FGBasic Interpreter - Complete Feature List

Core BASIC Language Features

Program Structure

- **Line Numbers**: Traditional BASIC line numbering (10, 20, 30, etc.)
- **Labels**: Modern label-based programming with `@LABEL` or `LABEL:` syntax
- **Auto Line Numbering**: Automatically assigns line numbers starting at 10, incrementing by 10

Control Flow

- **GOTO**: Jump to line number or label
- **GOSUB/RETURN**: Subroutine calls with stack-based return handling
- **IF...THEN**: Conditional execution with support for both jumps and inline statements
- **FOR...NEXT**: Loop constructs with optional STEP values
- **END**: Terminate program execution
- **REM**: Comments

Variable System

- **Typed Variables** with 6 data types:
  - `.b` (Byte): 8-bit signed integer (-128 to 127)
  - `.w` (Word): 16-bit signed integer (-32,768 to 32,767)
  - `.l` (Long): 32-bit signed integer
  - `.q` (Quick): 16.16 fixed-point arithmetic
  - `.f` (Float): Double-precision floating point (default)
  - `$` (String): Text strings
- **Automatic Range Clamping**: Values automatically clamped to type ranges
- **Case-Insensitive**: Variable names normalized to uppercase

Arrays

- **DIM**: Declare arrays with any variable type
- **Multi-dimensional Support**: Access elements with array(index) syntax
- **Typed Arrays**: Arrays inherit the type system (e.g., `DIM values.w(100)`)

Data Handling

- **DATA**: Store static data in program
- **READ**: Read values from DATA statements into variables
- **Queue-based Processing**: DATA statements processed in order

Input/Output

- **PRINT**: Output to console with formatting
  - `;` separator: No newline
  - `,` separator: Tab character
  - Automatic newline at end of statement
- **INPUT**: Prompt for user input (currently sets to 0.0)
- **LET**: Variable assignment (optional keyword)

Mathematical Operations

- **Arithmetic Operators**: +, −, *, /, ^ (power)
- **Comparison Operators**: =, <>, <, >, <=, >=
- **Precedence Handling**: Proper order of operations with parentheses

Built-in Functions

- **Trigonometric**: SIN(), COS(), TAN()
- **Mathematical**: SQR()/SQRT(), ABS(), INT(), POW()
- **Random**: RND() - Generate random numbers
- **String Functions**:
  - LEN() - String length
  - CHR$() - Character from ASCII code
  - ASC() - ASCII code from character

Number Formats

- **Decimal**: Standard numeric literals
- **Hexadecimal**: 0x prefix for hex values (e.g., 0xFF00FF)

Graphics Engine (GPU-Accelerated)

Screen Modes

10 predefined screen resolutions:

- Mode 0: 320×240
- Mode 1: 640×480 (default)
- Mode 2: 800×600
- Mode 3: 1024×768
- Mode 4: 1280×720 (HD)
- Mode 5: 1920×1080 (Full HD)
- Mode 6: 640×360
- Mode 7: 400×300
- Mode 8: 160×120 (Tiny)
- Mode 9: 1280×1024

Drawing Commands

- **CLS**: Clear screen (fill with black)
- **MODE/SCREEN**: Change screen resolution
- **LINE**: Draw lines with optional ARGB color
- **CIRCLE**: Draw circles with center point and radius
- **BOX**: Draw filled rectangles
- **PIXEL**: Set individual pixel colors
- **COLOR/COLOUR**: Set current drawing color (RGB or RGBA)

Graphics Features

- **Hardware Acceleration**: Java2D rendering with OpenGL/Direct3D support
- **Double Buffering**: Smooth rendering without flicker
- **Anti-aliasing**: High-quality rendering
- **Alpha Transparency**: Full ARGB color support
- **Separate Graphics Window**: Dedicated window for visual output

Sprite Engine

Sprite Management

- **256 Sprites**: Support for sprites 0-255
- **Two Creation Methods**:
    - SPRITE LOAD: Load from image file
    - SPRITE CREATE: Create programmatically with width, height, color

Sprite Commands

- **SPRITE SHOW/HIDE**: Toggle sprite visibility
- **SPRITE MOVE**: Set absolute position (x, y)
- **SPRITE MOVEBY**: Move relative to current position (dx, dy)
- **SPRITE SCALE**: Scale sprite (scaleX, scaleY)
- **SPRITE ROTATE**: Rotate sprite by angle in degrees
- **SPRITE FLIPH/FLIPV**: Flip horizontally or vertically
- **SPRITE PRIORITY**: Set rendering order (z-order)
- **SPRITE ALPHA**: Set transparency (0-255)
- **SPRITE SETPIXEL**: Modify individual sprite pixels
- **SPRITE GETX/GETY**: Get sprite position into variables (SPRITEX, SPRITEY)
- **SPRITE COLLISION**: Check collision between two sprites (sets COLLISION variable)
- **SPRITE CLEAR**: Remove all sprites

Sprite Features

- **Transformations**: Scale, rotation, flipping all supported
- **Priority Sorting**: Sprites render in priority order

- **Collision Detection**: Bounding box intersection testing
- **Per-Pixel Editing**: Direct pixel manipulation
- **Image Loading**: Support for common image formats (PNG, JPG, etc.)

Integrated Development Environment (IDE)

Code Editor

- **Syntax Input**: Multi-line code editor with monospace font
- **Syntax-aware**: Preserves line numbers and labels
- **Adjustable Font Size**: 8-32 point font size
- **Theme Support**: Light and Dark themes
- **Scroll Support**: Handle large programs

Output Console

- **Separate Output Area**: Dedicated console for program output
- **Error Reporting**: Highlighted error messages with line numbers
- **Auto-scroll**: Automatically scroll to latest output

Menu System

File Menu:

- New (Ctrl+N): Clear program
- Open (Ctrl+O): Load .bas/.basic files
- Save (Ctrl+S): Save programs
- Exit: Close application

Edit Menu:

- Change Theme: Toggle Light/Dark mode
- Change Font Size: Adjust editor font

Run Menu:

- Run (F5): Execute program
- Stop: Halt execution
- List: Display program with line numbers

Help Menu:

- Variable Types: Show type reference
- About: Application information

Toolbar

- Quick access buttons: New, Open, Save
- Run button (green, prominent)
- Stop button

- Visual feedback for actions

Advanced Features

Multi-threading

- **Virtual Threads**: Uses Java 24's virtual threads for program execution
- **Non-blocking IDE**: UI remains responsive during program execution
- **Concurrent Rendering**: Graphics render on separate thread

Modern Java 24 Features

- **Switch Expressions**: Clean, type-safe command handling
- **Records**: Immutable data structures (ForLoopContext, ScreenMode)
- **Text Blocks**: Multi-line string literals
- **Pattern Matching**: Enhanced type checking
- **Math.clamp()**: Built-in range clamping
- **Stream API**: Modern collection processing

Error Handling

- **Line Number Reporting**: Errors show exact line number
- **Exception Stack Traces**: Full debugging information
- **Graceful Degradation**: Errors don't crash IDE
- **User-friendly Messages**: Clear error descriptions

File Operations

- **Program Save/Load**: Preserve programs between sessions
- **File Format**: Plain text .bas files
- **Automatic Extensions**: Adds .bas if no extension provided
- **File Filters**: Only show BASIC files in dialogs

Performance Optimizations

- **Hardware Acceleration**: GPU rendering for graphics
- **Efficient Rendering**: Only redraw when necessary
- **Optimized Arithmetic**: Fast expression evaluation
- **Memory Management**: Efficient variable and array storage

Programming Model

Execution Model

- **Interpreted**: Line-by-line execution
- **Sequential**: Linear program flow with jumps

- **Stack-based**: GOSUB/RETURN use call stack
- **Stateful**: Variables persist during execution

Expression Evaluation

- **Recursive Descent Parser**: Proper operator precedence
- **Type Coercion**: Automatic numeric/string conversions
- **Function Calls**: Nested function evaluation
- **Array Access**: Dynamic index calculation

Compatibility

- **Classic BASIC Syntax**: Familiar to BASIC programmers
- **Modern Enhancements**: Labels, typed variables, sprites
- **Cross-platform**: Pure Java, runs anywhere
- **No External Dependencies**: Self-contained (except JDK)

This is a fully-featured BASIC interpreter with modern graphics capabilities, making it suitable for learning programming, creating simple games, or demonstrating classic BASIC concepts with contemporary enhancements!