Introduction

Overview

This project implements **BOOSE (Basic Object-Oriented Software Engineering) Interpreter**, a comprehensive medium-scale software engineering solution built with .NET 8. The interpreter provides a complete implementation of the BOOSE language runtime, enabling execution of custom domain-specific programs with support for variables, methods, control flow, and graphics rendering.

Project Purpose

The BOOSE Interpreter serves as an educational and functional implementation of a domain-specific language (DSL) designed for procedural programming with integrated graphics capabilities. It demonstrates core software engineering principles including architecture design, code organization, and systematic testing practices suitable for medium-scale applications.

Key Features

- **BOOSE Language Execution**: Full-featured interpreter that parses and executes BOOSE programs with support for:
    - Variable management (integer, real, and boolean types)
    - Method definitions with parameters and return values
    - Control flow structures (if/else, for loops, while loops)
    - Array operations (peek/poke functionality)
    - Boolean expressions with logical operators
- **Graphics Rendering**: Integrated canvas system enabling:
    - Shape drawing (circles, rectangles, triangles)
    - Color management via RGB values
    - Cursor positioning and line drawing
    - Text rendering capabilities
- **Robust Architecture**:
    - Interface-driven design with `IBooseRunner` and `IVariableManager`
    - Separation of concerns through `AppCanvas` and `VariableManager`
    - Comprehensive error handling and validation
    - Structured method definition storage

Technology Stack

- **Framework**: .NET 8
- **Language**: C#

- **UI Framework**: Windows Forms
- **Testing**: Microsoft Testing Framework (MSTest)

Project Structure

The solution is organized into several key components:

- **UnrestrictedBooseRunner**: Core interpreter engine responsible for parsing and executing BOOSE code
- **VariableManager**: Manages variable storage, scoping, and lifecycle
- **AppCanvas**: Graphics abstraction layer providing drawing primitives
- **Form1**: Windows Forms user interface for interacting with the interpreter
- **CommandFactory**: Factory pattern implementation for command handling
- **Test Suite**: Comprehensive unit tests validating interpreter functionality

Getting Started

To begin using the BOOSE Interpreter, refer to the Getting Started guide which covers installation, configuration, and basic usage examples.

Documentation

This documentation provides detailed information about:

- Setting up and configuring the interpreter
- Writing and executing BOOSE programs
- Architecture and design patterns
- API reference and method documentation
- Testing and quality assurance practices

Getting Started

Installation & Setup

Prerequisites

- **.NET 8 SDK** or later installed on your system
- **Visual Studio 2022** (recommended) or any compatible C# IDE
- **Windows Forms Runtime** (included with .NET 8)

Running the Application

1. **Clone or Open the Project**

2. **Open in Visual Studio**

    - Launch Visual Studio 2022
    - Open the solution file (`Medium_Scale_Software_Engineering_Project.sln`)
    - Build the project using **Build > Build Solution**
3. **Run the Application**

    - Press `F5` or select **Debug > Start Debugging**
    - The BOOSE Interpreter application window will launch

User Interface Overview

The BOOSE Interpreter features a Windows Forms-based interface with the following components:

Main Window Layout

Key Components

- **Multi-line Editor**: Write complex BOOSE programs with multiple statements
- **Single-line Input**: Execute quick one-line commands or statements
- **Canvas Area**: Visual output area where shapes and graphics are rendered
- **Debug Output**: Displays execution results, errors, and validation messages
- **Control Buttons**:
    - **Run**: Execute the code from either multi-line or single-line editor
    - **Clear**: Reset all text boxes and canvas
    - **Check**: Validate syntax without executing
    - **Save**: Export code, output, and canvas to a `.boose` file
    - **Load**: Import previously saved sessions

Writing Your First Program

Example 1: Simple Drawing

Draw a blue circle at position (100, 100) with radius 50:

```
// This is a comment
color(0, 0, 255) // Set color to blue
circle(100, 100, 50) // Draw a circle at (100, 100) with radius 50
```

Execution:

1. Enter the code in the **Multi-line Editor** or **Single-line Input**
2. Click **Run** or press `Enter`
3. A blue circle appears on the canvas at position (100, 100)

Example 2: Variables & Drawing

Draw rectangles with different sizes:

```
// This program demonstrates the use of variables
width = 200
height = 100
color(255, 0, 0) // Set color to red
rect(50, 50, width, height) // Draw a rectangle using variables for size
```

Execution:

1. Enter the code in the **Multi-line Editor** or **Single-line Input**
2. Click **Run** or press `Enter`
3. A red rectangle appears on the canvas with the specified width and height

Expected Result:

- First red rectangle (100  50) at position (50, 50)
- Second green rectangle (10050) at position (50, 150)

Example 3: Loops & Control Flow

Create a pattern using a loop:

```
// Draw a diagonal line of circles
color(0, 255, 0) // Set color to green
for i from 0 to 5
    circle(20*i, 20*i, 10) // Draw a circle at each step
```

Execution:

1. Enter the code in the **Multi-line Editor** or **Single-line Input**

2. Click **Run** or press `Enter`
3. A diagonal line of circles is drawn from the top-left to the bottom-right

Expected Result: A series of 6 blue circles arranged diagonally across the canvas.

Example 4: Boolean Variables & Conditions

Control drawing based on conditions:

```
// Draw based on a condition
drawCircles = true
color(255, 255, 0) // Set color to yellow
if drawCircles
    for i from 0 to 4
        circle(100 + 20*i, 100, 10) // Draw a circle at each step
```

Execution:

1. Enter the code in the **Multi-line Editor** or **Single-line Input**
2. Click **Run** or press `Enter`
3. Yellow circles are drawn in a row

Expected Result: A row of 5 yellow circles starting from position (100, 100). If `drawCircles` is false, no circles will be drawn.

Example 5: Methods with Parameters

Define and call a method:

```
// This program demonstrates defining and using a method
drawRectangle(x, y, w, h) {
    rect(x, y, w, h) // Draw a rectangle using the given parameters
}

// Draw two rectangles
color(255, 165, 0) // Set color to orange
drawRectangle(10, 10, 100, 50) // Call the method to draw a rectangle
drawRectangle(120, 10, 80, 40) // Call the method to draw another rectangle
```

Execution:

1. Enter the code in the **Multi-line Editor** or **Single-line Input**
2. Click **Run** or press `Enter`
3. Two rectangles are drawn at specified locations

Expected Result: An orange rectangle is drawn, and text "Shape drawn!" appears on the canvas.

Example 6: Handling Invalid Input Gracefully

Ensure the program can handle unexpected or invalid input without crashing:

```
// This program demonstrates error handling for invalid input
inputNumber = -10
if inputNumber < 0
    color(255, 0, 255) // Set color to magenta for error indication
    text("Invalid number! Please enter a positive value.", 10, 10)
else
    circle(inputNumber, inputNumber, 10) // Draw a circle if input is valid
```

Execution:

1. Enter the code in the **Multi-line Editor** or **Single-line Input**
2. Click **Run** or press `Enter`
3. If the number is negative, an error message is displayed instead of crashing

Expected Result: An error message in magenta text appears on the canvas if the number is negative.

Running Tests

1. Open **Test Explorer** in Visual Studio (**View > Test Explorer**)
2. Build the solution to discover all tests
3. Click **Run All** to execute all tests
4. Verify test methods have `[TestMethod]` attribute
5. Ensure test classes have `[TestClass]` attribute

Generating Documentation

To generate the documentation site using DocFX:

1. Navigate to the `BooseApp` folder (containing `docfx.json`)
2. Run `docfx docfx.json`
3. Open `_site/index.html` in your browser to view the documentation

Project Structure

- **BooseApp**: Main application with Windows Forms UI
- **Medium_Scale_Software_Engineering_Project_Test**: Test project with unit tests
- **docs**: Documentation files including this guide

Troubleshooting

Tests not discovered

- Ensure test classes have `[TestClass]` attribute
- Verify test methods have `[TestMethod]` attribute
- Check that MSTest packages are installed correctly
- Rebuild the solution

Application won't run

- Verify .NET 8 SDK is installed: `dotnet --version`
- Check that Visual Studio has the .NET desktop workload installed
- Try cleaning and rebuilding: **Build > Clean Solution** followed by **Build > Build Solution**

License

See LICENSE file for details.

Support & Contribution

For issues, questions, or contributions, please visit the GitHub repository.