Modern Basic Engine

Manual & Reference Guide

Version 1.0

From Banditware — made by Willem van Uden

Table of Contents

1. Introduction

Modern Basic Engine is a graphical game engine and integrated development environment (IDE) that lets you create retro-style games using the **Modern Basic** programming language. It is inspired by classic BASIC dialects like QBasic and GW-BASIC, but with modern additions for graphics, sprites, scrolling worlds, gamepad input, and peer-to-peer networking.

The engine runs entirely in your web browser — no installation or internet connection required. Projects are stored locally in your browser.

2. Installation & Getting Started

Step 1: Download

Download the file `modern-basic-engine.html` to your computer.

Step 2: Open

Double-click the file. It will automatically open in your default web browser (Firefox, Chrome, etc.).

Step 3: Ready!

You will immediately see the IDE with a code editor, game view, and console. You can start programming right away.

> **Tip:** No internet required! The file works completely offline.

Fullscreen Mode

Click the **Fullscreen** button (⛶) above the game view to expand the game canvas to fill your entire screen. Click it again — or press `Escape` — to return to the normal editing layout.

3. The IDE Environment

The IDE consists of the following parts:

Component	Description
Menu Bar	File, Edit, Run and Help menus
Code Editor	Left side: write your Modern Basic code with syntax highlighting
Game View	Top right: graphical output of your program. Includes a **Fullscreen** button (⛶) to expand the view to the whole screen
Console	Bottom right: text output (PRINT) and input (INPUT)
Sidebar	Manage projects (open via File → Open Project)
Status Bar	Bottom: status (Running/Stopped), line number, variable count

4. Keyboard Shortcuts

Key	Action
`F5`	Run program
`F6`	Stop program
`F8`	Step-by-step execution (debug)
`Ctrl+S`	Save project
`Ctrl+N`	New project
`Ctrl+O`	Open/close sidebar
`Ctrl+Z`	Undo
`Ctrl+Y`	Redo

5. The Modern Basic Language

Modern Basic is a simple, readable programming language. Each line contains one command. Comments start with `REM` or an apostrophe (`'`).

```
REM This is a comment
' This too
PRINT "Hello World!"
LET X = 42
```

> **Note:** Modern Basic is case-insensitive. `PRINT` , `Print` and `print` all work the same.

6. Basic Commands

Command	Description	Example
`PRINT expr`	Write text or value to the console	`PRINT "Hello!"`
`PRINT a ; b`	Concatenate multiple values	`PRINT "Score: " ; S`
`INPUT var`	Read input from the user	`INPUT NAME`
`INPUT "text", var`	Input with custom prompt	`INPUT "Your name? ", N`
`LET var = expr`	Assign a value to a variable	`LET X = 10 + 5`
`var = expr`	Short form (without LET)	`SCORE = SCORE + 1`
`DIM name(size)`	Create an array	`DIM SCORES(100)`
`END`	End the program	`END`
`STOP`	Same as END	`STOP`

7. Variables & Expressions

Variables are automatically created on first use. They can hold numbers or text.

```
LET NAME = "John"
LET AGE = 25
LET RESULT = AGE * 2 + 10
PRINT NAME ; " is " ; AGE ; " years old"
```

Arithmetic Operators

Operator	Meaning	Example
`+`	Addition (or string concatenation)	`5 + 3` → 8
`-`	Subtraction	`10 - 4` → 6
`*`	Multiplication	`3 * 7` → 21
`/`	Division	`20 / 4` → 5
`MOD`	Modulo (remainder)	`10 MOD 3` → 1

Comparison Operators

Operator	Meaning
`=`	Equal to
`<>`	Not equal to
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to

Logical Operators

`AND` and `OR` combine conditions: `IF X > 0 AND X < 100 THEN ...`

8. Control Structures

IF / THEN / ELSE

```
REM Single-line IF
IF SCORE > 100 THEN PRINT "You win!"

REM IF with ELSE
IF X > 0 THEN PRINT "Positive" ELSE PRINT "Negative or zero"

REM Multi-line IF block
IF LIVES = 0 THEN
  PRINT "Game Over"
  END
ELSE
  PRINT "Lives remaining: " ; LIVES
ENDIF
```

FOR / NEXT (Loop)

```
FOR I = 1 TO 10
  PRINT I
NEXT I

REM With STEP
FOR I = 10 TO 0 STEP -1
  PRINT I
NEXT I
```

WHILE / WEND

```
LET X = 0
WHILE X < 100
  LET X = X + 1
WEND
```

DO / LOOP

```
DO
  LET K$ = INKEY$
  SLEEP 16
LOOP UNTIL K$ <> ""
```

GOTO and Labels

```
MAINMENU:
PRINT "Choose 1 or 2"
INPUT CHOICE
IF CHOICE = 1 THEN GOTO GAME
GOTO MAINMENU

GAME:
PRINT "The game begins!"
```

GOSUB / RETURN (Subroutines)

```
GOSUB DRAWSCREEN
END

DRAWSCREEN:
CLS
DRAW "Hello", 100, 100, 24
RETURN
```

9. Graphics Commands

Command	Description
`SCREEN width, height`	Set the canvas size (default 640x480)
`CLS`	Clear the screen (fill with background color)
`COLOR fg [, bg]`	Set drawing and background color
`LINE x1, y1, x2, y2 [, color]`	Draw a line
`CIRCLE x, y, radius [, color [, fill]]`	Draw a circle (fill=1 for filled)
`RECT x, y, w, h [, color [, fill]]`	Draw a rectangle
`PLOT x, y [, color]`	Draw a pixel
`DRAW "text", x, y [, size]`	Draw text on the canvas

Example: Drawing Shapes

```
SCREEN 640, 480
COLOR 10
RECT 50, 50, 200, 100, 10, 1  ' Filled green rectangle
COLOR 12
CIRCLE 400, 200, 80, 12, 1    ' Filled red circle
COLOR 14
LINE 0, 0, 640, 480             ' Yellow diagonal line
COLOR 15
DRAW "Modern Basic Engine", 180, 400, 24
```

10. Sprites

Sprites are movable objects on the screen, ideal for game characters and items.

Command	Description
`SPRITE id, x, y, w, h [, color]`	Create a sprite with position and size
`MOVESPRITE id, x, y`	Move a sprite to a new position

```
SPRITE 1, 100, 200, 32, 32, 10  ' Green 32x32 sprite
MOVESPRITE 1, 150, 200                ' Move to the right
```

11. Sprite Images

You can load external images and assign them to sprites for rich visual games.

Command	Description
`LOADSPRITE imgId, "player.png"`	Load a local image (by filename) into image slot imgId. Load the file first via *File → Open Folder* or *Load Local Assets*. A full URL also works.
`LOADSPRITEFILE imgId`	Optional: open a file picker at run time to choose a PNG for the slot
`SETSPRITEIMG spriteId, imgId`	Assign a loaded image to a sprite
`HIDESPRITE id`	Make a sprite invisible
`SHOWSPRITE id`	Make a sprite visible again

Example: Loading and Using Sprite Images

```
SCREEN 640, 480

REM Load an image into slot 1
LOADSPRITE 1, "player.png"

REM Create a sprite and assign the image
SPRITE 1, 100, 100, 64, 64
SETSPRITEIMG 1, 1

REM Hide and show
HIDESPRITE 1   ' Sprite becomes invisible
SHOWSPRITE 1   ' Sprite becomes visible again
```

> **Tip:** First load your images into the engine with *File → Open Folder (with assets)* or *Load Local Assets*, then reference them by filename (e.g. `"player.png"`).

12. Scrolling & Viewport

Create worlds larger than the visible screen and scroll through them. This is essential for side-scrolling or top-down exploration games.

Command / Expression	Description
`WORLD w, h`	Define the world size (larger than the screen)
`SCROLL x, y`	Set the scroll offset (camera position)
`VIEWPORT x, y`	Alias for SCROLL — set the camera position
`SCROLLX`	Returns the current horizontal scroll offset
`SCROLLY`	Returns the current vertical scroll offset
`WORLDW`	Returns the world width
`WORLDH`	Returns the world height

Example: Side-Scrolling World

```
SCREEN 640, 480
WORLD 3000, 480  ' Wide world
LET PX = 320

GAMELOOP:
  LET K$ = INKEY$
  IF K$ = "ArrowRight" THEN LET PX = PX + 4
  IF K$ = "ArrowLeft" THEN LET PX = PX - 4

  REM Follow the player
  SCROLL PX - 320, 0

  CLS
  RECT PX, 200, 32, 32, 10, 1
  SLEEP 16
GOTO GAMELOOP
```

> **Tip:** Drawing coordinates are in world space. The engine automatically offsets them by the scroll position. Objects outside the visible viewport are simply not visible.

13. Sound & Music

Command	Description
`SOUND frequency, duration`	Play a tone (frequency in Hz, duration in ms)
`PLAY frequency, duration`	Same as SOUND
`PLAY "song.mp3"`	Play a local audio file by filename (load it first via *Open Folder* or *Load Local Assets*). A full URL also works.
`PLAYFILE`	Optional: open a file picker at run time to choose an MP3
`STOPMUSIC`	Stop the currently playing music
`MUSICPLAYING`	Returns 1 if music is playing, 0 if not

Sound Tones

```
SOUND 440, 500   ' Tone A4, half second
SOUND 880, 250   ' Tone A5, quarter second
SOUND 200, 1000  ' Low tone, 1 second
```

MP3 Music Playback

```
PLAY "song.mp3"  ' Play a local file by name
PLAYFILE           ' Optional: pick an MP3 at run time

REM Wait for music to finish
WAITMUSIC:
  IF MUSICPLAYING THEN GOTO WAITMUSIC
  PRINT "Music finished!"

STOPMUSIC  ' Stop music manually
```

14. Input (Keyboard & Mouse)

Command / Expression	Description
`INKEY$`	Returns the key currently held down (empty if none). Best for continuous movement.
`GETKEY var$`	Reads the next key from the input queue into var$ (empty if none). Each key press is delivered exactly once - best for typing and menus.
`MOUSEX`	X-position of the mouse on the canvas
`MOUSEY`	Y-position of the mouse on the canvas
`MOUSEBUTTON`	1 if mouse button is pressed, otherwise 0
`SLEEP ms`	Wait for the specified number of milliseconds
`WAIT`	Wait until next frame (~60fps, 16ms)

Key Names

Common key names for `INKEY$`:

Key	Value
Arrow Up	`"ArrowUp"`
Arrow Down	`"ArrowDown"`
Arrow Left	`"ArrowLeft"`
Arrow Right	`"ArrowRight"`
Space Bar	`" "`
Enter	`"Enter"`
Escape	`"Escape"`
Letters	`"a"`, `"b"`, `"A"`, `"B"`, etc.

```
REM Movement example (INKEY$ - held key)
LET K$ = INKEY$
```

```
IF K$ = "ArrowUp" THEN LET Y = Y - 5
IF K$ = "ArrowDown" THEN LET Y = Y + 5
```

INKEY$ vs GETKEY

`INKEY$` reports the key that is currently held down, sampled once per frame. This is perfect for movement (holding an arrow keeps moving) but it is **not** reliable for typing: fast presses can be missed and a held key repeats every frame.

`GETKEY` pulls from an input queue where **every** individual key press is stored exactly once, in order. Use it for text entry (chat, name input), menus, and anywhere you need each press to count once - even when the player types quickly. Note that `GETKEY` returns the raw key name, so a space bar press gives `" "`, Enter gives `"Enter"`, etc.

```
REM Typing example (GETKEY - one press each)
LET NAME$ = ""
TYPELOOP:
  LET KE$ = ""
  GETKEY KE$
  IF KE$ = "Backspace" THEN LET NAME$ = LEFT$(NAME$, LEN(NAME$) - 1)
  IF LEN(KE$) = 1 THEN LET NAME$ = NAME$ + KE$
  CLS
  DRAW "Name: " + NAME$, 40, 200, 24
  SLEEP 33
GOTO TYPELOOP
```

> **Important:** Click on the game view (canvas) before using keys! The canvas must have focus to receive key presses.

15. Gamepad Support

Modern Basic Engine supports gamepads and game controllers (via the browser Gamepad API). Connect a controller and use these expressions:

Expression	Description
`GAMEPAD`	Returns 1 if a gamepad is connected, 0 otherwise
`GPBUTTON(n)`	Returns 1 if button n is pressed, 0 otherwise
`GPAXIS(n)`	Returns the axis value (-1.0 to 1.0)

Common Button Numbers

Button	Standard Mapping
0	A / Cross
1	B / Circle
2	X / Square
3	Y / Triangle
12	D-Pad Up
13	D-Pad Down
14	D-Pad Left
15	D-Pad Right

Common Axis Numbers

Axis	Direction
0	Left stick horizontal (-1 = left, 1 = right)
1	Left stick vertical (-1 = up, 1 = down)
2	Right stick horizontal
3	Right stick vertical

Example: Gamepad Movement

```
SCREEN 640, 480
LET X = 320 : LET Y = 240

GAMELOOP:
  IF GAMEPAD THEN
    LET X = X + GPAXIS(0) * 4
    LET Y = Y + GPAXIS(1) * 4
    IF GPBUTTON(0) THEN SOUND 440, 100
  ENDIF
  CLS
  CIRCLE X, Y, 16, 10, 1
```

```
  SLEEP 16
GOTO GAMELOOP
```

16. Network / Multiplayer

Modern Basic Engine includes peer-to-peer networking for multiplayer games using WebRTC. One player hosts a session and others join using the host's peer ID.

Command / Expression	Description
`NETHOST`	Start hosting a network session
`NETJOIN "peerId"`	Join a host's session using their peer ID
`NETSEND "message"`	Send a message to all connected peers
`NETRECV`	Receive the next message (empty string if none)
`NETCLOSE`	Close the network connection
`NETACTIVE`	Returns 1 if the network connection is active
`NETHASDATA`	Returns 1 if there are unread messages
`NETPEERID$`	Returns your own peer ID (as string)

Example: Simple Chat

```
REM Host side
NETHOST
PRINT "Your ID: " ; NETPEERID$
PRINT "Waiting for connection..."

REM Wait until active
WHILE NETACTIVE = 0
  SLEEP 100
WEND

CHATLOOP:
  IF NETHASDATA THEN
    LET MSG$ = NETRECV
    PRINT "Peer: " ; MSG$
  ENDIF
  SLEEP 50
GOTO CHATLOOP
```

> **Tip:** The host must share their `NETPEERID$` with the other player(s). The join side uses `NETJOIN` with that ID to establish the connection.

17. Built-in Functions

Function	Description	Example
`RND`	Random number between 0 and 1	`LET X = RND`
`RND(n)`	Random integer from 0 to n-1	`LET D = RND(6) + 1`
`INT(x)`	Floor (round down)	`INT(3.7)` → 3
`ABS(x)`	Absolute value	`ABS(-5)` → 5
`SQR(x)`	Square root	`SQR(25)` → 5
`SIN(x)`	Sine (radians)	`SIN(3.14)`
`COS(x)`	Cosine (radians)	`COS(0)` → 1
`TAN(x)`	Tangent (radians)	`TAN(0.785)`
`LOG(x)`	Natural logarithm	`LOG(2.718)`
`SGN(x)`	Sign: -1, 0 or 1	`SGN(-7)` → -1
`NOT(x)`	Logical negation	`NOT(0)` → 1
`LEN(s$)`	Length of a string	`LEN("Hello")` → 5
`STR$(n)`	Number to string	`STR$(42)` → "42"
`VAL(s$)`	String to number	`VAL("3.14")` → 3.14
`CHR$(n)`	ASCII code to character	`CHR$(65)` → "A"
`ASC(s$)`	Character to ASCII code	`ASC("A")` → 65
`TIMER`	Current time in milliseconds	`LET T = TIMER`
`PI`	Constant π (3.14159...)	`LET C = 2 * PI * R`

18. Color Table

Use color numbers (0-15) or color names with the `COLOR` command:

Nr	Name	Color	Hex
0	BLACK	Black	#000000
1	BLUE	Dark Blue	#0000AA
2	GREEN	Dark Green	#00AA00
3	CYAN	Cyan	#00AAAA
4	RED	Dark Red	#AA0000
5	MAGENTA	Magenta	#AA00AA
6	BROWN	Brown	#AA5500
7	GRAY	Light Gray	#AAAAAA
8		Dark Gray	#555555
9		Light Blue	#5555FF
10		Light Green	#55FF55
11		Light Cyan	#55FFFF
12		Light Red	#FF5555
13	PINK	Pink	#FF55FF
14	YELLOW	Yellow	#FFFF55
15	WHITE	White	#FFFFFF

Extra color names: `ORANGE` (#FF8800)

You can also use hex colors: `COLOR "#FF6600"`

19. Creating a Game Loop

The foundation of every game is a **game loop**: a repeating cycle of reading input, updating logic, and drawing the screen.

```
SCREEN 640, 480
LET X = 320
LET Y = 240
LET SPEED = 4

GAMELOOP:
REM 1. Read input
LET K$ = INKEY$
IF K$ = "ArrowUp" THEN LET Y = Y - SPEED
IF K$ = "ArrowDown" THEN LET Y = Y + SPEED
IF K$ = "ArrowLeft" THEN LET X = X - SPEED
IF K$ = "ArrowRight" THEN LET X = X + SPEED

REM 2. Draw the screen
CLS
COLOR 10
RECT X, Y, 20, 20, 10, 1
COLOR 15
DRAW "Move with arrow keys", 10, 10, 14

REM 3. Wait for next frame
SLEEP 16
GOTO GAMELOOP
```

> **Tip:** Use `SLEEP 16` for ~60 frames per second, or `WAIT` for automatic frame synchronization.

20. Example: Snake

The Snake game demonstrates arrays, subroutines, collision detection and game loops. The snake is controlled with arrow keys.

Key Techniques:

- **Arrays** `DIM SX(200)` and `DIM SY(200)` store the positions of each snake segment
- **Subroutines** `GOSUB CHECKSELF` checks whether the snake hits itself
- **Control** via `INKEY$` and the variable `DIR` (direction 0-3)
- **Difficulty** increases: `SPEED` decreases with each apple collected

Load the Snake example via the sidebar to view the full code.

21. Example: Pong

Pong is a two-player game with paddles and a ball. Player 1 uses W/S, Player 2 uses the arrow keys.

Key Techniques:

- **Two players** with separate keys via `INKEY$`
- **Ball movement** with `BDX` and `BDY` (direction and speed)
- **Collision detection** with paddles via nested IF statements
- **Sound** on each collision via `SOUND`

22. Tips & Tricks

- **Debugging:** Use F8 to execute your program step by step and inspect the state
- **Save regularly:** Press Ctrl+S to save your work in the browser
- **Canvas focus:** Click on the game view if keyboard input does not work
- **Watch variables:** The status bar shows the number of active variables
- **Comments:** Use `REM` to document your code
- **Labels:** Use descriptive names for labels, like `GAMELOOP:` or `DRAWMENU:`
- **Subroutines:** Use `GOSUB` / `RETURN` to organize repetitive code
- **Error messages:** Check the console for error messages if your program does not work

- **.bas files:** Import and export your code as `.bas` files via the File menu
- **Gamepad:** Connect a controller and use `GAMEPAD`, `GPBUTTON()`, `GPAXIS()` for controller input
- **Large worlds:** Use `WORLD` and `SCROLL` to create scrolling levels
- **Multiplayer:** Use `NETHOST` and `NETJOIN` for peer-to-peer multiplayer games
- **Fullscreen:** Click the Fullscreen button (⛶) above the game view to play at full screen size; press `Escape` to exit

---