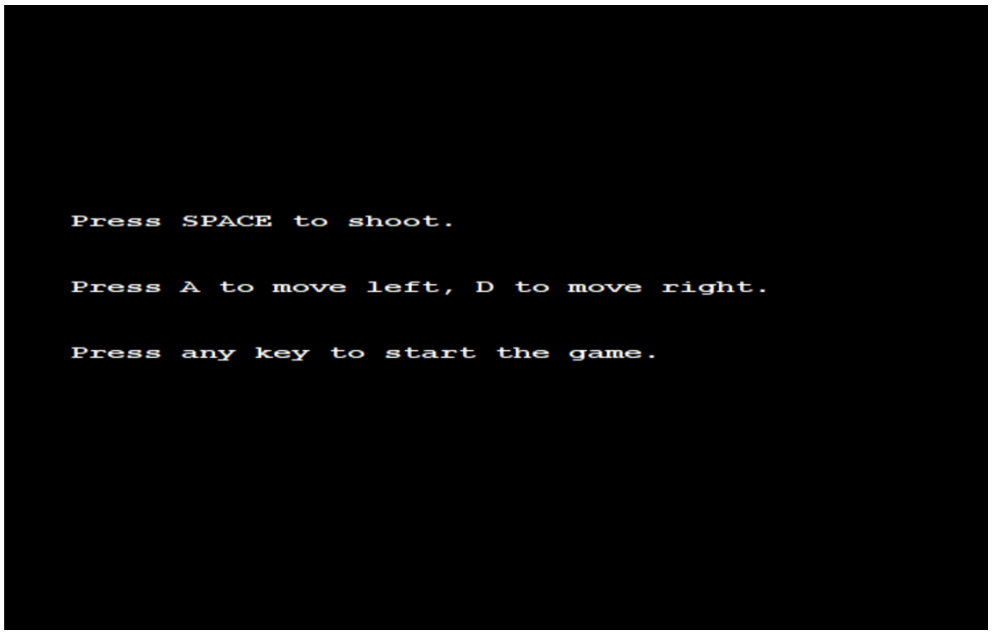


- End

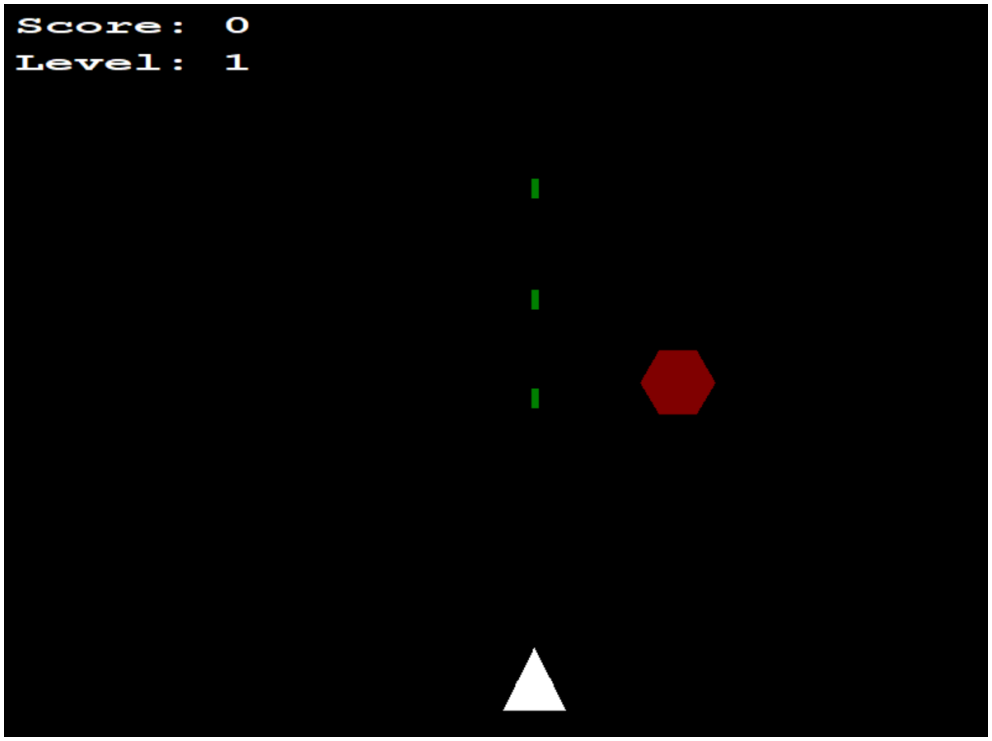


4 - Windows graphics mode/WinBGI

- Instructions



- Gameplay 1



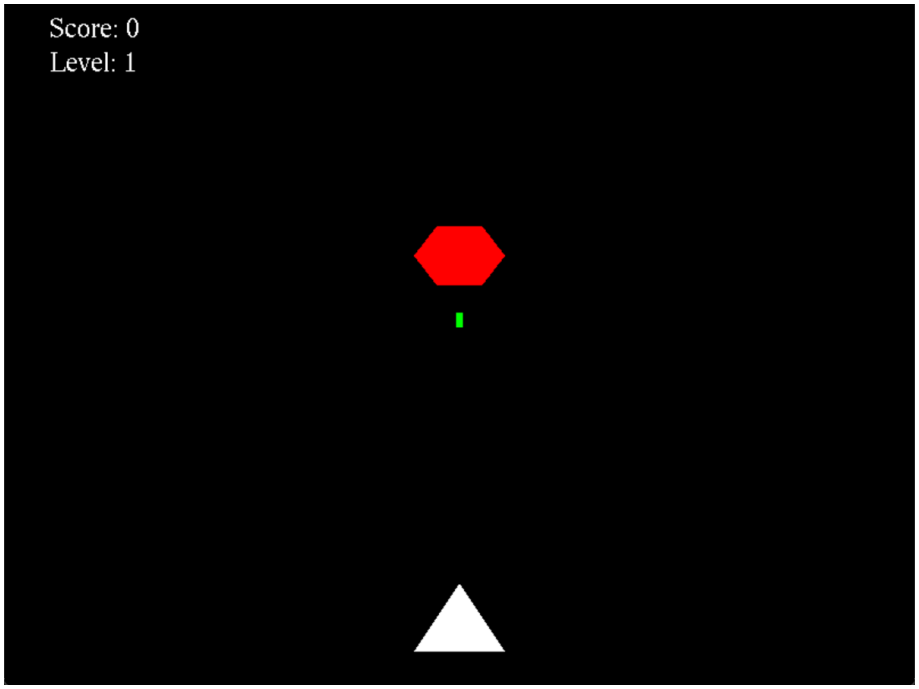
- Gameplay 2

- End

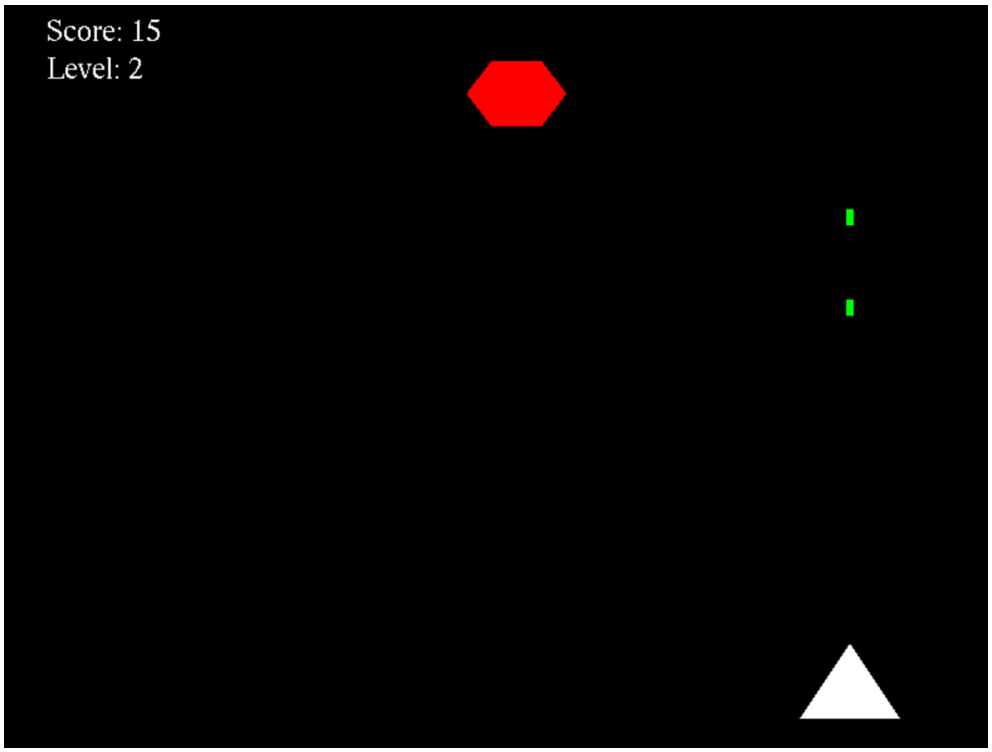
5 - OpenGL graphics mode

- Instructions

- Gameplay 1



- Gameplay 2



- End