Space Defender Game Developer Documentation

Overview

Space Defender is a 2D space shooter game developed using Windows Forms in C#. This document provides an in-depth explanation of the game's architecture, coding practices, and implementation details to aid developers in understanding and enhancing the project. The modular design supports extensibility, enabling future additions like new enemy types or gameplay features.

Project Structure

Namespaces

SpaceDefender: The primary namespace containing all classes and interfaces.

Key Components

Form: FormMain – The main window in which the game is played.

Classes:

- Ship: Represents the ship that the player controls during the game.
- Projectile: Represents a projectile in the game that can move and eliminate aliens.
- Alien: Abstract class which is used to implement all the different types of aliens be it bosses or just normal aliens.
- AlienType1 (Green alien) inherits from Alien: A normal alien which has a normal speed and downwards movement pattern.
- AlienType2 (Yellow alien) inherits from Alien: A normal alien which has a fast speed and downwards movement pattern.
- AlienType3 (Red alien) inherits from Alien: A normal alien which has a normal speed and strafing movement pattern.
- AlienBoss1(Grey alien) inherits from Alien: A boss alien which has a slow speed but a lot of health-points and downwards movement pattern. Can be reborn with more health-points
- AlienBoss2 (Purple alien) inherits from Alien: A boss alien which has a slow speed and teleporting movement pattern. Can be reborn with more health-points

- AlienBoss3 (Blue alien) inherits from Alien: A boss alien which has a slow speed but when shot, spawns 1 green alien, and with a downwards movement pattern. Can be reborn with less health-points, but spawns 2 instead of 1 green alien when shot.

- AlienUpdater: Implements the task of updating the aliens on the screen each tick, to move them downwards or to remove them from the screen and to play their death animation.
- BackgroundUpdater: Implements the task of updating the background of the screen each tick, by moving it downwards and looping the background picture.
- ProjectileUpdater: Implements the task of updating the movement and state of the projectiles on the screen each tick, by moving them upwards.
- CollisionManager: A class responsible for managing the detection between projectiles and aliens within the game, as well as to update the grid that tracks their positions for collision detection.
- AudioManager: Manages the playing of audio/music in the whole application, using WaveStreams. Makes use of NAudio, which is a NuGet package.
- LoopStream inherits from WaveStream: A custom implementation of the WaveStream class to provide looping of audio files once they reach the end of the song. Used for music (such as in the main menu).
- AudioManagerHelpers: A helper class for the AudioManager class. Used to have cleaner code – puts the MusicPlayer in a separate class to have it be static.
- WaveManager: Manages the waves of enemies in the game, including the configuration and progression of waves.
- Wave: Represents a wave of enemies in the game.

- Program: The main entry point for the application.
- GameController: Manages the overall game logic, including player actions, game state, and interactions between game entities.
- FormMain inherits from Form: The main window of our program, where we initialize custom elements and our window. Also contains the listeners for actions such as clicking buttons.
- FormMain.Designer: The designer class used to design the GUI. Initializes all of the GUI components.

- GlobalUtilsGUI: Contains utility constants and settings related to the graphical user interface (GUI), including color definitions and audio settings. It also defines a

custom slider control class used for volume or other settings, alongside the double buffered panels needed for smoother animations.

- GlobalVariablesGame: Stores global variables and settings related to the game, including difficulty level, wave count, and game status.

Implementation Details

Game Initialization

-File: Program.cs, FormMain.cs, FormMain.Designer.cs

-Purpose: Sets up the main window, initializes game components, and handles navigation to the game.

-Key Methods:

Application.Run(Form FormMain) – Starts the GUI.

LoadFontFromFile() – Loads the custom font from a file.

InitializeComponent() – Initializes the GUI components.

InitCustomSliders() – Initializes the custom slider for the Audio submenu.

Spaceship Mechanics

-File: Ship.cs

-Purpose: Handles player control and shooting mechanics.

-Key Methods:

MoveLeft() – Moves the position of the ship to the left.

MoveRight() – Moves the position of the ship to the right.

ResetShip() – Resets the image of the ship to its stationary status.

Projectile Logic

-File: Projectile.cs, ProjectileUpdater.cs

-Purpose: Manages the behavior and movement of projectiles.

-Key Methods:

Move(): Moves the projectile's position upwards.

StartProjectileUpdateTask(Panel gamePanelMain): Starts the task of updating the projectiles.

UpdateProjectiles(): Moves projectiles upwards and checks if they should still be rendered.

Enemy and Boss Behaviors

-File: Alien.cs, AlienUpdater.cs

-Purpose: Implements unique behaviors for each alien type and bosses, also managing their movement and animation updates.

-Key Methods:

Move(): Defines movement patterns for aliens, such as straight descent, strafing, or teleporting.

StartAlienUpdateTask(Panel gamePanelMain): Starts the task of updating the aliens.

UpdateAliens(): Updates the aliens' movement, checking if they should still be rendered too.

Wave Management

-File: WaveManager.cs

-Purpose: Controls wave progression and enemy spawning.

-Key Methods:

GetCurrentWaveAlienTypeCount(int alienType): Gets the current amount of aliens left in the current wave.

DecAlienTypeCount(int alienType): Removes an alien from the list of aliens left in the wave (when it gets eliminated).

Collision Handling

-File: CollisionManager.cs

-Purpose: Detects and manages collisions between projectiles and aliens.

-Key Methods:

CheckCollisions(Panel gamePanelMain, List<Projectile> projectiles, List<Alien> aliens): Identifies collisions and updates the game state accordingly.

UpdateGrid(List<Projectile> projectiles, List<Alien> aliens): Keeps track of entity positions for optimized collision detection.

Audio Management

-File: AudioManager.cs, AudioManagerHelpers.cs

-Purpose: Handles in-game audio, including music and sound effects.

-Key Methods:

PlayMusic(UnmanagedMemoryStream musicStreamData, string musicName, bool loopPlayer = false): Loops background music using the LoopStream class.

PlaySfx(): Plays specific sound effects for actions like shooting, explosions, or alien death.

Dynamic Background

-File: BackgroundUpdater.cs

-Purpose: Creates a scrolling background effect to simulate movement through space.

-Key Methods:

StartBackgroundUpdateTask(Panel gamePanelMain): Starts the background updating task.

UpdateBackground(Panel gamePanelMain): Updates the background by looping it and moving it upwards.

Game Controller

-File: GameController.cs

-Purpose: Manages the overall game logic, including player actions and interactions between game entities.

-Key Methods:

StartGameScreen(Panel gamePanelMain, Label gameLabelStart, Button gameButtonMenu, Label gameLabelWave, Action onGameEnd): Starts the game screen and initializes the game state. Also acts to create the fading transition between starting and starting to play.

InitializeGameTimers(Panel gamePanelMain, Label gameLabelWave, Label gameLabelStart, Action onGameEnd): Initializes the game timer, to update the game state and objects for each tick.

Global Utilities

File: GlobalUtilsGUI.cs, GlobalVariablesGame.cs

Purpose: Stores global settings and utility functions.

Key Components:

GUI settings for sliders and animations.

Game settings such as difficulty level, wave count, and game status.

Also implements utility classes for the double buffered panels and custom audio sliders.

Graphical User Interface

File: FormMain.cs, FormMain.Designer.cs

Purpose: Implements the main GUI for the game, including the main menu and in-game interface.

Key Features:

Button listeners for navigating menus and starting the game.

Volume sliders for adjusting audio settings.

Smooth animations using double-buffered panels.

Coding Standards

- Language: C#
- Framework: .NET Framework
- GUI Framework: Windows Forms
- Design Principles:

    Modular design using abstract classes for extensibility.

    Encapsulation of gameplay mechanics within distinct classes.

    Event-driven programming for user interactions.

- Comments:

    XML documentation for public methods and properties.

    Inline comments for complex logic.

Future Enhancements

- New Enemy Types: Introduce additional alien behaviors and animations.
- Power-Ups: Add collectable items that enhance the spaceship's abilities.
- Multiplayer Mode: Implement cooperative or competitive gameplay.
- Cross-Platform Support: Transition to .NET Core for broader compatibility.
- Enhanced UI: Improve menu and gameplay aesthetics with modern design libraries. Also change it so that there would be no need for Scale & Layout to be set to 125% to work.

Contact

For technical support or contribution inquiries, contact: [sebastian.soptelea@proton.me]