# Armada Game Documentation

## Table of Contents



## Introduction

Armada is a 2D shooter game where the player controls a spaceship and must defeat enemies while avoiding collisions. The game includes multiple levels and various enemy types, with increasing difficulty as the player progresses.

## How to Play

### Objective

The objective of Armada is to survive as long as possible while shooting down enemy ships and avoiding their bullets.

### Controls

- **Movement**: Use the arrow keys (Up, Down, Left, Right) to move the player's ship.
- **Pause**: Press the Spacebar to pause the game at any time.

### Gameplay

1. **Movement**: Use the arrow keys to navigate your ship through the battlefield, dodging enemy bullets and collecting power-ups.
2. **Shooting**: The ship will automatically fire bullets. Destroying enemy ships will earn you points.
3. **Power-ups**: Occasionally, power-up items will appear. Collect these by colliding with them to gain benefits such as increased ammo capacity.

4. **Avoid Enemy Bullets**: Stay clear of enemy bullets to avoid taking damage. Colliding with enemy bullets or ships will result in game over.
5. **Score and Level**: Earn points by destroying enemy ships. Your score will increase with each enemy destroyed. Reach higher levels by accumulating enough points, but be aware that as the level increases, so does the difficulty.
6. **Game Over**: The game ends when your ship collides with an enemy ship or bullet. You can restart the game by clicking the "Play" label on the game over screen.

### Tips

- Keep moving to avoid enemy bullets.
- Prioritize shooting down enemies to increase your score and level.
- Collect power-ups whenever possible to gain advantages.

## Game Setup

### Game Load

The `Armada_Load` method initializes the game by setting up initial values, loading assets, configuring enemies, bullets, stars, and audio. It also starts the background music.

```
private void Armada_Load(object sender, EventArgs e)
{
    // Initialize game values and load assets
    bgSpeed = 5;
    playerSpeed = 5;
    enemySpeed = 5;
    enemyBulletSpeed = 5;
    ammoSpeed = 20;
    ammo = new PictureBox[1];

    // Initialize default game state
    pause = false;
    isGameOver = false;
    score = 0;
    level = 1;
    difficulty = 9;

    // StartTimers(); // Commented out

    // Load images for the game
    Image bullet = Image.FromFile(@"assets\bullet.png");
    Image boss1 = Image.FromFile(@"assets\boss1.png");
    Image enemy1 = Image.FromFile(@"assets\enemy1.png");
    Image enemy2 = Image.FromFile(@"assets\enemy2.png");
    Image enemy3 = Image.FromFile(@"assets\enemy3.png");
    Image powerup1 = Image.FromFile(@"assets\star.gif");
    Image enemy5 = Image.FromFile(@"assets\enemy5.png");
    Image enemy6 = Image.FromFile(@"assets\enemy6.png");
    Image enemy7 = Image.FromFile(@"assets\enemy7.png");
    Image enemy8 = Image.FromFile(@"assets\enemy8.png");
    Image boss2 = Image.FromFile(@"assets\boss2.png");
```

```
enemies = new PictureBox[10];

// Initialize enemy picture boxes
for (int i = 0; i < enemies.Length; i++)
{
    enemies[i] = new PictureBox();
    enemies[i].Size = new Size(50, 50);
    enemies[i].SizeMode = PictureBoxSizeMode.Zoom;
    enemies[i].BorderStyle = BorderStyle.None;
    enemies[i].BackColor = Color.Transparent;
    enemies[i].ForeColor = Color.Transparent;
    enemies[i].Visible = false;
    this.Controls.Add(enemies[i]);
    enemies[i].Location = new Point((i + 1) * 50, -50);
}

// Assign images to enemies
enemies[0].Image = boss1;
enemies[1].Image = enemy1;
enemies[2].Image = enemy2;
enemies[3].Image = enemy3;
enemies[4].Image = powerup1;
enemies[5].Image = enemy5;
enemies[6].Image = enemy6;
enemies[7].Image = enemy7;
enemies[8].Image = enemy8;
enemies[9].Image = boss2;

// Initialize ammo
for (int i = 0; i < ammo.Length; i++)
{
    ammo[i] = new PictureBox();
    ammo[i].Size = new Size(8, 8);
    ammo[i].Image = bullet;
    ammo[i].SizeMode = PictureBoxSizeMode.Zoom;
    ammo[i].BorderStyle = BorderStyle.None;
    this.Controls.Add(ammo[i]);
}

// Windows Media Player Initialization
bgAudio = new WindowsMediaPlayer();
shootAudio = new WindowsMediaPlayer();
boomAudio = new WindowsMediaPlayer();
powerUpAudio = new WindowsMediaPlayer();

// Load audio files
bgAudio.URL = @"audio\background.mp3";
shootAudio.URL = @"audio\shoot.mp3";
boomAudio.URL = @"audio\boom.mp3";
powerUpAudio.URL = @"audio\powerup.mp3";

// Setup audio settings
bgAudio.settings.setMode("loop", true);
bgAudio.settings.volume = 9;
shootAudio.settings.volume = 7;
```

```
boomAudio.settings.volume = 10;
powerUpAudio.settings.volume = 10;

stars = new PictureBox[10];
random = new Random();
```

# Game Components

## Assets

Assets include images for bullets, enemies, and power-ups which are loaded from the specified file paths.

```
// Load images for the game
Image bullet = Image.FromFile(@"assets\bullet.png");
Image boss1 = Image.FromFile(@"assets\boss1.png");
Image enemy1 = Image.FromFile(@"assets\enemy1.png");
Image enemy2 = Image.FromFile(@"assets\enemy2.png");
Image enemy3 = Image.FromFile(@"assets\enemy3.png");
Image powerup1 = Image.FromFile(@"assets\star.gif");
Image enemy5 = Image.FromFile(@"assets\enemy5.png");
Image enemy6 = Image.FromFile(@"assets\enemy6.png");
Image enemy7 = Image.FromFile(@"assets\enemy7.png");
Image enemy8 = Image.FromFile(@"assets\enemy8.png");
Image boss2 = Image.FromFile(@"assets\boss2.png");

enemies = new PictureBox[10];

// Initialize enemy picture boxes
for (int i = 0; i < enemies.Length; i++)
{
    enemies[i] = new PictureBox();
    enemies[i].Size = new Size(50, 50);
    enemies[i].SizeMode = PictureBoxSizeMode.Zoom;
    enemies[i].BorderStyle = BorderStyle.None;
    enemies[i].BackColor = Color.Transparent;
    enemies[i].ForeColor = Color.Transparent;
    enemies[i].Visible = false;
    this.Controls.Add(enemies[i]);
    enemies[i].Location = new Point((i + 1) * 50, -50);
}
```

## Audio

Audio components are initialized using the `WindowsMediaPlayer` and include background music and sound effects for shooting, explosions, and power-ups.

```
// Windows Media Player Initialization
bgAudio = new WindowsMediaPlayer();
shootAudio = new WindowsMediaPlayer();
boomAudio = new WindowsMediaPlayer();
powerUpAudio = new WindowsMediaPlayer();
```

```
// Load audio files
bgAudio.URL = @"audio\background.mp3";
shootAudio.URL = @"audio\shoot.mp3";
boomAudio.URL = @"audio\boom.mp3";
powerUpAudio.URL = @"audio\powerup.mp3";

// Setup audio settings
bgAudio.settings.setMode("loop", true);
bgAudio.settings.volume = 9;
shootAudio.settings.volume = 7;
boomAudio.settings.volume = 10;
powerUpAudio.settings.volume = 10;
```

# Game Logic

## Movement

Player and enemy movements are handled through timers and key events. Timers control the automatic movement, while key events handle player input.

## Timers

Timers are used to update the game state periodically, such as moving the background, player, enemies, and bullets.

### Background Movement

Moves the background stars to create a scrolling effect.

```
private void MoveBgTimer_Tick(object sender, EventArgs e)
{
    // Move background stars with different speeds for a more dynamic effect
    for (int i = 0; i < stars.Length / 2; i++)
    {
        stars[i].Top += bgSpeed;
        if (stars[i].Top >= this.Height)
        {
            stars[i].Top = -stars[i].Height;
        }
    }

    for (int i = stars.Length / 2; i < stars.Length; i++)
    {
        stars[i].Top += bgSpeed * 2;  // Using bgSpeed directly instead of assigning again
        if (stars[i].Top >= this.Height)
        {
            stars[i].Top = -stars[i].Height;
        }
    }
}
```

### Player Movement

Handles player movement based on timer ticks and key presses.

```
// Move player based on key presses (assuming Left, Top, Right, and Bottom properties exist)
if (Player.Left > 10 && e.KeyCode == Keys.Left)
{
    Player.Left -= playerSpeed;
}

if (Player.Top > 10 && e.KeyCode == Keys.Up)
{
    Player.Top -= playerSpeed;
}

if (Player.Right < 600 && e.KeyCode == Keys.Right)
{
    Player.Left += playerSpeed; // Using Left property to move player right might be a typo, consider using Rig
}

if (Player.Bottom < 600 && e.KeyCode == Keys.Down)
{
    Player.Top += playerSpeed;
}
```

### Bullet Movement

Moves bullets upwards and checks for collisions.

```
private void BulletMvTimer_Tick(object sender, EventArgs e)
{
    // Move bullets and handle collisions
    shootAudio.controls.play();
    for (int i = 0; i < ammo.Length; i++)
    {
        if (ammo[i].Top > 0)
        {
            ammo[i].Visible = true;
            ammo[i].Top -= ammoSpeed;

            Collision();
        }
        else
        {
            ammo[i].Visible = false;
            ammo[i].Location = new Point(Player.Location.X + 20, Player.Location.Y - i * 30);
        }
    }
}
```

### Enemy Movement

Moves enemies down the screen and resets their positions if they move off-screen.

```
private void MoveEnemies(PictureBox[] array, int speed)
{
    // Move enemies and reset their positions if needed
    for (int i = 0; i < array.Length; i++)
    {
        if (!array[i].Visible) // Check if enemy is already visible
        {
            // Set random location based on form size
            array[i].Location = new Point(random.Next(-50, this.ClientSize.Width + 50), random.Next(-200, -50)
            array[i].Visible = true;
        }
        else
        {
            array[i].Top += speed;

            // Check if the enemy is outside form
            if (array[i].Top > this.ClientSize.Height)
            {
                // Reset enemy position if it is outside the form
                array[i].Location = new Point(random.Next(-50, this.ClientSize.Width + 50), random.Next(-200,
            }
        }
    }
}
```

## Collisions

Collision detection between bullets and enemies, and between the player and enemies or bullets.

```
private void Collision()
{
    // Handle collisions between bullets and enemies
    for (int i = 0; i < enemies.Length; i++)
    {
        bool bulletCollided = false;

        // Check collision for each bullet with the enemy
        for (int j = 0; j < ammo.Length; j++)
        {
            if (ammo[j].Bounds.IntersectsWith(enemies[i].Bounds))
            {
                if (i != 4)
                {
                    boomAudio.controls.play();

                    score += 1;
                    labelScore.Text = (score < 10) ? "Score: " + score.ToString() : "Score: " + score.ToString(

                    if (score % 30 == 0)
                    {
```

```
                    level += 1;
                    labelLevel.Text = (level < 10) ? "Level: 0" + level.ToString() : "Level: 0" + level.To

                    if (enemySpeed <= 10 && enemyBulletSpeed <= 10 && difficulty >= 0)
                    {
                        difficulty--;
                        enemySpeed++;
                        enemyBulletSpeed++;
                    }

                    if (level == 10)
                    {
                        GameOver("Well Done!");
                    }
                }

                enemies[i].Location = new Point((i + 1) * 50, -100);
                bulletCollided = true;
                break;
            }
            else
            {
                bulletCollided = true;
                break;
            }
        }
    }

    if (bulletCollided)
    {
        continue;
    }

    if (Player.Bounds.IntersectsWith(enemies[i].Bounds))
    {
        // Check if the picture box is the powerup
        if (i == 4)
        {
            IncreaseAmmo();
            powerUpAudio.controls.play();
            enemies[i].Visible = false;
            enemies[i].Location = new Point((i + 1) * 50, -100);
        }
        else
        {
            boomAudio.settings.volume = 30;
            boomAudio.controls.play();
            Player.Visible = false;
            GameOver("Game Over!");
        }
    }
}
}
```

## Collision With Enemy Bullet

Handles collisions between enemy bullets and the player.

```
private void CollisionWithEnemyBullet()
{
    // Handle collisions between enemy bullets and the player
    for (int i = 0; i < enemyBullet.Length; i++)
    {
        if (enemyBullet[i].Bounds.IntersectsWith(Player.Bounds))
        {
            enemyBullet[i].Visible = false;
            boomAudio.settings.volume = 30;
            boomAudio.controls.play();
            Player.Visible = false;
            GameOver("Game Over!");
        }
    }
}
```

### Game Over and Restart

Handles the game over state, stops the game, and provides options to restart or exit.

```
private void GameOver(String prompt)
{
    // Display game over prompt and stop the game
    label1.Text = prompt;
    label1.Visible = true;
    labelPlay.Visible = true;
    labelExit.Visible = true;
    labelHelp.Visible = true;

    bgAudio.controls.stop();
    StopTimers();
}
```

## Other Methods Used

### Increase Ammo

Increases the player's ammo when a power-up is collected.

```
private void IncreaseAmmo()
{
    // Increase the amount of player ammo
    int currentLength = ammo.Length;
    Array.Resize(ref ammo, currentLength + 1);
    ammo[currentLength] = new PictureBox();
    ammo[currentLength].Size = new Size(8, 8);
    ammo[currentLength].Image = Image.FromFile(@"assets\bullet.png");
```

```
    ammo[currentLength].SizeMode = PictureBoxSizeMode.Zoom;
    ammo[currentLength].BorderStyle = BorderStyle.None;
    this.Controls.Add(ammo[currentLength]);
}
```

## Game Over

Displays the game over prompt and stops all game activities.

```
private void GameOver(string prompt)
{
    // Display game over prompt and stop the game
    label1.Text = prompt;
    label1.Visible = true;
    labelPlay.Visible = true;
    labelExit.Visible = true;
    labelHelp.Visible = true;

    bgAudio.controls.stop();
    StopTimers();
}
```

## Stop Timers

Stops all game timers.

```
private void StopTimers()
{
    // Stop all game timers
    BgMvTimer.Stop();
    EnemyMvTimer.Stop();
    BulletMvTimer.Stop();
    EnemyBulletTimer.Stop();
}
```

## Start Timers

Starts all game timers.

```
private void StartTimers()
{
    // Start all game timers
    BgMvTimer.Start();
    EnemyMvTimer.Start();
    BulletMvTimer.Start();
    EnemyBulletTimer.Start();
}
```

## Play Label Click

Restarts the game when the "Play" label is clicked.

```
private void labelPlay_Click(object sender, EventArgs e)
{
    // Restart the game when Play label is clicked
    this.Controls.Clear();
    InitializeComponent();
    pictureBoxWelcome.Visible = false;
    Armada_Load(e, e);
    StartTimers();
}
```

### Welcome Game

Displays the welcome screen.

```
private void WelcomeGame()
{
    // Show welcome screen
    pictureBoxWelcome.Visible = true;
    labelPlay.Visible = true;
    labelExit.Visible = true;
    labelHelp.Visible = true;
}
```

## Key Event Handlers

### Key Down

Handles key press events to start player movement and pause functionality.

```
private void Armada_KeyDown(object sender, KeyEventArgs e)
{
    // Handle key press events
    if (!pause)
    {
        if (e.KeyCode == Keys.Right)
        {
            RightMvTimer.Start();
        }
        else if (e.KeyCode == Keys.Left)
        {
            LeftMvTimer.Start();
        }
        else if (e.KeyCode == Keys.Up)
        {
            UpMvTimer.Start();
        }
        else if (e.KeyCode == Keys.Down)
        {
```

```
            DownMvTimer.Start();
        }
    }
}
```

## Key Up

Handles key release events to stop player movement and toggle pause state.

```
private void Armada_KeyUp(object sender, KeyEventArgs e)
{
    // Handle key release events
    RightMvTimer.Stop();
    LeftMvTimer.Stop();
    DownMvTimer.Stop();
    UpMvTimer.Stop();

    if (e.KeyCode == Keys.Space)
    {
        if (!isGameOver)
        {
            if (pause)
            {
                StartTimers();
                label1.Visible = false;
                labelHelp.Visible = false;
                panelHelp.Visible = false;
                bgAudio.controls.play();
                pause = false;
            }
            else
            {
                label1.Text = "PAUSED";
                label1.Visible = true;
                labelHelp.Visible = true;
                bgAudio.controls.pause();
                StopTimers();
                pause = true;
            }
        }
    }
}
```