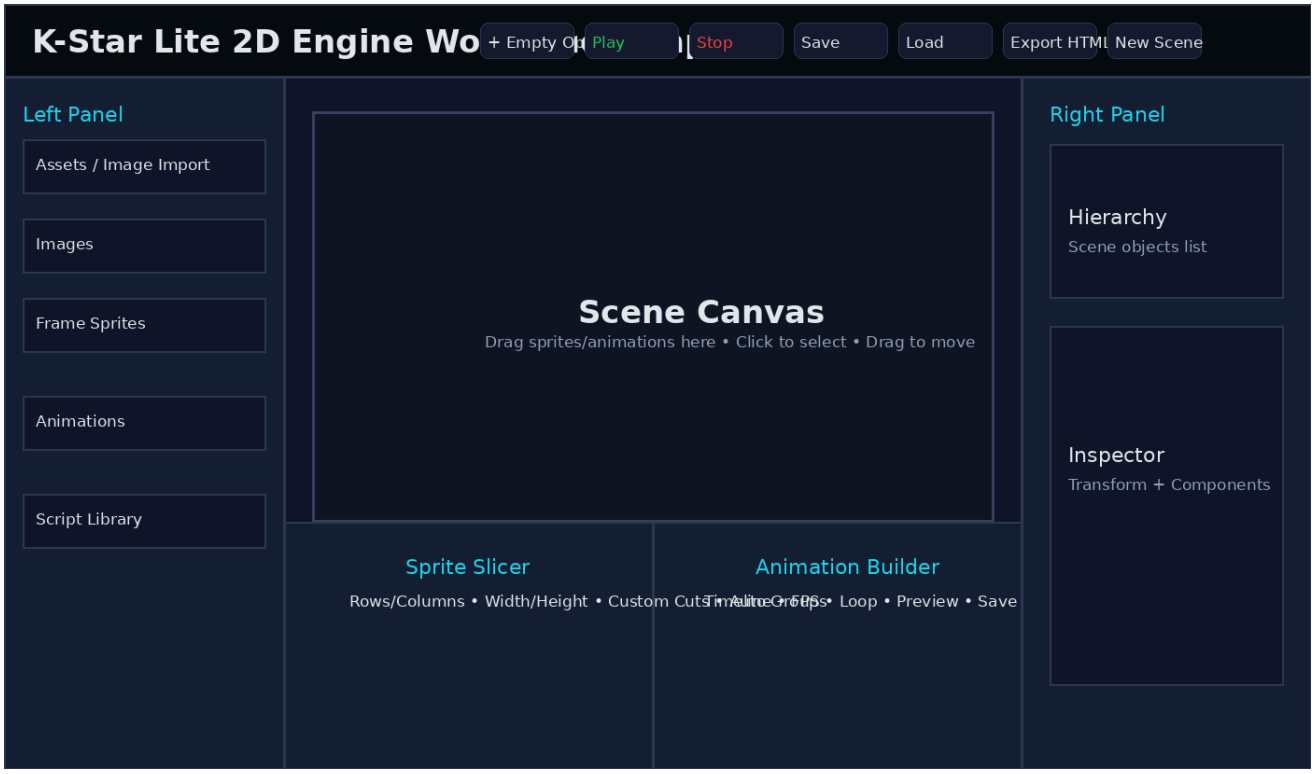
K-Star Lite 2D Engine

Starter User Manual

Version 0.1 — built from kstar_lite_2d_engine(6).zip

Prepared for K-Star



Manual status
This is a starter manual. It documents the current MVP build and is structured so future sections can be expanded as new tools, components, prefabs, project files, and export options are added.

Table of Contents

1. What the Engine Does

K-Star Lite 2D Engine is a lightweight browser-based 2D game editor. It runs from a single HTML file and lets you build simple sprite-based scenes, slice sprite sheets, build animations, attach components, test with Play Mode, save to browser localStorage, and export the current project as a standalone playable HTML game.

- Best for: rapid 2D prototyping, sprite animation testing, small HTML canvas games, teaching component-based game logic, and building lightweight web game demos.
- Main editor file: lightweight_game_engine.html.
- Current project save system: browser localStorage under the current browser/file origin.
- Export output: one playable HTML file containing the runtime, assets, scene data, and scripts.

Simple mental model
Import an image, slice it into frame sprites, turn frames into animations, drag sprites or animations into the scene, add components in the Inspector, press Play, then export the working game HTML.

2. Quick Start

2.1. Open the engine

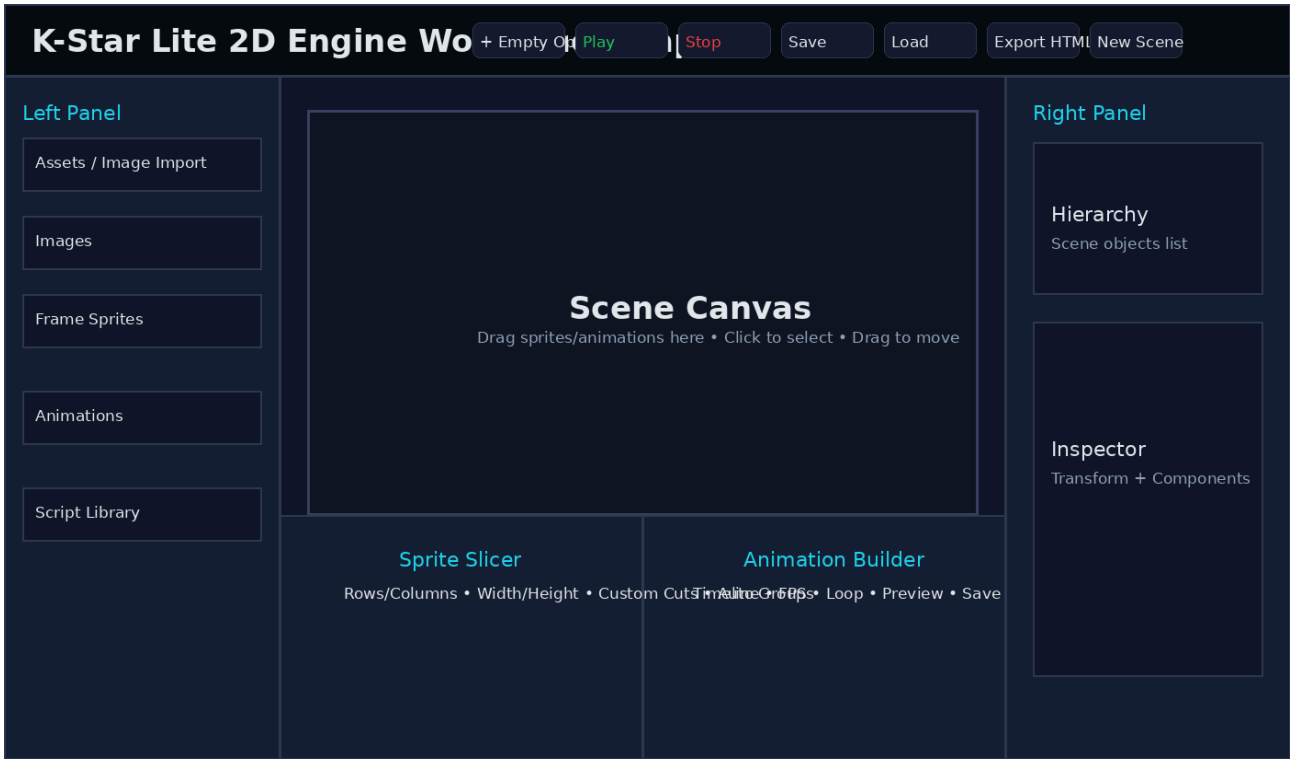
1. Unzip the K-Star Lite Engine folder.
2. Open lightweight_game_engine.html in a modern desktop browser such as Chrome, Edge, or Firefox.
3. The editor should open with a default scene containing a Camera marker and a Ground object.
4. Click the scene canvas once before testing keyboard controls, especially after pressing Play.

2.2. Build your first animated object

5. Use the Assets file picker to import a sprite sheet image.
6. Select the imported image in the Images list.
7. Choose a Sprite Slicer mode, preview the cut boxes, then click Slice.
8. Click frame sprites in the Frame Sprites list to add them to the Animation Builder timeline.
9. Name the animation, set FPS, choose Loop true or false, then click Save Animation.
10. Drag the saved animation from the Animations list onto the Scene Canvas.
11. Select the new object and adjust its Transform or components in the Inspector.
12. Press Play to test the animation, then Stop to return to Edit Mode.

3. Workspace Overview

The editor is divided into five main zones: toolbar, left asset panel, center scene canvas, bottom editors, and right hierarchy/inspector panel.



Area	Purpose	Main actions
Toolbar	Global project controls.	+ Empty Object, Play, Stop, Save Project, Load Project, Export Game HTML, New Scene.
Left Panel	Asset management and script editing.	Import images, view images, manage frame sprites, manage animations, edit script library.
Scene Canvas	Visual stage where the game scene is assembled.	Drag assets into scene, select objects, move selected object, view colliders and grid.
Sprite Slicer	Turns imported images into frame sprites.	Rows/columns slicing, width/height slicing, custom cuts, auto pixel groups, zoom controls.
Animation Builder	Turns frame sprites into named animations.	Add frames, reorder frames, remove frames, set FPS, set loop, preview, save.
Hierarchy	List of scene objects.	Select objects and delete objects.
Inspector	Edit selected object and components.	Rename, activate/deactivate, duplicate, delete, transform, add/remove components.

3.1. Pop-out panels

Major editor panels have pop-out buttons. A pop-out panel can be dragged and resized, which is especially useful for the Sprite Slicer when working with large sprite sheets. Dock the panel again when you want it back in the main layout.

4. Importing Assets

The engine imports images directly into the browser. Imported images appear in the Images list and can be selected for slicing.

13. Click the image file input under Assets.
14. Choose a PNG, JPG, WEBP, or other browser-supported image file.
15. After import, the image appears in the Images list with its pixel size.
16. Click the image in the Images list to make it active in the Sprite Slicer.
17. Use Delete beside an image only when you are sure you no longer need it.

Asset deletion behavior

Deleting an imported image removes the source image from the asset list. Frame sprites already sliced from that image may still exist unless they are deleted separately. Deleting a frame sprite cleans references from animations and scene objects.

5. Sprite Slicer

The Sprite Slicer creates trimmed frame sprites from the selected image. Transparency is ignored when each generated frame is created, so the saved frame is automatically trimmed to the furthest visible pixels.

Mode	Use when	Controls	Typical result
Rows / Columns	A sprite sheet is arranged in an even grid.	Rows, Columns, Preview Cuts, Slice.	One frame per grid cell.
Width / Height	Every frame has the same pixel width and height.	Frame W, Frame H, Preview Cuts, Slice.	Frames based on fixed pixel dimensions.
Custom Mouse Cuts	Frames are irregular or need manual selection.	Click-drag rectangles, Clear Cuts, Slice Custom Cuts.	One frame per drawn rectangle.
Auto Pixel Groups	Each sprite has transparent spacing around it.	Alpha, Min Pixels, Merge Gap, Padding, Auto Preview, Slice Auto Groups.	Frames based on detected non-transparent groups.

5.1. Rows / Columns workflow

18. Select an imported image.
19. Open the Rows / Columns tab.
20. Enter the number of rows and columns in the sprite sheet.
21. Click Preview Cuts and confirm the boxes line up with the frames.
22. Click Slice to create trimmed frame sprites.

5.2. Width / Height workflow

23. Select an imported image.
24. Open the Width / Height tab.
25. Enter the exact frame width and height in pixels.
26. Click Preview Cuts and inspect the cut boxes.
27. Click Slice to create frame sprites.

5.3. Custom Mouse Cuts workflow

28. Select an imported image.
29. Open the Custom Mouse Cuts tab.
30. Click and drag over each sprite frame you want to capture.
31. Use Clear Cuts if you need to restart the manual selections.
32. Click Slice Custom Cuts to create the selected frames.

5.4. Auto Pixel Groups workflow

33. Select an imported image that has transparent space separating the sprites.
34. Open the Auto Pixel Groups tab.
35. Click Auto Preview to see detected groups.
36. Adjust Alpha, Min Pixels, Merge Gap, and Padding until each group matches one intended sprite.
37. Click Slice Auto Groups to create trimmed frame sprites.

Fixing auto-slice frames that grab multiple sprites
Increase the spacing in the source image, reduce Merge Gap, or switch to Custom Mouse Cuts. If a sheet has touching pixels between two sprites, Auto Pixel Groups may treat them as one connected group.

6. Animation Builder

The Animation Builder creates named animations from frame sprites. Click frame sprites in the Frame Sprites list to add them to the timeline. You can reorder frames by dragging timeline chips and remove frames with the × button.

38. Click frame sprites in the order the animation should play.
39. Enter a clear animation name, such as Player_Walk_Right or Coin_Spin.
40. Set FPS. Higher FPS plays faster; lower FPS plays slower.
41. Set Loop to true for walk cycles, idle animations, spinning items, and repeated effects.
42. Set Loop to false for one-shot effects such as an attack swing, impact, or explosion.
43. Click Save Animation.

Editing an existing animation
Click Edit on an animation in the Animations list to load its frames into the builder. In the current build, saving creates a new animation entry rather than overwriting the old one, so delete the older version when it is no longer needed.

7. Scene Editing

The Scene Canvas is the stage where objects are placed. You can drag frame sprites or animations from the left panel into the scene. The engine creates a scene object with the needed rendering component automatically.

- Drag a frame sprite into the scene to create a static sprite object with a SpriteRenderer component.
- Drag an animation into the scene to create an animated object with SpriteRenderer and Animator components.
- Click an object in the scene or Hierarchy to select it.
- Drag the selected object in the scene to reposition it.

- Use the Inspector to precisely edit X, Y, rotation, scale, components, and settings.
- Use + Empty Object to create an object without a sprite, useful for invisible triggers, spawners, markers, or future logic.

7.1. Hierarchy

The Hierarchy lists all scene objects. Select an object from the Hierarchy when it is hard to click in the Scene Canvas. Use the Delete button beside an object to remove it from the scene.

7.2. Object selection and deletion

When an object is selected, the Inspector shows its settings. The selected object also receives a dashed selection outline in the scene. Deleting a scene object removes that object only; it does not delete the source sprite or animation asset.

8. Inspector and Components

The Inspector is where selected scene objects are named, activated, transformed, duplicated, deleted, and given behavior through components.

Component	Purpose	Key fields	Notes
Transform	Position, rotation, and scale for every object.	X, Y, Rotation, Scale X, Scale Y.	Always present as object data, not listed as removable.
SpriteRenderer	Draws a frame sprite or fallback colored block.	Sprite, Opacity, Layer, Flip X, Flip Y.	Layer controls draw order; lower layers render earlier.
Animator	Plays a saved animation on the object.	Animation, Playing, Loop, FPS Override.	FPS Override uses animation FPS when set to 0.
Rigidbody	Adds movement and gravity-style physics.	Velocity, Acceleration, Gravity Scale, Mass, Drag, Kinematic.	Kinematic objects do not move from physics.
BoxCollider	Rectangular collision area.	Width, Height, Offset X, Offset Y, Trigger.	Good for platforms, walls, and rectangular objects.
CircleCollider	Circular collision area.	Radius, Offset X, Offset Y, Trigger.	Good for balls, coins, projectiles, or round enemies.
PolygonCollider	Custom polygon outline shown in editor.	Points, Offset X, Offset Y, Trigger.	Current physics uses its bounding box for collision resolution.
Script	Runs custom behavior from the Script Library.	Script Name.	Script Name must match an Engine.registerScript name.
Camera	Default marker for future camera behavior.	No current controls.	The editor and exported runtime currently use a fixed 960×540 viewport.

8.1. Adding components

44. Select an object.
45. Scroll to Add Component in the Inspector.

46. Choose the component type from the dropdown.
47. Click Add.
48. Fill in the new component fields.

8.2. Triggers vs. solid colliders

Collider components include a Trigger option. A solid collider can physically resolve collisions when Rigidbody components are involved. A trigger is useful for detecting overlaps without pushing objects apart, such as pickups, checkpoints, doors, or invisible event zones.

8.5. Particle Effects (New Section)

✨ Particle System Editor

The Particle Effects tool (in the bottom-right **FX + Tile Tools** panel) lets you create and reuse simple particle systems such as sparks, explosions, fire, smoke, magic bursts, dust, or coin glints. Once saved, you can drag them into the scene or emit them from scripts.

Key Features

- Real-time preview burst on the small canvas.
- Parameters you can tweak live.
- Attach via **ParticleEmitter** component or call from scripts with Engine.particles.emit(...).

Creating a Particle Effect

1. Go to the **FX + Tile Tools** panel → **Particle System Editor**.
2. Adjust the parameters:
    - **Name**: Clear name like ExplosionBurst, WalkDust, MagicSparkle.
    - **Color**: Base particle color (can be overridden in scripts).
    - **Max**: Maximum number of particles alive at once.
    - **Rate**: Particles per second (continuous emission).
    - **Life**: Particle lifetime in seconds.
    - **Speed**: Initial speed in pixels per second.
    - **Size**: Base particle radius.
    - **Spread°**: Emission angle spread (360 = omnidirectional).
    - **Gravity**: Downward force (positive = falls, negative = rises).
    - **Burst**: Number of particles for one-shot emissions.
3. Click **Preview Burst** to test the current settings on the preview canvas.
4. When happy, click **Save Particle FX**. It appears in the **Particle Effects** list in the left panel.

Using Particle Effects in the Scene

Method 1 – ParticleEmitter Component (Recommended for looping/continuous effects)

1. Select or create a GameObject (e.g., a player, campfire, or empty object).
2. In the Inspector, add a **ParticleEmitter** component.
3. Set **Effect ID** to the exact name (or ID) of your saved particle effect.

4. Enable **Play On Start** for automatic emission when the scene starts.
5. Enable **Loop** for continuous emission.
6. Optionally set a custom **Rate** or **Burst On Start**.

Method 2 – Drag & Drop

- Drag a particle effect from the left **Particle Effects** list onto the scene. The engine automatically creates an object with a ParticleEmitter.

Method 3 – From Scripts

JavaScript

```
// One-shot burst
Engine.particles.emit("ExplosionBurst", x, y, 48, {angle: -90, color: "#ffaa00"});

// Continuous emission handled by the component
```

Tips for Good Particles

- Use **Preview Burst** frequently.
- For fire/smoke: low gravity + upward spread + color variation via script.
- For sparks/trails: short life, high speed, low gravity.
- Performance: Keep **Max** reasonable (under 300 total particles) in exported games.

8.6. Tile Map Editor (New Section)

🧱 Tile Map Tools

The Tile Map system (also in the **FX + Tile Tools** panel) allows you to paint large tile-based backgrounds, platforms, or levels directly in the scene.

Workflow

1. **Prepare Tiles**
    - Import a tile sheet image in the left **Assets** panel.
    - Select the image in the **Images** list.
    - In the Tile Map Editor:
        - Set **Tile W** and **Tile H** (e.g., 32×32).
        - Click **Slice Selected Image Into Tiles**. This creates individual tile assets from the grid.
2. **Create a TileMap Object**
    - Click **+ TileMap Object**.
    - This adds a new object with a **TileMap** component (and a **TilemapCollider** for physics).
3. **Painting Tiles**
    - Select a tile from the **Tiles** palette (bottom of left panel).
    - In the Tile Map Editor, toggle **Paint** mode.
    - Click and drag on the scene canvas over the TileMap object to paint tiles.

    - Toggle **Erase** mode to remove tiles.
    - The **tileToolInfo** text at the bottom shows current status.

Important TileMap Component Settings (Inspector)

- **Tile W / Tile H**: Size of each tile.
- **Cols / Rows**: Grid dimensions of this map.
- **Layer**: Set to a low number (e.g., -2 or -3) so tiles draw behind sprites.
- **Cells**: Automatically managed while painting.

Collision

- The **TilemapCollider** component makes painted tiles solid for physics.
- You can disable solidity or make it a trigger if needed.

Tips & Best Practices

- Use a single TileMap object for your main level background/platforms.
- For multiple layers (background, midground, foreground), create several TileMap objects with different **Layer** values.
- Keep tile size consistent (e.g., 32×32) across your project.
- You can still move, rotate, and scale the entire TileMap object like any other object.
- For performance in large levels, avoid extremely large Cols/Rows values.

Common Workflow: Platformer Level

1. Slice a tileset image into tiles.
2. Create a TileMap.
3. Paint ground, platforms, and decoration.
4. Add BoxCollider or rely on TilemapCollider for solid platforms.
5. Place player and other objects on top.

Suggested Table for Components (Update Existing Table in Section 8)

Add these two rows to the Component table:

Component	Purpose	Key fields	Notes
ParticleEmitter	Emits saved particle effects	Effect ID, Play On Start, Loop, Rate	Great for fire, sparks, dust, explosions
TileMap	Grid-based level painting	Tile W/H, Cols, Rows, Layer	Paint tiles directly on canvas; includes collider

9. Play Mode

Play Mode runs a clone of the current scene. This lets you test physics, animation, keyboard input, and scripts without permanently changing your edit scene.

- Press Play to start runtime testing.
- Press Stop to leave Play Mode and restore the edit scene snapshot.
- The status text changes from Edit Mode to Play Mode.
- Keyboard input is captured during Play Mode. Click the canvas if keyboard controls do not respond.
- Runtime script errors appear as toast messages and in the browser console.

Important Play Mode behavior
Changes made by scripts during Play Mode are not saved back into the edit scene when Stop is pressed. This is useful for safe testing, but it also means runtime movement and spawned objects disappear after Stop.

10. Script Library

The Script Library stores JavaScript behavior scripts in a Unity-style wrapper. Scripts are registered with Engine.registerScript and then attached to scene objects using a Script component.

10.1. Built-in script names

- WASDMove: moves the object directly with WASD or arrow keys.
- PlayerController: uses a Rigidbody for left/right movement and jumping.
- Spin: rotates the object over time.
- BounceOnHit: flips horizontal velocity and bounces upward when a collision begins.

10.2. Script method pattern

A script can define properties, start, update, and onCollisionEnter. The Script component must use the exact script name.

```
Engine.registerScript("MyScriptName", {
  properties: { speed: 180 },

  start(gameObject, Engine) {
    // Runs once when the script instance is created in Play Mode.
  },

  update(gameObject, dt, Engine) {
    // Runs every frame in Play Mode.
    // dt is delta time in seconds.
  },

  onCollisionEnter(gameObject, other, Engine) {
    // Runs when this object first collides with another collider.
  }
});
```

10.3. Useful Engine API

API	Example	Purpose

Engine.input.key(key)	Engine.input.key("a")	True while a key is held down.
Engine.input.keyPressed(key)	Engine.input.keyPressed("Space")	True only on the frame the key is pressed.
Engine.getComponent(obj, type)	Engine.getComponent(gameObject, "Rigidbody")	Find the first component of a type on an object.
Engine.find(name)	Engine.find("Player")	Find the first object with a matching name.
Engine.findAll(name)	Engine.findAll("Enemy")	Find all objects with a matching name.

10.4. Example: simple left-right mover

```
Engine.registerScript("SimpleSideMove", {
  properties: { speed: 180 },
  update(gameObject, dt, Engine) {
    if (Engine.input.key("a") || Engine.input.key("ArrowLeft")) {
      gameObject.transform.x -= this.speed * dt;
    }
    if (Engine.input.key("d") || Engine.input.key("ArrowRight")) {
      gameObject.transform.x += this.speed * dt;
    }
  }
});
```

10.5. Attaching a script to an object

49. Select the object in the scene or Hierarchy.
50. In the Inspector, add a Script component.
51. Set Script Name to the registered script name, such as WASDMove or SimpleSideMove.
52. Press Play and test the behavior.

11. Saving, Loading, and Exporting

11.1. Save Project

Save Project stores the current assets, scene, and script library in browser localStorage. This is quick and useful for same-browser work, but it is not a portable project file yet.

11.2. Load Project

Load Project restores the saved localStorage project for the same browser and file origin. If the engine file is moved or opened through a different path or server origin, the browser may treat it as a different save location.

11.3. Export Game HTML

Export Game HTML creates a standalone playable HTML file named kstar-lite-exported-game.html. The exported file includes the runtime, embedded assets, scene data, and the current script library. It does not include the editor panels.

Export testing habit

After exporting, open the exported HTML file and test controls, collision, animation, and scripts. The exported runtime should be treated as the player-facing build.

12. Common Workflows

12.1. Create a static sprite prop

53. Import an image or sprite sheet.
54. Slice the needed frame sprite.
55. Drag the frame sprite into the scene.
56. Select the object and rename it, such as Tree, Rock, Door, or Coin.
57. Adjust X, Y, Scale, Layer, Flip X, or Flip Y in the Inspector.

12.2. Create an animated player

58. Slice the player walking frames.
59. Build and save an animation named Player_Walk.
60. Drag Player_Walk into the scene.
61. Select the player object and add Rigidbody and BoxCollider.
62. Add a Script component and set Script Name to PlayerController or WASDMove.
63. Press Play and test movement.

12.3. Create a platform or wall

64. Click + Empty Object or drag in a sprite.
65. Rename it Ground, Platform, Wall, or Block.
66. Add a BoxCollider and set Width and Height to match the visible shape.
67. Do not add a Rigidbody if the object should stay fixed.
68. If it has a SpriteRenderer, set Layer lower than the player so it draws behind when needed.

12.4. Create a spinning collectible

69. Create a coin or item animation and drag it into the scene.
70. Add a CircleCollider and enable Trigger if you only want overlap detection.
71. Add a Script component with Spin for visual rotation, or create a custom pickup script later.
72. Use Layer to draw the item in front of the background.

12.5. Create a simple physics object

73. Drag a sprite into the scene.
74. Add Rigidbody.
75. Add BoxCollider or CircleCollider.
76. Set Gravity Scale above 0 for falling behavior.
77. Set Drag to slow the object over time.
78. Press Play and watch how it interacts with the Ground collider.

13. Troubleshooting

Problem	Likely cause	Fix
Keyboard controls do not work.	Canvas is not focused or Play Mode is not running.	Click the scene canvas, press Play, then test again.

Animation does not play.	Animator has no animation selected, Playing is off, or animation has no frames.	Select the object, choose an animation in Animator, enable Playing, verify frames.
Sprite is invisible.	No sprite selected, opacity is 0, object inactive, or layer/position makes it hard to see.	Check Active, SpriteRenderer Sprite, Opacity, X/Y position, and scale.
Auto slicing combines frames.	Sprites are touching or Merge Gap is too high.	Lower Merge Gap, add more spacing to the source image, or use Custom Mouse Cuts.
Object falls through ground.	Missing collider, missing Rigidbody, trigger enabled, or collider sizes do not overlap.	Add colliders, check Trigger setting, adjust collider sizes and offsets.
Script does not run.	Script Name does not match registered name or script has a syntax error.	Check exact capitalization in Engine.registerScript and browser console errors.
Saved project disappeared.	localStorage is tied to browser and origin/path.	Open the engine from the same location and browser; export HTML for playable builds.
Exported game behaves differently.	Editor-only assumptions or script/runtime mismatch.	Test the exported file after each export and keep scripts simple/compatible.

14. Glossary

Term	Meaning
Asset	An imported image, sliced frame sprite, or saved animation.
Frame Sprite	A single trimmed image created by slicing an imported image.
Animation	A named sequence of frame sprites with FPS and loop settings.
Scene Object	An item placed in the scene with transform data and components.
Component	A modular piece of behavior or data attached to an object.
Rigidbody	Component that allows velocity, acceleration, gravity, drag, and physics movement.
Collider	Shape used for collision detection and resolution.
Trigger	Collider mode for overlap detection without solid pushing.
Script Library	The editor text area where registered gameplay scripts are written.
Play Mode	Runtime test mode that runs a clone of the scene.
Exported Runtime	The standalone HTML game produced by Export Game HTML.

15. Current Limitations and Next Manual Sections

This starter manual reflects the current MVP-style foundation. These notes are useful for future development and for setting expectations while the engine grows.

- Project saving currently uses one browser localStorage project rather than a downloadable project file system.
- The editor canvas and exported game use a fixed 960×540 canvas in the current build.
- PolygonCollider displays polygon points, but current physics resolution uses a bounding box style approach.
- The exported game contains the runtime and current project, not the editor interface.
- There is no formal prefab system yet; duplicate objects and reusable scripts can fill this role for now.
- There is no full project asset folder yet; imported images are embedded in browser/project data as data URLs.
- Future manual sections should cover a full platformer tutorial, a top-down RPG tutorial, projectile scripts, animation controller patterns, save-file export/import, prefab workflow, and mobile/Wix embedding.

Next recommended manual expansion

The best next chapter would be a complete “Build Your First Game” tutorial using the current tools: player movement, ground collision, enemy object, collectible, score text placeholder, and exported HTML test.