AI Game Builder — Getting Started Guide

From zero to a playable game in 10 minutes.

Prerequisites

Before you begin, install these three tools:

Tool	Version	Install
Godot Engine	4.3+	godotengine.org/download — download the **Standard** version
Node.js	20+	nodejs.org — use the LTS version
Claude Code	Latest	`npm install -g @anthropic-ai/claude-code`

Verify everything is installed:

```
godot --version      # Should show 4.3.x or higher
node --version       # Should show v20.x or higher
claude --version     # Should show the Claude Code version
```

Step 1: Get the AI Game Builder

Option A: Clone from GitHub

```
git clone https://github.com/HubDev-AI/godot-ai-builder.git ~/ai-game-builder
```

Option B: Use a local copy

If you already have the folder (e.g., you built it yourself or received it as a zip):

```
# Just note where it lives, e.g.:
ls ~/ai-game-builder     # or wherever you put it
```

The folder is a **Claude Code plugin** — it contains skills, hooks, MCP config, and more:

```
ai-game-builder/
├── .claude-plugin/    # Plugin manifest (makes this a Claude Code plugin)
├── skills/            # 14 game generation skills
├── hooks/             # Stop hook (build guard)
├── .mcp.json          # MCP server configuration
├── mcp-server/        # MCP bridge (Claude ↔ Godot)
├── godot-plugin/      # Godot editor plugin
├── knowledge/         # GDScript reference, patterns, etc.
├── setup.sh           # Godot editor plugin installer
└── README.md
```

Step 2: Create a New Godot Project

Open the Godot editor. You'll see the **Project Manager**.

1. Click **"New Project"**
2. Set the **Project Name** (e.g., `my-first-game`)
3. Choose a **Project Path** (e.g., `~/games/my-first-game`)
4. Select **Renderer**: choose **Forward+** (default) or **Compatibility** for 2D games
5. Click **"Create & Edit"**

Godot opens the editor with an empty project. This is fine — Claude will create all the files.

> **Tip**: You can close the Godot editor after creating the project. The setup script only needs the project folder to exist with a `project.godot` file. You'll reopen Godot in Step 4.

Step 3: Install the Godot Editor Plugin

The setup script installs the HTTP bridge plugin into your Godot project:

```
cd ~/ai-game-builder
./setup.sh ~/games/my-first-game
```

Replace `~/games/my-first-game` with the actual path to your Godot project.

```
================================================
  AI Game Builder — Godot Editor Setup
================================================

Godot project: /Users/you/games/my-first-game

Installing MCP server dependencies...
✓ MCP server ready
Installing Godot editor plugin...
✓ Plugin installed to /Users/you/games/my-first-game/addons/ai_game_builder/
Installing knowledge base...
✓ Knowledge base installed

================================================
  Godot editor setup complete!
================================================
```

What just happened: - The Godot HTTP bridge plugin was copied into your project's `addons/` folder - MCP server dependencies were installed (Node.js) - A knowledge base was added with GDScript references and game patterns

---

Step 4: Enable the Plugin in Godot

1. Open your project in the **Godot editor** (double-click it in the Project Manager)
2. Go to **Project → Project Settings**

3. Click the **Plugins** tab at the top
4. Find **"AI Game Builder"** in the list
5. Check the **Enable** checkbox

You should see a new **"AI Game Builder"** panel appear in the bottom dock. It shows: - A green dot when the HTTP bridge is active (port 6100) - Real-time progress log: file writes, phase transitions, error fixes, and all MCP operations - When Claude spawns sub-agents for parallel work, each agent's progress shows here too

> **Important**: Keep Godot open. Claude Code communicates with it in real-time through the plugin.

Step 5: Open Claude Code with the Plugin

This is the critical step. You MUST load the plugin when starting Claude Code, otherwise the MCP tools and skills won't be available.

Open a new terminal and navigate to your Godot project:

```
cd ~/games/my-first-game
claude --plugin-dir ~/ai-game-builder
```

> **`--plugin-dir` is required for local usage.** It tells Claude Code to load the AI Game Builder plugin from your local folder. Without it, Claude won't have the MCP tools to talk to Godot.
>
> **Alternative**: If you installed via marketplace (`/plugin install godot-ai-builder`), the plugin loads automatically and you can just run `claude`.

How to verify the plugin is loaded: After Claude starts, you can check: - Skills show up when you type `/godot-ai-builder:` (tab completion) - Claude mentions MCP tools like `godot_get_project_state` when you ask it to build something - If Claude uses `curl http://localhost:6100/...` instead of MCP tools, the plugin is NOT loaded — restart with `--plugin-dir`

Claude Code starts with full awareness of: - The Godot project structure - All 14 game generation skills (namespaced as `/godot-ai-builder:godot-*`) - MCP tools to control the Godot editor (reload, run, errors, assets) - The Stop hook that keeps it focused during builds

---

Step 6: Tell Claude to Build Your Game

You have three ways to start:

Option A: Type a short prompt

Give Claude a brief idea and it will ask you to choose scope:

```
Make a shooter game
```

```
Create a platformer
```

Claude will respond:

> I can build this two ways:
>
> **A) Full game** — I'll design everything: multiple enemy types, UI screens, progressive difficulty, visual polish. I'll write a PRD for your approval first.
>
> **B) Simple game** — Minimal and focused. Basic player, basic gameplay, just enough to be playable. Fast to build, easy to extend later.
>
> Which would you prefer?

Option B: Type a detailed prompt
If you include enough detail (3+ features, enemy types, UI screens, etc.), Claude skips the question and goes straight to building:

```
Create a top-down shooter where I move with WASD, aim with mouse,
and shoot enemies that chase me. Include a main menu, HUD with health
and score, game over screen, 2 enemy types, and visual polish.
```

```
Create a complete tower defense game with 3 tower types, 10 waves,
gold economy, lives system, full UI, and polish.
```

Option C: Point to a folder with game design documents
If you already have a game design document (GDD), feature list, art references, level designs, or any planning docs — put them all in a folder and tell Claude:

```
Build the game from the docs in ~/my-game-design/
```

```
Take the folder ~/desktop/rpg-project/ and start working on this game.
```

```
Read all the documents in ./game-plan/ and build this game in Godot.
```

Claude will: 1. Scan every file in the folder (`.md`, `.txt`, `.pdf`, `.json`, images, etc.) 2. Read each document and report what it found 3. Generate a PRD from your documents 4. Ask for your approval 5. Build the game phase by phase

This is powerful for complex games — prepare your design offline, then let Claude build it.

What Happens Next

> **Note about the MCP server**: You don't need to start it manually. Claude Code automatically launches the MCP server (Node.js process) when the plugin loads. The server bridges Claude Code to the Godot editor via HTTP on port 6100. Just make sure Godot is open with the plugin enabled.

If you chose "Simple game"
Claude builds directly — no PRD, no phases. It writes the minimum files needed (player, main scene, basic gameplay), runs the game, fixes any errors, and you're done. Fast and focused.

You can always extend later: “Add enemies”, “Add a menu”, “Make it look better”.

If you chose “Full game” (or gave a detailed prompt / docs folder)
Claude follows the **Game Director** protocol — a 6-phase build process:

Phase 0: PRD (Product Requirements Document)
Claude writes a detailed game design doc and asks for your approval before writing any code. It also asks about your preferred **visual tier**: procedural (shaders + gradients + glow — default), custom art, AI-generated art, or prototype shapes.

Phase 1: Foundation
Creates the project structure, player movement, camera, and background. Runs the game to verify it works.

Phase 2: Player Abilities
Adds shooting, jumping, or whatever the player’s actions are. Tests everything.

Phase 3: Enemies & Challenges
Creates enemy types, spawn systems, collision, health, scoring, and difficulty ramping.

Phase 4: UI & Game Flow
Builds the main menu, HUD, game over screen, pause menu, and screen transitions. Wires up the full game loop: menu ⟶ play ⟶ die ⟶ retry.

Phase 5: Polish
Adds game “juice” — shader-based effects (hit flash, dissolve death, glow outlines), screen shake, death particles, spawn animations, floating score numbers, multi-layer backgrounds (gradient shader + grid + ambient particles + vignette), styled UI buttons, and smooth scene transitions.

Phase 6: Final QA
Runs the game multiple times, checks for errors, edge cases, and memory leaks. Reports the final status.

The Stop hook keeps Claude locked in until all 6 phases are complete. You don’t need to worry about it quitting halfway through.

During the Build

While Claude is working, you’ll see activity in two places:

In the terminal (Claude Code): - Claude reports progress after each phase - Shows quality gate results (e.g., “Phase 3 complete. 6/6 gates passed.”) - Asks for your input if needed (e.g., PRD approval)

In the Godot editor (AI Game Builder dock): - Shows real-time progress messages: file writes, phase transitions, error fixes - Shows bridge activity (reload, run, error check commands) -

When Claude spawns sub-agents, each agent logs its progress to the dock too - The game runs automatically in the editor when Claude tests it - You can see the game come to life in real-time

You can interact during the build: - Approve or modify the PRD when Claude presents it - Ask for changes ("make the enemies faster", "change the color palette") - Claude will adjust and continue

---

After the Build

When all 6 phases complete: 1. Claude reports what was built (file list, features, quality gates) 2. The Stop hook disengages (Claude can finish normally) 3. Your game is ready to play in the Godot editor 4. Press **F5** in Godot (or the play button) to run it anytime

Your project folder now contains:

```
my-first-game/
├── project.godot        # Game settings + input mappings
├── scripts/             # All game logic
│   ├── main.gd          # Game loop, spawning, score
│   ├── player.gd        # Player controller
│   ├── player_visual.gd  # Layered procedural player art
│   ├── enemy.gd         # Enemy AI
│   ├── bullet.gd        # Projectile logic
│   ├── enemies/         # Enemy types + visual scripts
│   ├── effects/         # Grid background, particles
│   └── ui/              # HUD, menus, game over
├── shaders/             # GLSL shaders (glow, dissolve, hit flash, gradient)
├── scenes/              # Scene files
├── assets/              # Generated sprites (SVG with gradients/shadows)
├── docs/PRD.md          # The game design document
├── addons/              # Godot editor bridge plugin
└── knowledge/           # Reference docs
```

Troubleshooting

Claude uses `curl` instead of MCP tools (MOST COMMON ISSUE)

This means the plugin wasn't loaded. Claude is bypassing the MCP bridge and hitting port 6100 directly. - **Fix**: Restart Claude with the plugin flag: `claude --plugin-dir ~/ai-game-builder` - **Verify**: Ask Claude "what MCP tools do you have?" — it should list `godot_get_project_state`, `godot_run_scene`, etc.

Claude writes files silently (no progress output)

The skills instruct Claude to report progress. If it's not doing so: - Tell Claude: "Report every file you write and every action you take. I want to see progress." - Or invoke the Director skill explicitly: `/godot-ai-builder:godot-director`

“Godot bridge not connected”

- Make sure the Godot editor is open with your project
- Check that the AI Game Builder plugin is enabled (Project → Project Settings → Plugins)
- The plugin runs an HTTP server on port 6100 — make sure nothing else uses that port

Game breaks after project.godot edit

- Claude may accidentally strip sections from project.godot (like `[autoload]`)
- **Fix**: Tell Claude “Check project.godot — make sure the [autoload] section is intact”
- The skills now instruct Claude to ALWAYS read project.godot before editing it

“No project.godot found”

- Run setup.sh with the path to a valid Godot project (the folder containing `project.godot`)
- If you haven’t created a project yet, do Step 2 first

Claude stops unexpectedly

- If Claude hits a rate limit, wait a moment and type “continue” — it will resume
- If the build lock is stuck, manually remove it: `rm .claude/.build_in_progress`
- Re-run Claude with `claude --plugin-dir ~/ai-game-builder` and tell it to continue

Game has errors after build

- Open Claude Code and say: “Fix the errors in my game”
- Claude will call `godot_get_errors`, read the problematic files, and fix them

Want to start over

- Delete the generated files: `rm -rf scripts/ scenes/ assets/ docs/`
- Tell Claude: “Create a new game from scratch”

Quick Reference

Action	How
Quick simple game	“Make a shooter” → choose “Simple”
Full polished game	“Make a shooter” → choose “Full”, or give a detailed prompt
Build from docs	“Build the game from the docs in ~/folder/”
Add a feature	“Add enemies that shoot back”
Fix errors	“Fix the errors” or “Check for errors and fix them”
Polish the game	“Add screen shake, particles, and juice”
Change something	“Make the player faster” / “Change the background to dark blue”
Cancel mid-build	`rm .claude/.build_in_progress` then Claude can stop
Run the game	Press F5 in Godot, or Claude runs it automatically
See game errors	Claude checks automatically, or say “Get errors”

Built with AI Game Builder — Claude Code + Godot 4