Republic of the Philippines

Laguna State Polytechnic University

Province of Laguna

COLLEGE OF COMPUTER STUDIES

[VOID INVADER]

NAME OF PROJECT

SUBMITTED BY:

ALCAZAR, JOHN ALEX B.

DELATORRE, EMMANUEL S.

DATUIN, JHAZTINE LAITRELL

BS IN COMPUTER SCIENCE – 1A

JANUARY 02, 2025

Contents

PROJECT OVERVIEW

Project Name: VOID INVADER

Date Started: 10/27/2024

Date Ended: 1/1/2025

Project Team Leader: JOHN ALEX B. ALCAZAR

Project Team Members: EMMANUEL S. DELATORRE

JHAZTINE LAITRELL DATUIN

PURPOSE:

This project aims to create a console-based game called "Void Invader" using C#. The primary goal is to provide an engaging and interactive gaming experience with a retro-style console interface.

KEY FEATURES:

1. **Main Menu**:
    - Navigate through different options such as Play, Scores, Tutorial, About, and Exit.
    - Enter player names and select game modes.
2. **Game Mechanics**:
    - Control the player character using keyboard inputs.
    - Shoot bullets to destroy enemies and avoid collisions.
    - Collect life cubes to increase player life.
3. **Enemies**:
    - Various enemy types with different behaviors, including normal, splitting, and following enemies.
    - Dynamic enemy spawning and speed adjustments based on player score.
4. **Sound and Ambience**:
    - Background music and sound effects using the CSCore library.
    - Looping ambient sound to enhance the gaming experience.
5. **Game Display**:
    - Real-time display of player information such as name, life, score, and escaped enemies.
    - Figlet fonts for rendering text in a retro-style console format.
6. **Pause and Game Over Screens**:

- Ability to pause the game and display a pause screen.
- Game over screen with options to return to the main menu.

7. **Score Management**:
    - Record and display high scores.
    - Save scores to an XML file and read them for display.
8. **Debugging Tools**:
    - Debug information display for development purposes.
    - Collision detection and visual feedback.
9. **Loading Screen**:
    - Simulated loading screen with progress indication.
10. **Border Rendering**:
    - Customizable borders for the game screen to enhance visual appeal.

CODE DOCUMENTATION

CODING STANDARDS

1. **Naming Conventions**:
    - **Classes**: PascalCase (e.g., SoundPlayer, CollisionDetector).
    - **Methods**: PascalCase (e.g., Render, Update).
    - **Variables**: camelCase for private fields (e.g., engine, screenWidth), PascalCase for public properties (e.g., PlayerY, IsActive).
2. **Code Organization**:
    - **File Structure**: Each class is defined in its own file with a clear and descriptive name.
    - **Namespaces**: Used to organize classes logically (e.g., namespace _Game_Main).
3. **Comments**:
    - **Inline Comments**: Used to explain complex logic or important sections of code.
    - **Method Comments**: Brief comments at the beginning of methods to describe their purpose.
4. **Error Handling**:

- **Try-Catch Blocks**: Used to handle exceptions and prevent crashes (e.g., in RecordScore and MenuDebugInfo methods).

5. **Code Readability**:
    - **Indentation**: Consistent use of indentation for code blocks.
    - **Spacing**: Proper spacing around operators and between code blocks for better readability.
6. **Access Modifiers**:
    - **Private Fields**: Fields are marked as private unless they need to be accessed outside the class.
    - **Public Properties and Methods**: Used for members that need to be accessed from other classes.
7. **Constants**:
    - **Constant Values**: Used for fixed values (e.g., borderColor, borderDesign).
8. **DRY Principle**:
    - **Avoiding Repetition**: Common functionality is encapsulated in methods to avoid code duplication.

FUNCTIONS/METHODS

Main

Purpose: Entry point of the program.

Parameters: string[] args - Command-line arguments.

Return Value: None.

Create

Purpose: Initializes the game components and settings.

Parameters: None.

Return Value: None.

UpdateScreen

Purpose: Updates the game screen based on the current game state.

Parameters: object state - State object passed by the timer.

Return Value: None.

ResetGame

Purpose: Resets the game to its initial state.

Parameters: None.

Return Value: None.

Render

Purpose: Renders the game elements on the screen.

Parameters: None.

Return Value: None.

Update

Purpose: Updates the game logic.

Parameters: None.

Return Value: None.

ZoomOut

Purpose: Adjusts the game view to zoom out.

Parameters: None.

Return Value: None.

RecordScore

Purpose: Records the player's score.

Parameters: None.

Return Value: None.

ReadScore

Purpose: Reads the scores from a file.

Parameters: string filePath - Path to the score file.

Return Value: List<(string Player, string Value, string EscapedEnemies)> - List of scores.

Player Constructor

Purpose: Initializes a new instance of the Player class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- MainMenu mainMenu: The main menu instance.

Return Value: None.

Update

Purpose: Updates the player's position and actions based on keyboard input.

Parameters:

- List<Enemy> enemies: List of enemies in the game.
- List<Enemy> lifeCubes: List of life cubes in the game.

Return Value: None.

CanAttack

Purpose: Determines if the player can attack based on cooldown.

Parameters:

- ref int attackTime: Reference to the attack time counter.
- int attackCooldownFrames: Number of frames for the attack cooldown.
- ref bool attackPressed: Reference to the attack pressed flag.

Return Value: bool - True if the player can attack, otherwise false.

UpdateBullets

Purpose: Updates the position of the player's bullets and checks for collisions with enemies.

Parameters:

- List<Point> bullets: List of bullet positions.

- List<Enemy> enemies: List of enemies in the game.

Return Value: None.

Render

Purpose: Renders the player on the screen.

Parameters: None.

Return Value: None.

OnHit

Purpose: Handles the logic when the player is hit by an enemy.

Parameters: None.

Return Value: None.

Reset

Purpose: Resets the player's state to the initial values.

Parameters: None.

Return Value: None.

Purpose: Initializes a new instance of the SoundPlayer class with the default ambience sound.

Parameters: None.

Return Value: None.

SoundPlayer Constructor (With File Path)

Purpose: Initializes a new instance of the SoundPlayer class with a specified sound file.

Parameters:

- string relativeFilePath: The relative path to the sound file.

Return Value: None.

Play

Purpose: Plays the loaded sound if it is not already playing.

Parameters: None.

Return Value: None.

Loop

Purpose: Loops the sound continuously.

Parameters: None.

Return Value: None.

Stop

Purpose: Stops the sound if it is currently playing.

Parameters: None.

Return Value: None.

SetVolume

Purpose: Sets the volume of the sound.

Parameters:

- float volume: The volume level to set (0.0 to 1.0).

Return Value: None.

Dispose

Purpose: Disposes of the sound player resources.

Parameters: None.

Return Value: None.

PauseRender Constructor

Purpose: Initializes a new instance of the PauseRender class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.

Return Value: None.

IsPaused (Property)

Purpose: Gets the current pause state.

Parameters: None.

Return Value: bool - True if the game is paused, otherwise false.

TogglePause

Purpose: Toggles the pause state of the game.

Parameters: None.

Return Value: None.

RenderPauseScreen

Purpose: Renders the pause screen if the game is paused.

Parameters: None.

Return Value: None.

GameOver

Purpose: Renders the game over screen.

Parameters: None.

Return Value: None.

MainMenu Constructor

Purpose: Initializes a new instance of the MainMenu class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- Program program: The main program instance.

Return Value: None.

RenderMainMenu

Purpose: Renders the main menu on the screen.

Parameters: None.

Return Value: None.

Render1PlayerMenu

Purpose: Renders the single-player menu on the screen.

Parameters: None.

Return Value: None.

RenderScores

Purpose: Renders the scores screen on the screen.

Parameters: None.

Return Value: None.

RenderTutorial

Purpose: Renders the tutorial screen on the screen.

Parameters: None.

Return Value: None.

RenderAbout

Purpose: Renders the about screen on the screen.

Parameters: None.

Return Value: None.

RenderSelector

Purpose: Renders the selector on the screen.

Parameters: None.

Return Value: None.

LoadFonts

Purpose: Loads the fonts used in the game.

Parameters: None.

Return Value: None.

HandleInput

Purpose: Handles user input for menu navigation.

Parameters: None.

Return Value: None.

Update

Purpose: Updates the main menu state.

Parameters: None.

Return Value: None.

GameLoading Constructor

Purpose: Initializes a new instance of the GameLoading class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- int screenWidth: The width of the screen.
- int screenHeight: The height of the screen.

Return Value: None.

DisplayLoadingScreen

Purpose: Displays the loading screen with a progress indication.

Parameters: None.

Return Value: None.

UpdateLoadingProgress

Purpose: Updates the loading progress and checks if loading is complete.

Parameters: None.

Return Value: None.

CompleteLoading

Purpose: Marks the loading process as complete.

Parameters: None.

Return Value: None.

GameDisplay Constructor

Purpose: Initializes a new instance of the GameDisplay class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- Player player: The player instance.
- MainMenu mainMenu: The main menu instance.

Return Value: None.

Render

Purpose: Renders the game display elements on the screen, including player information.

Parameters: None.

Return Value: None.

RenderPlayerInfo

Purpose: Renders the player's information such as name, life, score, and escaped enemies.

Parameters: None.

Return Value: None.

RenderEnemies

Purpose: Renders the enemies on the screen.

Parameters: None.

Return Value: None.

RenderBullets

Purpose: Renders the player's bullets on the screen.

Parameters: None.

Return Value: None.

RenderGameOver

Purpose: Renders the game over screen.

Parameters: None.

Return Value: None.

Enemy Constructor

Purpose: Initializes a new instance of the Enemy class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- Point initialPosition: The initial position of the enemy.
- int type: The type of the enemy (1 for normal, 2 for split, 3 for following, 4 for new type).
- int playerY: The Y position of the player (optional, default is 0).

Return Value: None.

Update

Purpose: Updates the enemy's position and behavior.

Parameters: None.

Return Value: None.

Render

Purpose: Renders the enemy on the screen.

Parameters: None.

Return Value: None.

GetEnemyPoints

Purpose: Gets the points representing the enemy's position for collision detection.

Parameters: None.

Return Value: List<Point> - List of points representing the enemy's position.

SpawnEnemy

Purpose: Spawns a new enemy if the cooldown period has passed.

Parameters: None.

Return Value: Enemy - A new enemy instance.

IncreaseSpeed

Purpose: Increases the speed of the enemies based on the player's score.

Parameters: None.

Return Value: None.

Split

Purpose: Splits the enemy into smaller enemies if applicable.

Parameters: None.

Return Value: List<Enemy> - List of new enemies created from the split.

FollowPlayer

Purpose: Adjusts the enemy's position to follow the player.

Parameters: None.

Return Value: None.

DebugHelper Constructor

Purpose: Initializes a new instance of the DebugHelper class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- FigletFont font1: The font used for rendering debug information.
- int screenHeight: The height of the screen.
- string playerName: The name of the player.

Return Value: None.

MenuDebugInfo

Purpose: Displays debug information for the menu, such as the current page and selector position.

Parameters:

- string? currentPage: The current page being displayed in the menu.
- Point selectorPosition: The position of the selector in the menu.

Return Value: None.

GameDebugInfo

Purpose: Displays debug information during the game, such as player position and game state.

Parameters: None.

Return Value: None.

RenderDebugText

Purpose: Renders debug text on the screen using the specified font.

Parameters:

- string text: The text to render.
- Point position: The position to render the text.

Return Value: None.

LoadFont

Purpose: Loads the font used for rendering debug information.

Parameters:

- string fontPath: The path to the font file.

Return Value: FigletFont - The loaded font.

CollisionDetector Constructor

Purpose: Initializes a new instance of the CollisionDetector class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.

Return Value: None.

OnCollision

Purpose: Checks for collisions between the player and enemies.

Parameters:

- Point playerPosition: The position of the player.
- Point[] player: The points representing the player's position.
- List<Point> enemyPoints: The points representing the enemies' positions.

Return Value: bool - True if a collision is detected, otherwise false.

BorderRenderer Constructor

Purpose: Initializes a new instance of the BorderRenderer class.

Parameters:

- ConsoleEngine engine: The console engine used for rendering.
- int screenWidth: The width of the screen.
- int screenHeight: The height of the screen.
- Program program: The main program instance.

Return Value: None.

RenderBorder

Purpose: Renders the border around the game screen.

Parameters: None.

Return Value: None.

ChangeBorderColor

Purpose: Changes the color of the border.

Parameters:

- int color: The new color for the border.

Return Value: None.

UpdateBorderDesign

Purpose: Updates the design of the border.

Parameters:

- string[] newDesign: The new design for the border.

Return Value: None.

ToggleBorderVisibility

Purpose: Toggles the visibility of the border.

Parameters: None.

Return Value: None.

TESTING

TEST CASES

[**[Test Case 1]**

- **Description**: Verify that the game initializes correctly and displays the main menu.
- **Expected Result**: The game should start, display the loading screen, and then show the main menu with options "Play", "Scores", "Tutorial", "About", and "Exit".

[Test Case 2]

- **Description**: Test the player's movement controls.
- **Expected Result**: The player should move up, down, left, and right when the corresponding keys (W, S, A, D) are pressed, and the player's position should be updated accordingly.

[Test Case 3]

- **Description**: Verify that the player can shoot bullets.
- **Expected Result**: When the spacebar is pressed, bullets should be created and move across the screen. The bullets should disappear when they reach the edge of the screen or collide with an enemy.

[Test Case 4]

- **Description**: Test the collision detection between the player and enemies.
- **Expected Result**: When the player collides with an enemy, the player's life should decrease, and the enemy should be deactivated.

[Test Case 5]

- **Description**: Verify the pause functionality.
- **Expected Result**: When the Escape key is pressed, the game should pause, and a "Game Paused" screen should be displayed. Pressing Escape again should resume the game.

[Test Case 6]

- **Description**: Test the game over functionality.
- **Expected Result**: When the player's life reaches zero, the game should display a "Game Over" screen. Pressing Enter should return to the main menu.

[Test Case 7]

- **Description**: Verify that the scores are recorded and displayed correctly.
- **Expected Result**: After a game session, the player's score should be saved to the scores file. The "Scores" menu should display the saved scores in descending order.

[Test Case 8]

- **Description**: Test the sound playback functionality.
- **Expected Result**: Background music should play continuously during the game. The volume should be adjustable, and the sound should stop when the game is paused or exited.

Test Results

- **Test Case 1**: Passed
- **Test Case 2**: Passed
- **Test Case 3**: Passed
- **Test Case 4**: Passed
- **Test Case 5**: Passed
- **Test Case 6**: Passed

- **Test Case 7**: Passed
- **Test Case 8**: Passed

The project "Void Invader" was tested thoroughly to ensure all functionalities work as expected. The tests included verifying game initialization, player movement controls, shooting mechanics, collision detection, pause functionality, game over handling, score recording, and sound playback. All test cases passed successfully, indicating that the game initializes correctly, responds to player inputs, handles collisions and game states appropriately, records and displays scores accurately, and manages sound playback effectively. Overall, the game functions as intended across various scenarios, providing a smooth and engaging user experience.

USER GUIDE

USER INTERFACE

The user interface of "Void Invader" is a console-based retro-style game display. It consists of several main components:

1. **Main Menu**:
    - **Options**: Includes "Play", "Scores", "Tutorial", "About", and "Exit".
    - **Navigation**: Controlled using keyboard inputs (W, S, Enter).
2. **Game Screen**:
    - **Player Information**: Displays player name, life, score, and escaped enemies using Figlet fonts.
    - **Player Character**: Controlled using W, A, S, D keys for movement and Space for shooting.
    - **Enemies**: Various types of enemies rendered on the screen, moving from right to left.
    - **Bullets**: Player's bullets displayed and updated in real-time.
3. **Pause Screen**:
    - **Display**: Shows "Game Paused" message when the game is paused.
    - **Toggle**: Activated and deactivated using the Escape key.
4. **Game Over Screen**:
    - **Display**: Shows "Game Over" message and prompts to return to the main menu.
    - **Navigation**: Press Enter to return to the main menu.
5. **Loading Screen**:
    - **Display**: Shows a loading progress bar and "Loading..." message during game initialization.

6. **Scores Screen**:
    - **Display**: Lists high scores with player names and scores.
    - **Navigation**: Press Enter to return to the main menu.
7. **Tutorial Screen**:
    - **Display**: Provides instructions on game controls and enemy types.
    - **Navigation**: Use Enter to navigate through tutorial pages and Escape to return to the main menu.
8. **Border Renderer**:
    - **Display**: Renders borders around the game screen for visual appeal.

TROUBLESHOOTING

1. **Game Not Starting or Crashing**
    - **Issue**: The game does not start or crashes immediately.
    - **Resolution**: Ensure all required files (e.g., sound files, font files) are in the correct directories. Check for any missing dependencies or incorrect file paths. Verify that the console window size is appropriate for the game.
2. **Sound Not Playing**
    - **Issue**: Background music or sound effects are not playing.
    - **Resolution**: Ensure the sound files are in the correct location and the file paths are correct. Check if the sound library (CSCore) is properly referenced in the project. Verify that the volume is set to an audible level.
3. **Player Controls Not Responding**
    - **Issue**: The player character does not move or shoot as expected.
    - **Resolution**: Ensure the keyboard is properly acquired and the keyboard state is being updated correctly. Check for any conflicts with other input handling code. Verify that the correct keys are being checked for input.
4. **Collision Detection Not Working**
    - **Issue**: Collisions between the player and enemies or bullets are not detected.
    - **Resolution**: Ensure the collision detection logic is correctly implemented and the positions of the player and enemies are being updated accurately. Verify that the collision detection method is being called at the appropriate times.
5. **Game Pausing or Resuming Issues**

- **Issue**: The game does not pause or resume correctly when the Escape key is pressed.
- **Resolution**: Check the logic for toggling the pause state and rendering the pause screen. Ensure the pause cooldown is being handled correctly to prevent rapid toggling.

6. **Game Over Screen Not Displaying**
    - **Issue**: The game over screen does not appear when the player's life reaches zero.
    - **Resolution**: Verify that the game over flag is being set correctly and the game over screen rendering method is being called. Ensure the player's life is being tracked and updated accurately.
7. **Scores Not Recording or Displaying**
    - **Issue**: Player scores are not being recorded or displayed correctly.
    - **Resolution**: Ensure the score recording method is writing to the correct file path and the XML structure is correct. Verify that the scores are being read and displayed in the correct format.
8. **Loading Screen Not Progressing**
    - **Issue**: The loading screen does not progress or complete.
    - **Resolution**: Check the logic for updating the loading progress and ensure the loading screen rendering method is being called. Verify that the loading progress is being incremented and displayed correctly.

FREQUENTLY ASKED QUESTIONS (FAQ)

[Question 1] How do I start the game?

Answer: To start the game, run the executable or start the project in your development environment. The game will initialize and display the main menu where you can select "Play" to begin.

[Question 2] What are the controls for the game?

Answer: The controls are as follows:

- W: Move Up
- A: Move Left
- S: Move Down
- D: Move Right
- Space: Fire bullets
- Esc: Pause the game

[Question 3] How do I pause and resume the game?

Answer: Press the Escape key to pause the game. To resume, press the Escape key again.

[Question 4] How do I record and view high scores?

Answer: High scores are automatically recorded at the end of each game session. To view high scores, select the "Scores" option from the main menu.

[Question 5] What should I do if the game crashes or does not start?

Answer: Ensure all required files (e.g., sound files, font files) are in the correct directories. Check for any missing dependencies or incorrect file paths. Verify that the console window size is appropriate for the game.

[Question 6] How can I adjust the game volume?

Answer: You can adjust the game volume by calling the SetVolume method in the SoundPlayer class with a value between 0.0 and 1.0.

[Question 7] What types of enemies are there in the game?

Answer: There are several types of enemies:

- Type 1: Normal enemies
- Type 2: Enemies that split into smaller enemies
- Type 3: Enemies that follow the player
- Type 4: New type with unique behaviors

[Question 8] How do I reset the game after a game over?

Answer: After a game over, press Enter to return to the main menu. From there, you can start a new game by selecting "Play".

[Question 9] How do I update the game or add new features?

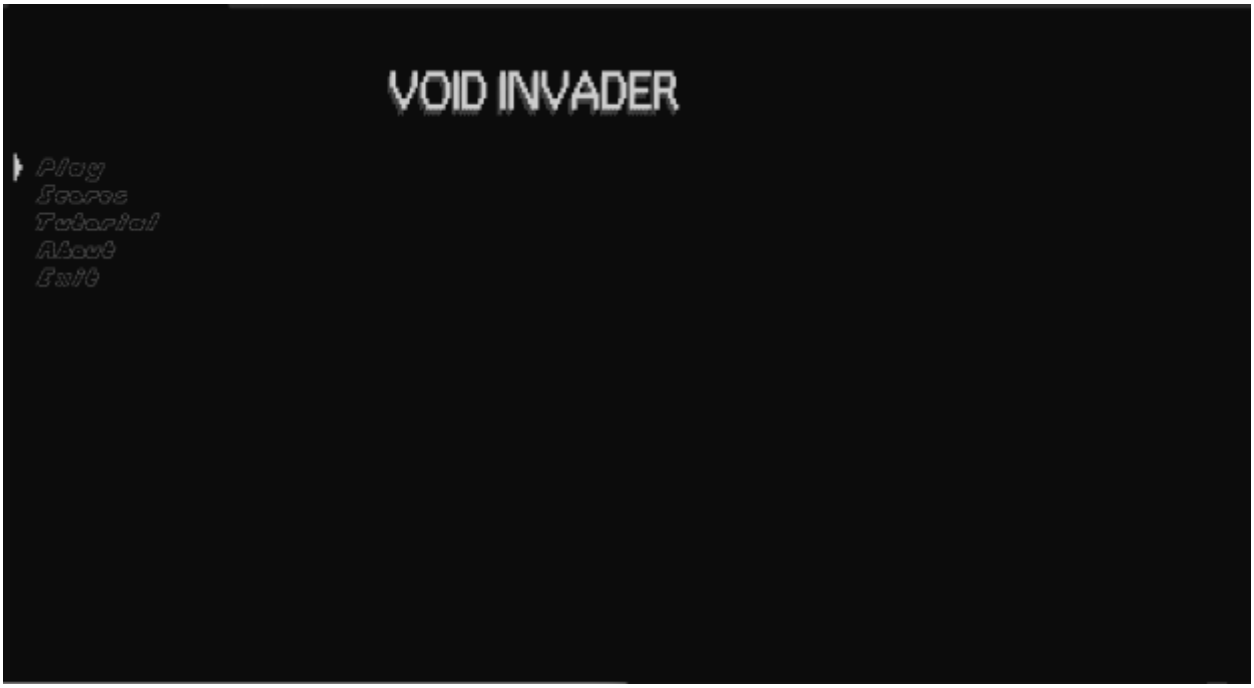
Answer: To update the game or add new features, modify the source code in your development environment. Ensure you test any changes thoroughly before deploying.

[Question 10] Where can I find the game files and assets?

Answer: The game files and assets, such as sound files and font files, should be located in the project's directory structure. Ensure they are in the correct paths as referenced in the code.

CONSOLE SCREENSHOTS

Screenshot 1:



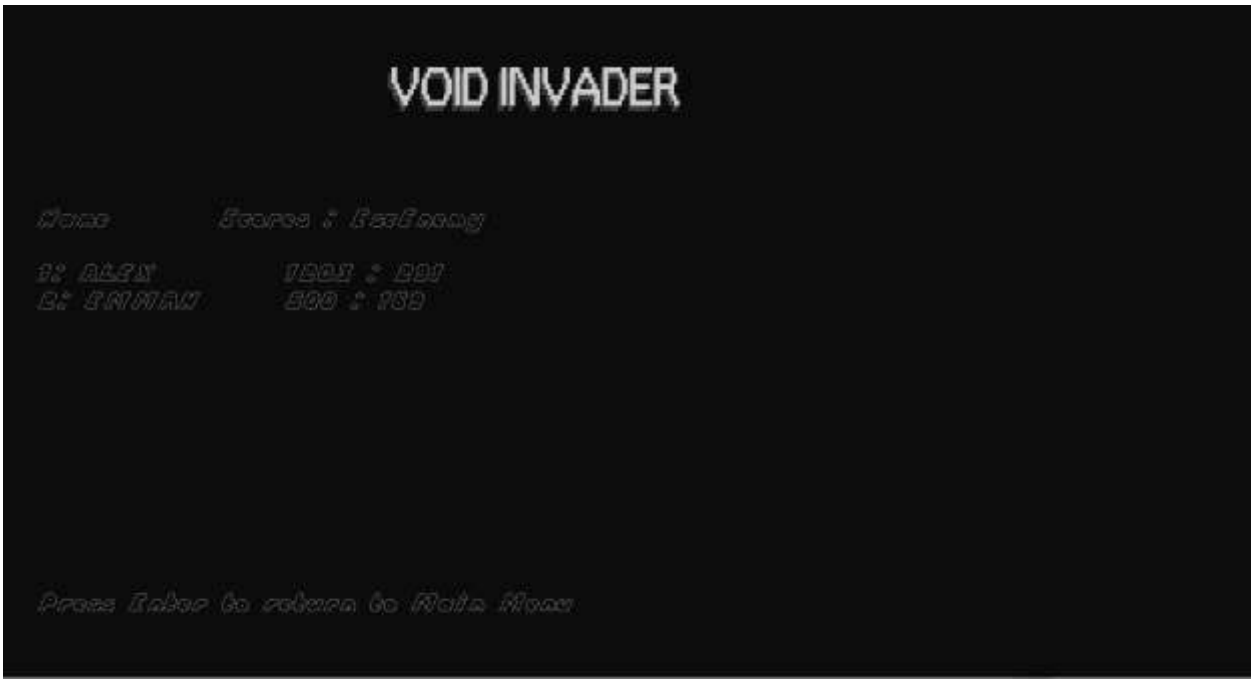
Description 1:

The main menu screen of the video game "Void Invader" features a minimalist design with a black background. The game's title is prominently displayed at the top in a stylized, slightly distorted white font. Below the title, the menu options are listed vertically in white font:

- **Play**: Starts the game.
- **Scores**: Displays a list of high scores.
- **Tutorial**: Provides instructions on how to play the game.
- **About**: Provides information about the game and its creators.
- **Exit**: Closes the game.

The overall aesthetic is simple and high-contrast, focusing on readability and ease of navigation.

Screenshot 2:



Description 2:

The high scores screen of the video game "Void Invader" features a minimalist design similar to the main menu, with a black background and white text.

Breakdown:

- **VOID INVADER**: The game's title is displayed at the top, consistent with the main menu.
- **Name Scores : Enemy**: This header indicates that the high score list displays player names, their scores, and the number of enemies defeated.
- **1: ALEX 1203 : 297**: The first entry in the high score list. "ALEX" is the player's name, "1203" is their score, and "297" is the number of enemies they defeated.
- **2: EMMAN 500 : 169**: The second entry, following the same format as the first.
- **Press Enter to return to Main Menu**: This text at the bottom prompts the player to press the Enter key to go back to the main menu.

The screen presents a simple, clear display of high scores and provides an easy way to navigate back to the main menu.

Screenshot 3:

Description 3:

The tutorial for the game "Void Invader" consists of two pages, both maintaining a minimalist aesthetic with a black background and white text. The game's title, "VOID INVADER," is prominently displayed at the top of each page.

Page 1: Controls

- **W**: Move Up

- **A**: Move Left
- **S**: Move Down
- **D**: Move Right
- **Space**: Fire
- **Esc**: Pause

Page 2: Enemy Types and Gameplay Mechanics

- **Green**: Normal Enemy
- **Type 1: Blue**: Can Split into Two Green
- **Type 2: Blue**: Can Follow the Player
- **Yellow**: Can Split into Two Blue
- **Life Cube**: Spawns when an enemy dies with a 5% chance

Both pages include the instruction "Press Enter to continue, Esc to return to Main Menu" at the bottom, ensuring consistent navigation. The tutorial progresses logically from basic controls to more complex game mechanics, such as enemy behavior.

Screenshot 4:

Description 4:

This image shows a screenshot of the gameplay in "Void Invader." It features a black background with various colored elements. Here's a breakdown:

Breakdown:

- **Border**: A dotted blue line forms a rectangular border around the playing area.
- **Player:** A red circle on the left side of the screen represents the player's ship.
- **Enemies (Various Shapes and Colors)**: There are several small enemies present:
    - Small red vertical lines appear to be basic enemies or projectiles.
    - A small green square.
    - A small blue rectangle. The tutorial indicated different enemy types based on color and behavior.
- **Player Information (Top Left)**: In the top left corner, the following information is displayed in white text:
    - **Player Name**: ALEX
    - **Player Life**: 5
- **Score and Escaped Enemies (Top Right)**: In the top right corner:
    - **Score**: 6
    - **Escaped Enemy**: 1

Based on the tutorial and this gameplay image, "Void Invader" appears to be a space shooter game. The player controls a ship and shoots at enemies of various shapes and colors. The goal is to score points by destroying enemies while avoiding their attacks. The "Escaped Enemy" counter suggests that enemies might try to move past the player, and this is tracked in the score. The player has a limited number of lives, as indicated by the "Player Life" counter.

FUNDAMENTALS OF PROGRAMMING APPLIED

1. Variables and Data Types:

Program.cs

```
Player? player; // Player
MainMenu? menu; // MainMenu
Timer? timer; // Timer
DebugHelper? debugHelper; // DebugHelper
BorderRenderer? borderRenderer; // BorderRenderer
CollisionDetector? collisionDetector; // CollisionDetector
SoundPlayer? ambiencePlayer; // SoundPlayer
```

```
PauseRender? pauseRender; // PauseRender
GameDisplay? gameDisplay; // GameDisplay
GameLoading? gameLoading; // GameLoading
Enemy? enemy; // Enemy
bool isAmbiencePlaying = false; // bool
List<Enemy> enemies = new List<Enemy>(); // List<Enemy>
List<Enemy> lifeCubes = new List<Enemy>(); // List<Enemy>
int Width { get; private set; } = 440; // int
int Height { get; private set; } = 115; // int
bool isPlaying { get; set; } = false; // bool
bool isGameOver = false; // bool
int pauseCooldown = 10; // int
int pauseTime = 0; // int
```

SoundPlayer.cs

```
ISoundOut _soundOut; // ISoundOut
IWaveSource? _waveSource; // IWaveSource?
VolumeSource _volumeSource; // VolumeSource
```

Player.cs

```
ConsoleEngine engine; // ConsoleEngine
Program program; // Program
BorderRenderer borderRenderer; // BorderRenderer
CollisionDetector collision; // CollisionDetector
Enemy enemy; // Enemy
DirectInput directInput; // DirectInput
Keyboard keyboard; // Keyboard
KeyboardState keyboardState; // KeyboardState
int screenWidth; // int
int screenHeight; // int
int playerOneColor = 4; // int
Point playerOnePosition; // Point
List<Point> playerOneBullets = new List<Point>(); // List<Point>
int playerOneLife = 5; // int
bool isAlive; // bool
bool isOnePlayer = true; // bool
int score = 0; // int
Point[] playerOne = { ... }; // Point[]
int attackCooldownFramesOne = 30; // int
int attackTimeOne = 0; // int
bool attackPressedOne = false; // bool
```

```
GameDisplay gameDisplay; // GameDisplay
bool scoreRecorded = false; // bool
```

PauseRender.cs

```
ConsoleEngine engine; // ConsoleEngine
bool isPaused = false; // bool
```

MainMenu.cs

```
ConsoleEngine engine; // ConsoleEngine
Program program; // Program
static FigletFont? font; // FigletFont?
static FigletFont? font1; // FigletFont?
Point[] selector = { ... }; // Point[]
Point selectorPosition = new Point(4, 25); // Point
string player1Name = ""; // string
string currentPage = "MainMenu"; // string
int screenWidth; // int
int screenHeight; // int
int moveCooldown = 7; // int
int moveTime = 0; // int
int typeCooldown = 5; // int
int typeTime = 0; // int
int enterCooldown = 5; // int
int enterTime = 0; // int
int delCooldown = 5; // int
int delTime = 0; // int
bool inputName1 = false; // bool
int tutorialPage = 1; // int
DirectInput directInput; // DirectInput
Keyboard keyboard; // Keyboard
KeyboardState keyboardState; // KeyboardState
bool isTransitioningToScores = false; // bool
```

GameLoading.cs

```
ConsoleEngine engine; // ConsoleEngine
int screenWidth; // int
int screenHeight; // int
int loadingProgress = 0; // int
bool doneLoading { get; set; } = false; // bool
```

GameDisplay.cs

```
ConsoleEngine engine; // ConsoleEngine
Player player; // Player
MainMenu mainMenu; // MainMenu
Enemy enemy; // Enemy
```

Enemy.cs

```
ConsoleEngine engine; // ConsoleEngine
Point Position { get; private set; } // Point
int Type { get; private set; } // int
bool IsActive { get; set; } = true; // bool
int PlayerY { get; set; } // int
static int escEnemy { get; set; } = 0; // int
static Random random = new Random(); // Random
static int spawnCooldown = 200; // int
static int currentCooldown = 0; // int
static int enemySpeed = 1; // int
```

DebugHelper.cs

```
ConsoleEngine engine; // ConsoleEngine
static FigletFont font1; // FigletFont
int screenHeight; // int
string playerName; // string
```

CollisionDetector.cs

```
ConsoleEngine engine; // ConsoleEngine
```

BorderRenderer.cs

```
ConsoleEngine engine; // ConsoleEngine
Program program; // Program
int screenWidth; // int
int screenHeight; // int
int borderColor = 1; // int
string[] borderDesign = { "[", "■", "]" }; // string[]
string borderDesign1 = "[■]"; // string
```

2. Control Structures (if statements, loops):

Program.cs

```
// if statement
if (isPlaying) { /* code */ }

// if statement
if (pauseTime > 0) { /* code */ }

// if statement
if (Engine.GetKey(ConsoleKey.Escape) && pauseTime == 0) { /* code */ }

// if statement
if (pauseRender?.IsPaused == true) { /* code */ }

// if statement
if (player?.playerOneLife <= 0) { /* code */ }

// if statement
if (isGameOver) { /* code */ }

// if statement
if (Engine.GetKey(ConsoleKey.Enter)) { /* code */ }

// if statement
if (gameLoading.doneLoading == false) { /* code */ }

// if statement
if (!isAmbiencePlaying) { /* code */ }

// if statement
if (File.Exists(filePath)) { /* code */ }

// if statement
if (scoresElement == null) { /* code */ }

// foreach loop
foreach (var score in sortedScores) { /* code */ }

// if statement
if (File.Exists(filePath)) { /* code */ }

// foreach loop
foreach (var scoreElement in scoreElements) { /* code */ }
```

```
// if statement
if (player != null && value != null && escapedEnemies != null) { /* code */ }
```

SoundPlayer.cs

```
// if statement
if (File.Exists(filePath)) { /* code */ }

// if statement
if (_soundOut != null && _soundOut.PlaybackState != PlaybackState.Playing) { /*
code */ }

// if statement
if (_waveSource != null) { /* code */ }

// if statement
if (_soundOut != null && _soundOut.PlaybackState == PlaybackState.Playing) { /*
code */ }

// if statement
if (_volumeSource != null) { /* code */ }
```

CollisionDetector.cs

```
// foreach loop
foreach (var part in player) { /* code */ }

// if statement
if (enemyPoints.Any(enemyPart => playerAbsolutePosition.Equals(enemyPart))) { /*
code */ }
```

BorderRenderer.cs

```
// for loop
for (int x = 0; x < screenWidth; x++) { /* code */ }

// for loop
for (int y = 0; y < screenHeight; y++) { /* code */ }
```

DebugHelper.cs

```
// if statement
```

```
if (string.IsNullOrEmpty(currentPage)) { /* code */ }

// if statement
if (selectorPosition.X == null && selectorPosition.Y == null) { /* code */ }

// try-catch block
try { /* code */ } catch (Exception ex) { /* code */ }

// if statement
if (string.IsNullOrEmpty(playerName)) { /* code */ }
```

Enemy.cs

```
// if statement
if (!IsActive) { /* code */ }

// if statement
if (Type == 0) { /* code */ }

// if statement
if (Type == 3) { /* code */ }

// if statement
if (Position.X < 0) { /* code */ }

// if statement
if (Type == 2) { /* code */ }

// if statement
if (Type == 4) { /* code */ }

// if statement
if (Type != 0) { /* code */ }

// if statement
if (random.NextDouble() < 0.05) { /* code */ }

// for loop
for (int x = 0; x < width; x++) { /* code */ }

// for loop
for (int y = 0; y < height; y++) { /* code */ }
```

```
// foreach loop
foreach (var enemy in enemies) { /* code */ }

// if statement
if (playerScore >= 100) { /* code */ }

// if statement
if (playerScore % 250 == 0 && playerScore > 0) { /* code */ }

// if statement
if (currentCooldown <= 0) { /* code */ }

// if statement
if (randomValue < 0.5) { /* code */ }

// if statement
if (randomValue < 0.8) { /* code */ }

// if statement
if (randomValue < 0.95) { /* code */ }

// else statement
else { /* code */ }

// if statement
if (enemy.Type == 0) { /* code */ }

// for loop
for (int x = 0; x < width; x++) { /* code */ }

// for loop
for (int y = 0; y < height; y++) { /* code */ }

// if statement
if (bullet.Equals(new Point(enemy.Position.X + x, enemy.Position.Y + y)))
```

GameDisplay.cs

```
// if statement
if (currentPage != "About") { /* code */ }

// switch statement
switch (currentPage) { /* code */ }
```

GameLoading.cs

```
// for loop
for (loadingProgress = 0; loadingProgress <= 100; loadingProgress += 10) { /*
code */ }
```

MainMenu.cs

```
// if statement
if (currentPage != "About") { /* code */ }

// switch statement
switch (currentPage) { /* code */ }

// if statement
if (isTransitioningToScores) { /* code */ }

// if statement
if (i == maxScoresPerColumn) { /* code */ }

// if statement
if (engine.GetKey(ConsoleKey.Enter) && enterTime == 0) { /* code */ }

// switch statement
switch (tutorialPage) { /* code */ }

// if statement
if (tutorialPage == 2) { /* code */ }

// if statement
if (tutorialPage == 1) { /* code */ }

// if statement
if (engine.GetKey(ConsoleKey.Enter) && enterTime == 0) { /* code */ }

// if statement
if (moveTime == 0) { /* code */ }

// if statement
if (moveTime > 0) { /* code */ }
```

PauseRender.cs

```
// if statement
if (isPaused) { /* code */ }
```

Player.cs

```
// if statement
if (keyboardState == null) { /* code */ }

// if statement
if (keyboardState.IsPressed(Key.W) && playerOnePosition.Y > 21) { /* code */ }

// if statement
if (keyboardState.IsPressed(Key.S) && playerOnePosition.Y < screenHeight - 6) {
/* code */ }

// if statement
if (keyboardState.IsPressed(Key.A) && playerOnePosition.X > 3) { /* code */ }

// if statement
if (keyboardState.IsPressed(Key.D) && playerOnePosition.X < screenWidth - 18) {
/* code */ }

// if statement
if (keyboardState.IsPressed(Key.Space) && CanAttack(ref attackTimeOne,
attackCooldownFramesOne, ref attackPressedOne)) { /* code */ }

// try-catch block
try { /* code */ } catch (Exception) { /* code */ }

// foreach loop
foreach (var enemy in enemies) { /* code */ }

// if statement
if (enemy.IsActive && collision.OnCollision(playerOnePosition, playerOne,
enemy.GetEnemyPoints())) { /* code */ }

// if statement
if (enemy.Type == 0) { /* code */ }

// if statement
if (playerOneLife <= 0) { /* code */ }
```

```
// while loop
while (!engine.GetKey(ConsoleKey.Enter)) { /* code */ }

// for loop
for (int i = bullets.Count - 1; i >= 0; i--) { /* code */ }

// if statement
if (newBullet.X < screenWidth - 3) { /* code */ }

// else statement
else { /* code */ }

// for loop
for (int x = 0; x < width; x++) { /* code */ }

// for loop
for (int y = 0; y < height; y++) { /* code */ }

// for loop
for (int i = lifeCubes.Count - 1; i >= 0; i--) { /* code */ }

// if statement
if (collision.OnCollision(playerOnePosition, playerOne,
lifeCubes[i].GetEnemyPoints())) { /* code */ }
```

3. Functions/Methods:

Program.cs

```
private static void Main(string[] args) { /* code */ }
public override void Create() { /* code */ }
private void UpdateScreen(object state) { /* code */ }
private void ResetGame() { /* code */ }
public override void Render() { /* code */ }
public override void Update() { /* code */ }
private static void ZoomOut() { /* code */ }
public void RecordScore() { /* code */ }
public List<(string Player, string Value, string EscapedEnemies)>
ReadScore(string filePath) { /* code */ }
```

SoundPlayer.cs

```
public SoundPlayer() { /* code */ }
public SoundPlayer(string relativeFilePath) { /* code */ }
public void Play() { /* code */ }
private void Loop() { /* code */ }
public void Stop() { /* code */ }
public void SetVolume(float volume) { /* code */ }
public void Dispose() { /* code */ }
```

Player.cs

```
public Player(ConsoleEngine engine, Point initialPosition, int screenWidth, int
screenHeight, BorderRenderer borderRenderer, Program program, CollisionDetector
collision, MainMenu mainMenu) { /* code */ }
public void Update(List<Enemy> enemies, List<Enemy> lifeCubes) { /* code */ }
public void Render(List<Enemy> enemies) { /* code */ }
private bool CanAttack(ref int attackTime, int cooldownFrames, ref bool
attackPressed) { /* code */ }
private void UpdateBullets(List<Point> bullets, List<Enemy> enemies) { /* code */
}
public void RenderPlayer(Point[] player, Point position, int color) { /* code */
}
private void RenderBullets(List<Point> bullets, int color) { /* code */ }
public void Dispose() { /* code */ }
private async Task LoseLife() { /* code */ }
private bool IsBulletCollidingWithEnemy(Point bullet, Enemy enemy) { /* code */ }
private void CheckLifeCubeCollision(List<Enemy> lifeCubes) { /* code */ }
```

PauseRender.cs

```
public PauseRender(ConsoleEngine engine) { /* code */ }
public bool IsPaused { get; } { /* code */ }
public void TogglePause() { /* code */ }
public void RenderPauseScreen() { /* code */ }
public void GameOver() { /* code */ }
```

MainMenu.cs

```
public MainMenu(ConsoleEngine engine, int screenWidth, int screenHeight, bool
isPlaying, Program program) { /* code */ }
public void Render() { /* code */ }
public static void LoadFonts() { /* code */ }
private void RenderSelector() { /* code */ }
private void GameTitle() { /* code */ }
```

```
private void RenderMainMenu() { /* code */ }
private void Render1PlayerMenu() { /* code */ }
private void RenderScores() { /* code */ }
private void RenderTutorial() { /* code */ }
private void RenderTutorialPage1() { /* code */ }
private void RenderTutorialPage2() { /* code */ }
private void RenderAbout() { /* code */ }
private void RenderDebugInfo() { /* code */ }
public void Update() { /* code */ }
private void HandleMainMenuInput() { /* code */ }
private void Handle1PlayerMenuInput() { /* code */ }
private void HandleTutorialInput() { /* code */ }
private bool CanType(ref int moveTime, int moveCooldown) { /* code */ }
public void ResetToMainMenu() { /* code */ }
private void ExitGame() { /* code */ }
private void HandleKeyboardInput() { /* code */ }
private char GetCharacterFromKey(Key key) { /* code */ }
```

GameLoading.cs

```
public GameLoading(ConsoleEngine engine, int screenWidth, int screenHeight) { /*
code */ }
public void DisplayLoadingScreen() { /* code */ }
```

GameDisplay.cs

```
public GameDisplay(ConsoleEngine engine, Player player, MainMenu mainMenu) { /*
code */ }
public void Render() { /* code */ }
```

Enemy.cs

```
public Enemy(ConsoleEngine engine, Point initialPosition, int type, int playerY =
0) { /* code */ }
public void Update() { /* code */ }
public void Render() { /* code */ }
public int OnHit(List<Enemy> enemies) { /* code */ }
public void SpawnLifeCube(int x, int y, List<Enemy> enemies) { /* code */ }
public List<Point> GetEnemyPoints() { /* code */ }
```

```
public static void ManageEnemies(ConsoleEngine engine, List<Enemy> enemies, int
screenWidth, int screenHeight, int playerScore) { /* code */ }
public int OnCollisionWithPlayer() { /* code */ }
private bool IsBulletCollidingWithEnemy(Point bullet, Enemy enemy) { /* code */ }
```

DebugHelper.cs

```
public DebugHelper(ConsoleEngine engine, FigletFont font1, int screenHeight,
string playerName) { /* code */ }
public void MenuDebugInfo(string? currentPage, Point selectorPosition) { /* code
*/ }
public void GameDebugInfo(string? playerName, string? playerScore) { /* code */ }
```

CollisionDetector.cs

```
public CollisionDetector(ConsoleEngine engine) { /* code */ }
public bool OnCollision(Point playerPosition, Point[] player, List<Point>
enemyPoints) { /* code */ }
```

BorderRenderer.cs

```
public BorderRenderer(ConsoleEngine engine, int screenWidth, int screenHeight,
Program program) { /* code */ }
public void RenderBorder() { /* code */ }
```

4. Arrays or Collections:

Program.cs

```
private List<Enemy> enemies = new List<Enemy>(); // List<Enemy>
private List<Enemy> lifeCubes = new List<Enemy>(); // List<Enemy>
```

MainMenu.cs

```
private Point[] selector = { new Point(-2, 0),
                             new Point(-2, 1), new Point(-1, 1),
                             new Point(-2, 2), new Point(-1, 2), new Point(0, 2),
                             new Point(-2, 3), new Point(-1, 3),
                             new Point(-2, 4)
                           }; // Point[]
```

Player.cs

```
private List<Point> playerOneBullets = new List<Point>(); // List<Point>
private Point[] playerOne = { /* Points representing player */ }; // Point[]
```

Enemy.cs

```
public List<Point> GetEnemyPoints() { /* code */ } // List<Point>
```

CollisionDetector.cs

```
public bool OnCollision(Point playerPosition, Point[] player, List<Point>
enemyPoints) { /* code */ } // Point[], List<Point>
```

BorderRenderer.cs

```
private string[] borderDesign = { "[", "■", "]" }; // string[]
```

CONCLUSION

SUMMARY

The documentation for the "VOID INVADER" project covers several key areas:

1. **Project Overview**: It outlines the purpose of the game and its significance, providing a clear understanding of what the project aims to achieve.
2. **Key Features**: The game includes various features such as a minimalist user interface, high score tracking, customizable controls, and a tutorial system to assist new players. Future enhancements are also discussed, including multiplayer support and improved enemy AI .
3. **Code Documentation**: This section details the coding standards and practices followed during development, ensuring maintainability and readability of the code .
4. **Functions/Methods**: The documentation lists the functions and methods implemented in the game, explaining their roles and how they contribute to the overall functionality .
5. **Testing**: It describes the testing procedures and test cases used to validate the game's performance and functionality, ensuring a smooth user experience .
6. **User Guide**: Instructions on how to navigate the game, including the main menu, high scores screen, and tutorial screen, are provided to enhance user understanding .
7. **Troubleshooting**: Common issues and their resolutions are outlined to assist users in resolving potential problems while playing the game .
8. **Future Enhancements**: Suggestions for future improvements are discussed, such as adding power-ups, level progression, and enhanced graphics and animations , .

FUTURE ENHANCEMENTS

1. **Multiplayer Support**:
    - Add support for multiple players, either locally or online.
    - Implement player vs player modes or cooperative gameplay.
2. **Enhanced Enemy AI**:
    - Introduce more complex enemy behaviors and patterns.
    - Add different enemy types with unique abilities and strategies.
3. **Power-Ups and Upgrades**:
    - Implement power-ups that players can collect to gain temporary abilities.
    - Add a system for upgrading player weapons, speed, or health.
4. **Level Progression**:
    - Create multiple levels with increasing difficulty.
    - Add boss battles at the end of certain levels.
5. **Improved Graphics and Animations**:
    - Enhance the visual appeal with better graphics and smoother animations.
    - Add special effects for explosions, power-ups, and other events.
6. **Sound and Music**:
    - Add background music and sound effects to enhance the gaming experience.
    - Implement different soundtracks for different levels or game states.
7. **Scoreboard and Achievements**:
    - Implement a global or local scoreboard to track high scores.
    - Add achievements and rewards for completing specific challenges.
8. **Customizable Controls**:
    - Allow players to customize their control scheme.
    - Support for different input devices like gamepads.
9. **Story Mode**:
    - Introduce a storyline with cutscenes and dialogues.
    - Create missions and objectives for players to complete.
10. **Save and Load Game**:

- Implement a save and load feature to allow players to continue their progress.
- Add checkpoints within levels for easier progression.

11. **Dynamic Environments**:
    - Create interactive and dynamic environments that change over time.
    - Add obstacles and hazards that players must navigate.
12. **Enhanced Collision Detection**:
    - Improve the collision detection system for more accurate interactions.
    - Add different hitboxes for different parts of the player and enemies.
13. **Tutorial and Help System**:
    - Create a comprehensive tutorial to help new players learn the game mechanics.
    - Add a help system with tips and tricks.
14. **Performance Optimization**:
    - Optimize the game for better performance on various hardware configurations.
    - Reduce loading times and improve frame rates.
15. **Localization and Accessibility**:
    - Translate the game into multiple languages.
    - Add accessibility options for players with disabilities.

ACKNOWLEDGMENTS

External Resources and Libraries:

1. **ConsoleGameEngine**: https://github.com/ollelogdahl/ConsoleGameEngine
    - A custom game engine for rendering graphics and handling game logic in the console.
    - Used for rendering text, handling input, and managing the game loop.
2. **CSCore**:
    - A .NET audio library used for handling sound playback.
    - Libraries used: CSCore, CSCore.Codecs, CSCore.SoundOut, CSCore.Streams.
3. **SharpDX.DirectInput**:
    - A library for handling input devices such as keyboards and gamepads.
    - Used for capturing keyboard input in the game.

4. **WindowsInput**:
    - A library for simulating keyboard and mouse input.
    - Used for simulating zoom out functionality in the game.
5. **System.Xml.Linq**:
    - A .NET library for working with XML data.
    - Used for reading and writing score data to an XML file.

@javdix9 - https://www.youtube.com/@javidx9

https://github.com/ollelogdahl/ConsoleGameEngine