

THEAIGAMEDEV.COM

The AI Game Dev Prompt Pack

121 copy-and-paste prompts for solo and indie developers, grouped by the job you are doing.

Concept and core loop · Player feel · Level design · Godot, Unity, Unreal and Roblox · Blender · ComfyUI · Sprites · Textures · NPCs and quests · Audio · XR

Free to keep and share. Please do not resell.

© 2026 theaigamedev.com

Contents

1. Concept and core loop	8 prompts
2. Player feel, juice and onboarding	10 prompts
3. Level and world design	10 prompts
4. Code and systems: Godot	8 prompts
5. Code and systems: Unity	7 prompts
6. Code and systems: Unreal	5 prompts
7. Sprites and 2D art	12 prompts
8. Textures and materials	10 prompts
9. 3D models and sculpts	10 prompts
10. NPC dialogue, quests and behaviour	10 prompts
11. Audio and music	6 prompts
12. XR design and comfort	4 prompts
13. Code and systems: Roblox (Studio and Luau)	7 prompts
14. 3D pipeline: Blender	7 prompts
15. Image pipelines: ComfyUI	7 prompts

Replace the [square brackets] with your project's details. Ask for one thing per prompt.

121 copy-and-paste prompts for solo and indie developers building with AI.

Written by the team at theaigamedev.com. Free to keep and share. Please do not resell.

How to use it: every prompt is a template. Replace the `[square brackets]` with your project's details, keep the structure, and paste it into your AI chat, image, audio, or code tool. The prompts are grouped by job, with dedicated sections for six tools: Godot, Unity, Unreal, Roblox (Studio and Luau), Blender and ComfyUI.

Two rules that make the output better:

- Give the model your real constraints (engine and version, target platform, art style, budget in minutes or draw calls).
- Ask for one thing per prompt. Big asks produce vague answers.

1. Concept and core loop (8)
2. Player feel, juice and onboarding (10)
3. Level and world design (10)
4. Code and systems: Godot (8)
5. Code and systems: Unity (7)
6. Code and systems: Unreal (5)
7. Sprites and 2D art (12)
8. Textures and materials (10)
9. 3D models and sculpts (10)
10. NPC dialogue, quests and behaviour (10)
11. Audio and music (6)
12. XR design and comfort (4)
13. Code and systems: Roblox (Studio and Luau) (7)
14. 3D pipeline: Blender (7)
15. Image pipelines: ComfyUI (7)

1. Concept and core loop

1. Core loop one-pager.

Turn a rough idea into the loop players repeat.

```
Act as a game designer. My game is [one sentence]. Write a core loop as a numbered cycle of 5 player actions, then state for each what the player wants, what they do, and what changes as a result. End with the single emotion the loop should produce. Keep it under 200 words.
```

2. Mechanic stress test.

Find where a mechanic breaks before you build it.

Here is my core mechanic: [describe it]. List 5 ways players will break, exploit, or ignore it. For each, propose one fix and rate the fix's cost as low, medium, or high. Do not praise the idea.

3. Scope cut for a solo dev.

Turn a big idea into something shippable.

I am one developer with [X] hours a week and [Y] weeks. My concept is [describe it]. Cut it to a first playable slice: name the 3 systems to build, the 3 to fake, and the 3 to remove. For each fake, say how to fake it convincingly.

4. Genre promise check.

Make sure your game delivers what the genre promises.

My game is a [genre] for [platform]. List the 6 things players of this genre expect on day one, then score my plan [paste plan] against each as met, partial, or missing. For anything partial or missing, suggest the smallest change that fixes it.

5. Progression curve.

Lay out unlocks and difficulty.

Design a progression curve for [game] lasting [N] hours. Give a table with hour, new unlock, new enemy or obstacle, and target difficulty (1-10). The curve should teach a new idea every 20-30 minutes and never spike more than 2 points at once.

6. One-sheet PRD.

A single page you can build and pitch from.

Write a one-page PRD for [game]. Sections: hook, target player, core loop, 3 pillars, what makes it different from [nearest comparison], first-milestone scope, and the one riskiest assumption. Be specific and cut anything generic.

7. Pillar test.

Kill features that fight the game's identity.

My 3 pillars are [pillar 1], [pillar 2], [pillar 3]. Here is my feature list: [list]. For each feature, say which pillar it serves or mark it as off-pillar. Recommend cutting off-pillar features.

8. Vertical slice plan.

Decide what the demo must prove.

I want a 10-minute vertical slice of [game] to show [audience]. List what the slice must prove, what it can safely cut, and a 5-step build order. Assume limited art and no voice acting.

2. Player feel, juice and onboarding

9. Movement tuning.

Diagnose floaty or stiff controls.

My character moves with [acceleration], [max speed], [friction], [jump height]. Playtesters say it feels [floaty / stiff / slippery]. Give 5 specific value changes to try, each with the number to move and the direction, and explain what each change should fix.

10. Game feel checklist.

Add juice without clutter.

For a [genre] game, list 10 feedback moments (impacts, pickups, deaths, jumps) and for each suggest a screen, sound, particle, or timing response. Order them by how much they improve feel per hour of work.

11. Input response.

Tighten the link between press and result.

My jump feels unresponsive. Walk me through measuring input latency in [engine], the 4 most common causes of lag between press and action, and a fix for each. Keep it practical.

12. Camera feel.

Make the camera help rather than fight.

Design camera behaviour for [genre]. Cover follow distance, smoothing, look-ahead, screen shake rules, and what the camera does at high speed and at rest. Give starting values for [engine] and say which two are most worth tuning.

13. Juice for a hit.

Make combat land.

My melee hit feels flat. Give 8 layered responses for a single hit: anticipation, impact frame, hit-stop duration, camera, particles, sound layers, controller rumble, and the target's reaction. Rank by effort.

14. Onboarding without a tutorial.

Teach by doing.

My game teaches [mechanic]. Design a 3-minute opening that teaches it without a text tutorial. Describe the space, the props, the first safe action, and the first real test. No walls of text.

15. First 60 seconds.

Hook a new player fast.

Write the first 60 seconds of my [genre] game as a beat sheet. Second by second, say what the player sees, does, and learns. The player should make one meaningful choice before the 60 seconds end.

16. Failure feedback.

Make losing fair and clear.

When the player dies in [game], design the feedback: what is shown, for how long, what they keep, and how fast they can retry. Suggest a way to turn a death into a lesson rather than a punishment.

17. Accessibility pass.

Widen who can play.

Review [game] for accessibility. List the settings to add for vision, hearing, motor, and cognitive needs, with an easy implementation note for each. Flag anything that would change the game's design rather than just add options.

18. HUD declutter.

Show only what matters.

My HUD elements are [list]. For each, say whether it should be always on, contextual, or removed, and why. Suggest a diegetic alternative for the two most intrusive.

3. Level and world design

19. Guide without signs.

Teach a space through its shape.

Design a level that teaches [mechanic] using architecture only, no text prompts. Describe the entrance sightline, the first affordance, the safe practice area, and the first test. Explain how the geometry guides the player.

20. Greybox beat chart.

Plan pacing before art.

Write a beat chart for a [N]-minute level. Give a table: beat, purpose, player action, tension (1-10), and reward. Alternate tension and rest, and place the difficulty peak at about 70 percent through.

21. Metroidvania gating.

Lock and key placement.

My abilities are [list]. Design a map where each ability opens the next area. Give the ability order, the gate each one clears, and one optional shortcut that rewards players who backtrack.

22. Blockout from a sketch.

Turn a rough idea into a layout.

I have this level idea: [describe]. Produce a top-down blockout as a text grid, mark the entrance, exit, key encounter, secret, and sightline landmarks. Explain the player's route and one place they will get lost, and how you fixed it.

23. Secret placement.

Reward exploration honestly.

Place 5 secrets in [level]. For each, give the hiding method (visual, spatial, mechanic), the hint that it exists, and the reward. Make sure 2 are findable by accident and 3 require intent.

24. Difficulty pacing across a world.

Keep a whole game fair.

Lay out difficulty across my [N] levels for [game]. Give each level a target challenge (1-10), the new mechanic it introduces, and the rest beat that follows it. No two spikes back to back.

25. Open zone flow.

Make a hub readable.

Design a hub area that connects to [N] zones. Describe the landmarks that help players orient, how each exit is signposted without signs, and how the hub changes as the player progresses.

26. Encounter design.

Make one fight interesting.

Design a single encounter for a [genre] game against [enemy type]. Specify the arena, the enemy's 3 behaviours, the tell for each, the player's counters, and the moment the fight should feel hardest.

27. Readable cover.

Make combat positions clear at a glance.

My combat is [describe]. Suggest how to make player and enemy positions readable at a glance: silhouettes, ground markers, contrast, and camera angle. Give 6 concrete rules.

28. World lore through place.

Tell the story with the environment.

My world's history is [describe]. Show it through 6 environmental details a player can find, with no dialogue and no lore dumps. For each, say where it sits and what it implies.

4. Code and systems: Godot

29. Godot state machine.

Organise character states cleanly.

```
Write a finite state machine for a Godot 4 [2D/3D] character in GDScript. Include an idle, run, jump, and attack state, a clean transition method, and no code duplication. Comment the key lines. Keep it under 120 lines.
```

30. Godot object pooling.

Stop spawning stutter.

```
Show a bullet pooling system for Godot 4 in GDScript. Preallocate [N] instances, reuse them, and return them to the pool. Explain when to reset state and how to handle overflow.
```

31. Godot save system.

Persist progress simply.

```
Design a save and load system for a Godot 4 game. Save player stats, level index, and settings to user:// as JSON. Handle missing files, a version field for future changes, and a corrupted-file fallback.
```

32. Godot signals decoupling.

Untangle direct references.

```
Refactor this Godot script so systems talk through signals instead of direct node references: [paste code]. Keep behaviour identical and explain each signal you introduce.
```

33. Godot performance pass.

Find the frame spikes.

```
My Godot 4 game drops frames when [situation]. List the 6 most likely causes, how to confirm each with the profiler, and the fix. Start with the cheapest to test.
```

34. Godot procedural map.

Generate a fair layout.

```
Write Godot 4 GDScript that generates a [room / cave / dungeon] using [algorithm]. Guarantee a path from entrance to exit, add a seed for reproducibility, and expose parameters for size and density.
```

35. Godot UI state.

Keep menus sane.

Design a menu system for Godot 4 using scenes and an autoload singleton. Cover pause, settings, and a pause stack so opening one menu from another works. Give the node structure and the key scripts.

36. Godot shader starter.

A specific visual effect.

Write a Godot 4 canvas shader that creates a [water / dissolve / outline / pixelate] effect. Explain each uniform, give sensible defaults, and note the performance cost.

5. Code and systems: Unity

37. Unity movement.

A solid controller base.

Write a Unity 2022+ character controller in C# for [genre]. Include acceleration, friction, coyote time, jump buffering, and ground checks with a raycast. Explain the tuning variables and keep it under 150 lines.

38. Unity ScriptableObject data.

Separate data from code.

Design a ScriptableObject system for [items / enemies / abilities] in Unity. Show the data class, one concrete asset, and how gameplay reads it. Explain why this beats a big switch statement.

39. Unity object pooling.

Reuse expensive objects.

Implement object pooling in Unity for [bullets / enemies / particles]. Use a manager, a factory, and an interface for resetting pooled objects. Explain when pooling is not worth it.

40. Unity async loading.

Load without a hitch.

Show how to load a scene asynchronously in Unity with a loading bar. Handle progress, a minimum display time, and errors. Include the coroutine and the UI hookup.

41. Unity profiler triage.

Read the numbers.

My Unity game stutters on [platform]. Give a step-by-step profiling routine: which modules to check first, what healthy numbers look like for CPU and GPU, and the 5 most common culprits in mobile and PC builds.

42. Unity new Input System.

Map across devices.

Set up Unity's new Input System for a game that plays with keyboard, gamepad, and touch. Show the action map, how to read values, and how to swap control schemes at runtime.

43. Unity addressables basics.

Ship assets without bloat.

Explain Unity Addressables for a solo dev shipping on [platform]. Cover grouping, a remote catalog, and one concrete workflow for moving a large asset out of the base build. Keep it beginner-friendly.

6. Code and systems: Unreal

44. Unreal Blueprint to C++.

Move a hot path into code.

This Blueprint ticks every frame and hurts performance: [describe it]. Show how to move it to a C++ actor component, what to keep in Blueprint, and how to expose the key variables back to the designer.

45. Unreal AI behaviour tree.

A readable enemy.

Design an Unreal behaviour tree for an enemy that [behaviour]. Show the tree structure, the blackboard keys, the tasks and decorators, and the service that updates target data. Explain the logic in plain terms.

46. Unreal Niagara effect.

A specific VFX.

Describe how to build a Niagara effect for [impact / trail / weather] in Unreal 5. List the emitters, the modules to set, and 4 values to expose for reuse. Note the performance setting that matters most.

47. Unreal Nanite and Lumen decisions.

Know when to use them.

I am building [project] for [platform]. For each of Nanite, Lumen, and virtual shadow maps, say whether to enable it, the target hardware that justifies it, and the fallback for weaker machines.

48. Unreal save game.

Persist without pain.

Write an Unreal SaveGame system in C++. Save player progress, world state, and settings, include a version number, and show how to handle a save from an older version.

7. Sprites and 2D art

49. Character sprite sheet.

A consistent walk cycle.

Create a 2D sprite sheet, [style], of a [character] with a 6-frame walk cycle facing [direction]. Add a 4-frame idle and a single attack pose. Transparent background, even frame spacing, consistent lighting from [direction]. No text.

50. Sprite style lock.

Match an existing look.

Generate a 2D [character/enemy] in this exact style: [describe style and palette]. Same line weight, same shading, same proportions as the reference. Flat side-on view, transparent background, no text.

51. Icon set.

A matching loadout of icons.

Create a set of 6 game icons for [fire, water, heal, shield, coin, key] in [style]. Same outline weight, same light direction, same padding, transparent background, readable at 32 pixels. Arrange in a single row.

52. Tileset.

A seamless 2D tileset.

Create a 2D tileset, top-down, of [theme] for a tile size of [32/48] pixels. Include grass, path, water edge, rock, and a wall corner that tiles seamlessly. Consistent palette, no text, flat clean edges.

53. Parallax background.

Depth for a 2D level.

Create a 3-layer parallax background for [scene]. Far layer: sky and distant shapes. Mid layer: [landmarks]. Near layer: [foreground detail]. Consistent palette, no characters, each layer on its own transparent strip.

54. Boss silhouette.

Readable and menacing.

Design a [boss] creature as a bold silhouette, [style]. Strong readable shape, distinct weak-point area, side-on view, transparent background. Give me one front pose and one attack pose. No text.

55. UI kit.

A matching HUD.

Create a [style] game UI kit: health bar, mana bar, 4-slot hotbar, and a minimap frame. Same corner radius, same border weight, same palette. Transparent background, arranged on a grid. No text or numbers.

56. Portrait set.

Dialogue-ready faces.

Create 4 character portrait busts for [characters] in [style]. Same framing, same lighting, consistent proportions. Neutral expression, transparent background, one image per character. No text.

57. Pickups and props.

Small readable items.

Create a set of 8 small game pickups for [coins, gems, keys, potions]. [Style], consistent light direction, transparent background, readable at 24 pixels, arranged in a tidy grid. No text.

58. Sprite animation frames only.

No blending between poses.

Create a 4-frame [jump / spin / cast] animation of a [character], [style]. Keep the pose distinct in every frame, no motion blur, no onion skinning, transparent background, even spacing in one row. No text.

59. Item rarity variants.

One item, four tiers.

Take a single [sword / potion / crystal] and create 4 rarity variants: common, rare, epic, legendary. Keep the base shape identical, change only palette and one glow effect. Transparent background, same size and angle. No text.

60. Reference sheet for 3D.

A turnaround to model from.

Create a character turnaround sheet for a [character] in [style]: front, side, and back views, plus one 3/4 pose. Consistent proportions and palette, plain background, T-pose. No text or labels.

8. Textures and materials

61. PBR material set.

A full texture stack.

Create a seamless PBR texture set for [material], 2048x2048: albedo, normal, roughness, and ambient occlusion. Matches a [style] game, no baked lighting, no text. Keep the pattern tileable on all edges.

62. Stylised ground.

Gamified terrain, not a photo.

Create a seamless, stylised [grass / sand / stone] ground texture for a [style] game. Hand-painted feel, readable at distance, no harsh photographic detail, tileable on all four edges, no text.

63. Metal and rust.

Age a surface honestly.

Create a PBR texture set for [painted metal / iron / copper] with wear. Show base material, edge rust, and scratches as separate layers if possible. Seamless, 2048x2048, no text.

64. Fabric and cloth.

Weave and folds.

Create a seamless fabric texture for [burlap / linen / leather]. Include a subtle normal map with weave depth. 2048x2048, tileable, no directional lighting baked in, no text.

65. Trim sheet.

One sheet, many props.

Create a trim sheet for a [sci-fi / medieval] kit: 8 vertical strips of distinct materials, each tileable top to bottom, on one 2048x2048 sheet. Flat lighting, clear separation, no text.

66. Decals.

Damage and detail.

Create 6 decals for [sci-fi / fantasy] environments: scorch mark, crack, bullet hole, moss patch, warning stripe, floor grate. Transparent background, top-down, consistent scale, no text.

67. Skybox.

A full surround.

Create an equirectangular 360 skybox for [time of day] in [style]. Seamless left to right, horizon low, no sun flare in the centre, 4096x2048, no text.

68. Water material.

Shallow, deep and foam.

Describe a stylised water material for [engine] with shallow, deep, and foam zones. Give the blend logic, the normal detail, and the two parameters for wave speed and colour. Keep it performant on [platform].

69. Emissive panel.

Glowing UI surfaces.

Create a sci-fi panel texture with an emissive mask for [a console / a door / a screen]. Base colour plus a separate emissive channel, 2048x2048, seamless if used as a wall, no readable text.

70. Gradient and grunge pack.

Overlays for many uses.

Create a pack of 8 overlays: 4 gradient ramps and 4 grunge masks. Grayscale, 2048x2048, transparent where useful, tileable. For quick use in [engine] shaders as multiply or overlay layers. No text.

9. 3D models and sculpts

71. Prop from a photo prompt.

A clean game-ready asset.

Create a [style] 3D model of [prop]. Low-poly, game-ready, under [N] triangles, clean quads, UV unwrapped, no textures baked. Side and 3/4 views. No text.

72. Modular building kit.

Snap-together pieces.

Design a modular 3D kit for [theme]: 6 pieces that snap on a [grid size] grid, sharing one texture atlas and one pivot convention. Explain the sizing rules so every piece lines up. No text.

73. Hero weapon.

One well-formed asset.

Create a [style] hero weapon model: [sword / rifle / wand]. Game-ready topology, under [N] triangles, separated grip and blade for animation, clean UVs, plain material. Show it at an angle. No text.

74. Creature base mesh.

A sculpt to build on.

Create a base mesh for a [creature type]. Clean topology that supports animation, symmetric, mouth and eye areas with enough loops to deform. Neutral pose, wireframe view, no text.

75. Hard-surface detail pass.

Sci-fi panel lines.

Add hard-surface detail to a [spaceship / crate / machine]: panel lines, chamfers, vents, and bolts. Keep it game-ready under [N] triangles, clean UVs, readable silhouette. No text.

76. Environment rock set.

Varied and tileable.

Create a set of 6 [style] rocks that share one material and fit a [size] grid. Range from small clutter to a large boulder. Game-ready topology, no text.

77. Vegetation set.

Foliage that reads.

Create a [style] vegetation set for [biome]: 3 trees, 2 bushes, and 3 ground plants. Low-poly, alpha-card friendly, shared palette, consistent scale. No text.

78. Modular character parts.

Swap-friendly.

Design a modular character for the [gender / body type] base: 3 heads, 3 torsos, 3 leg sets that swap cleanly. Same joint placement and proportions across all parts, one shared texture atlas. No text.

79. Terrain stamps.

Sculpt faster.

Create 6 grayscale height stamps for [mountains / craters / dunes / riverbeds]. 1024x1024, tileable where noted, for use in [engine] terrain tools. No text.

80. Retopology guide.

Clean up an AI mesh.

This AI-generated mesh has messy topology: [describe]. Give me a step-by-step retopology plan for a game-ready version, the target triangle budget for [use], and where I must keep edge loops.

10. NPC dialogue, quests and behaviour

81. Character voice card.

A consistent NPC.

Write a character card for [NPC]: name, role, wants, fears, and 5 speech habits. Then write 6 sample lines that show the voice without saying anything about their personality. Keep it to [era / setting].

82. Branching dialogue.

Choices with weight.

Write a branching dialogue between the player and [NPC] about [topic]. Give 3 choices at the first decision, short responses, and one branch that changes a relationship value. No dead ends.

83. Quest design.

A quest with a real decision.

Design a side quest for [game]: hook, 3 steps, one complication, and a choice with two defensible outcomes. Say what each outcome changes and how the player learns the quest exists.

84. Bark pack.

In-the-moment lines.

Write 20 short barks for [enemy / ally] in [situation]: idle, spotting the player, taking damage, dying, and victory. One line each, under 8 words, no repetition, fits a [tone] game.

85. Behaviour tree in words.

Plan before you code.

Describe a behaviour tree for an enemy that [behaviour]. List the nodes top to bottom, the blackboard keys, and the priority order. Then name 3 edge cases and how the tree should handle each.

86. Combat AI difficulty.

Fair escalation.

Design enemy AI at 3 difficulty levels for [enemy]. For each level, set reaction time, accuracy, aggression, and team behaviour. Keep higher difficulty fair: no aimbot, no reading inputs.

87. Faction opinion model.

Relationships that shift.

Design a faction system where player actions change 4 factions' opinions of them. Define the scale, 6 actions and their effects, and how opinion changes NPC dialogue and prices. Keep the maths simple.

88. Procedural quest text.

Fill templates without repetition.

Write 12 quest templates for [game], each with fillable slots for location, target, and reward. Vary the sentence structure so repeated quests do not feel copy-pasted. Keep each under 40 words.

89. Companion behaviour.

Helpful, not in the way.

Design a companion AI for [game]. Cover follow distance, combat help, the rule for when they act alone, pathing through tight spaces, and how they call out threats. List 4 ways companions usually annoy players and the fix.

90. Dialogue localisation check.

Keep lines translatable.

Review these lines for localisation: [paste lines]. Flag idioms, wordplay, and gendered phrasing that will not translate cleanly, and rewrite each with a neutral alternative.

11. Audio and music

91. Adaptive music brief.

Music that follows play.

Design adaptive music for [game]. Pick 3 intensity layers, define what triggers each transition, and describe how stems loop and crossfade. Keep the palette to 3 instruments.

92. SFX prompt: impact.

A layered hit.

Create a sound effect: a [style] [metal / flesh / wood] impact. Layer a transient, a body, and a tail. 1 second, dry, no music, mono-friendly. Give one high and one low intensity version.

93. SFX prompt: ambience.

A loopable room tone.

Create a 30-second seamless ambience loop for [location]. Include [distant detail, mid detail, close detail] layers. No discernible music, no voices, consistent loudness for looping.

94. UI sounds.

A matching set.

Create a set of 6 UI sounds for [click, confirm, back, error, coin, level-up]. Same family of tone, short, dry, consistent loudness. Match a [style] game. No reverb.

95. Music cue by mood.

A track that fits a moment.

Create a [length] music loop for [scene] at [tempo] BPM in a [genre] style. Mood: [describe]. Instrument set: [list]. Build to a small peak and return so it loops cleanly. No vocals.

96. Mixing check.

Find what is masking.

My game mix feels muddy during combat. List the 6 most likely frequency clashes between [SFX, music, dialogue], the tool to find each, and the fix. Give target loudness in LUFS for each bus.

12. XR design and comfort

97. Locomotion comfort.

Reduce motion sickness.

Design a locomotion scheme for a VR game set in [environment]. Compare teleport, snap turn, and smooth motion, and recommend a default for comfort with an option for experienced players. List 4 comfort settings to include.

98. VR interaction.

Hands, controllers, gaze.

For a VR game where the player [action], choose the interaction method (hands, controllers, or gaze) and justify it. Cover grab feedback, reach comfort, and the interaction distance. Note the tracking limits.

99. VR UI placement.

Comfortable to read.

Place UI for a VR game: main menu, health, and prompts. State the distance and angle from the player for each, the text size for readability, and how the UI behaves when the player looks away.

100. XR performance budget.

Hit the frame target.

I am building a [Quest / PCVR] game. Give me a performance budget: target frame rate, draw calls, triangle count, texture memory, and the 4 most important optimisations for this platform. Explain which to check first.

13. Code and systems: Roblox (Studio and Luau)

101. Place skeleton.

Lay out a Studio project you can grow.

I am building a Roblox experience: [one sentence]. Give me a `ServerScriptService`, `ReplicatedStorage` and `StarterPlayer` folder layout, name the 8 scripts and modules to create, and say what belongs on the server, on the client, and in a `ModuleScript` both can require. Keep it to what a first playable needs.

102. Server and client split.

Stop trusting the client.

Here is my Roblox feature: [describe the action, e.g. buying an item, taking damage, opening a door]. Tell me which logic must run on the server, which can run on the client for responsiveness, and exactly what the client is allowed to decide. List 5 things an exploiter would try and how the server should reject each.

103. Remote events and validation.

Safe client-server messages.

Write the Luau for a `RemoteEvent` that [action], split into a server script and a client script. Validate every argument on the server (type, range, rate), use a cooldown, and never trust distances or amounts sent by the client. Add a short comment on each guard so I know why it is there.

104. DataStore saves.

Persist player progress without data loss.

Write a Luau `DataStore` module that saves and loads [what to save] for each player. Include a session lock so two servers cannot overwrite each other, a retry with backoff on failure, and a `BindToClose` handler that flushes on shutdown. Explain the request budget I must stay under and how to avoid throttling.

105. Rojo and version control.

Keep the place in a git repo.

Set up a Rojo workflow for a Roblox project: the `default.project.json` mapping for `ServerScriptService`, `ReplicatedStorage` and `StarterPlayerScripts`, the folder layout on disk, and the commands to sync into Studio. Tell me which files belong in git and which to ignore, and how to keep two people from clashing.

106. Pathfinding NPC.

A character that walks the map sensibly.

Write Luau for an NPC that [behaviour, e.g. patrols waypoints, chases the nearest player, returns to a post]. Use PathfindingService, recompute when the path is blocked, and handle stuck states with a timeout. List the PathfindingModifiers I should add for doors and hazards.

107. Pass and product copy.

Honest store descriptions.

Write Roblox store copy for a game pass and a developer product: [what each gives the player]. Give a short name under 30 characters, one line that says exactly what the buyer gets, and a longer description. No pay-to-win claims, no pressure, no promises about outcomes, and no wording that could read as targeting under-13s.

14. 3D pipeline: Blender

108. Blender Python batch export.

Export many objects at once.

Write a Blender Python script (bpy) that exports every object in a collection to [FBX / glTF] with the settings a game engine expects: apply modifiers, correct scale, forward axis [Y or -Z], and one file per object named after the object. Add error handling for objects with no mesh and print a summary at the end.

109. Game-ready rig checklist.

A skeleton that animates cleanly.

I have a character mesh for [game type] that I need to rig for [engine]. Give me a bone checklist: the minimum bone set, naming convention, where to place the root and pelvis, and the joints to weight-paint first. List the 5 weighting mistakes that cause distorted deformation and how to spot each in weight-paint mode.

110. UV unwrap and packing plan.

Use texture space well.

This mesh is [describe it] and will use a [resolution] texture. Give me a step-by-step UV plan: which seams to mark and why, how to keep texel density even across the model, and how to pack islands to waste the least space. Note where a trim sheet would save more space than a full unwrap.

111. Bake normal and ambient occlusion maps.

Move detail from high to low poly.

Explain how to bake a normal map and an ambient occlusion map from my high-poly model to my low-poly game mesh in Blender. Give the bake settings, the image size to use for [target], the cage and ray-distance tips to avoid artefacts, and the common causes of black or wavy bakes with the fix for each.

112. Geometry Nodes scatter.

Fill a scene with vegetation or props.

Write a Geometry Nodes setup that scatters [asset] across a surface with natural variation: random rotation, non-uniform scale within [range], density control, and exclusion where the surface is too steep or too close to a path. Give me the node list in order and which values to expose as a panel for quick tweaking.

113. LOD plan.

Cut triangles without ugly pop.

Here is my asset at [triangle count] for [use]. Give me an LOD plan: the triangle target for each of 4 levels, what to remove first (silhouette-preserving), and the screen-size switch points for [engine]. Tell me whether to build LODs by hand, decimate, or generate in-engine, and why.

114. Engine export settings.

Land the asset cleanly in the engine.

I am exporting a [type] from Blender to [Unity / Unreal / Godot] at [real-world size]. Give the exact export settings: scale, forward and up axis, unit system, whether to apply transforms, and how to name the file. List the 5 import problems this avoids (flipped normals, wrong scale, missing materials) and how to check each after import.

15. Image pipelines: ComfyUI

115. Character consistency.

One character across many images.

Build a ComfyUI workflow to keep one character consistent across [number] images of [poses / expressions]. Describe the nodes in order, how to fix the character's identity (reference image, IPAdapter or a trained LoRA), and the seed and prompt-format rules to keep features stable. Note what still drifts and how to correct it.

116. ControlNet pose and layout.

Match a reference exactly.

I want a generated image where the [subject] holds this pose: [describe or attach]. Give me the ComfyUI node graph using ControlNet for pose and, if needed, depth for layout. State the preprocessor for each, the strength to start with, and how to stop the ControlNet from fighting the prompt.

117. Style LoRA brief.

Train the model on your own art.

I have [number] images in my own style for [subject]. Write a training brief: captioning rules (what to describe, what to leave implied), the trigger word to use, a starting learning rate and epoch count for a [small / medium] dataset, and how to tell overfitting from underfitting in the test generations. Note the resolution and aspect-ratio advice.

118. Seamless texture pipeline.

Tileable materials from a prompt.

Give me a ComfyUI workflow that produces a seamless, tileable [material, e.g. stone, sand, fabric] texture at [resolution]. Describe the nodes, the prompt structure, how to make edges wrap (offset and blend or a tile-aware node), and how to verify there is no visible seam before it goes into the engine.

119. Sprite sheet from a grid.

Consistent 2D frames.

Build a ComfyUI workflow that outputs a sprite sheet of [character] in a [N]-frame [walk / idle / attack] cycle, all frames the same size and consistent between frames. Explain how to lock the character, keep the grid aligned, and why AI usually breaks frame-to-frame consistency, with the setting that helps most.

120. Batch generation and naming.

Produce many assets in one run.

I need [number] variations of [asset type] for [game]. Give me a ComfyUI setup for batch generation from a list of prompts (a text file or a batch node), a naming convention that encodes the category and a counter, and an output-folder structure an engine can import. Add one quality-control step to reject obviously broken images automatically.

121. Troubleshoot a ComfyUI graph.

Diagnose what broke.

My ComfyUI workflow [does this wrong: e.g. outputs noise, runs out of VRAM, ignores the prompt]. Here is my node graph: [paste]. List the 6 most likely causes in order of probability, the exact node or setting to check for each, and the fix. Ask me for the one piece of information you need if the graph alone is not enough.

Written by theaigamedev.com. Free to keep and share; please do not resell. (c) 2026.