Godot For Scrap Metal

Porting Your Own Games To Retro Gaming Handhelds

April 15, 2025 – By Pyral of Dinoleaf

TALK
CHECK
GEAR
TECH
STATUS
QUIT
HP 240/240
"Plant monsters have tough hides against bludgeoning. A strong slashing blade can make them cower."
X
Y
A
B

Table of Contents

Introduction

For a while now, there’s been a growing market of cheap handheld game consoles. While some of these are pretty bad, there’s a number of surprisingly good value-for-cash budget consoles. They’re not very powerful, but they’re just strong enough that with the right software, they can run Godot games.

This guide won’t cover the history or what console’s best for what, because I only have so much time. I can only give some personal recommendations based on what I’ve used in passing. This is primarily focused on low-specs Linux handhelds, but there are stronger units which can support both Linux and Android-based firmware.

This is a very rough overview of the process, from selecting a device of your own, to getting Godot games to run on it, to having some extra tips and tricks for optimization. Consider this an early draft for materials that can explain this better in the future.

The scope is to get a device and get games running, using a compatibility layer called Westonpack, which is provided by Portmaster. Portmaster is effectively a crew and kit that helps with porting games that don’t belong on these insane handhelds… to these insane handhelds.

I must preface that this isn’t the most simple guide, and there is some novelty to this. It is also in early phases where streamlining isn’t feasible across every possible device and firmware, and we can only assume so much of your personal tech skills. Anyone can do this, but it does take a bit of exploration, reading, and figuring out. Perhaps sometime a better guide will arise to help take random Godot devs from zero to running. You best temper expectations with developing games specifically for this hardware if you really want to get into this niche, scaling your games up for more powerful/standard hardware after.

Current Limitations And Notes

Godot 3.x and 4.x work well, but there are some limitations due to the insane tech stack and limited hardware to make note of.

- As of writing, C# Mono builds are unavailable. But this may possibly change in the future, so keep an eye on this scene if you’re still interested.
- The game cannot rely on GDNative/GDExtension libs that don’t provide ARM64 Linux builds.
- Vulkan is not typically available. Your game will be running in GLES2 (Compatibility). This means some shaders and features may not be available, especially in 3D.
- Due to performance limitations, using 3D in Godot 4.x is mostly viable with 4.4 and beyond, as 4.4 has added back support for project-wide **vertex lighting**. Prior 4.x versions only provide pixel shading which is too intensive for this hardware.

- Your game must be able to build to ARM64 and the PCK must contain either no texture compression (which is more expensive on RAM) or you must enable ETC2/ASTC compression in the export settings.
- You can use custom Godot runtimes, but this guide assumes you only need vanilla ARM64 Linux runtimes.

Technical Specs Minimum Recommendations

For running Godot games, you want a device with at least 1GB of RAM. This provides a lot of space for simple games, and enough space for games with lots of complex assets to be cached, while optimizing for memory if you do manage to hit the limit. The OS, Godot runtime, and compatibility layers will take a chunk of RAM, which could typically be about 300MB, so you functionally should assume you have 500-700MB of free RAM. Most devices on the market have at least 1GB RAM at a low price point.

There are two particular CPUs these devices use that are good minimum chips: The RK3326 and the H700. Both are ARM64 processors.

The RK3326 is a bit slower, but still serviceable for simple 2D Godot games and maybe the bare simplest of 3D games. If you want bottom of the barrel budget for target hardware, the **G350** handheld is serviceable. It has no wifi and requires either a dongle or manually updating files on the SD card, but it’s also one of the cheapest devices around. The **R36S** is equivalent in power, but older and typically less well built, and has a lot of knockoffs on the market that are easy to be confused with as of writing. I personally use this as a secondary low-specs testing device, but don’t recommend it without a wifi dongle for an absolute beginner, as Portmaster takes time to set up.

The H700 is a bit faster, and holds surprising potential. It handles 2D Godot games like a champ, and can even do respectable 3D games. It is still very limited, however. Many devices use this, namely the Anbernic RG XX series. I personally use the **Anbernic RG35XX-SP** because of the clamshell form factor, and **Anbernic RG40XX-H** for analogue stick testing. Anbernic is just one company that has a lot of different form factors, but they’re also just one company.

If you can make your game run well on this hardware, it will likely run well on more powerful expensive stuff just fine. You might be able to run simple Godot games on even lower specs, but it will become increasingly difficult due to the hardware limitations, OS + engine overhead, and software support. Eventually, it becomes better value-per-dollar to just stick to this device range as a minimum.

Device Setup

When you receive your device of choice, there are a few things to make note of to get to a comfortable position of development.

First, **the SD card it comes with probably sucks.** For low end Anbernic devices, for example, they tend to come on Kioxia cards, and may even come with some unbranded SD cards. These are basically sugar wafers and will corrupt and die sooner than later, especially as you update your game for testing often. While some devices have internal memory, some have two SD card slots. Most firmware will support using one SD card for the OS, and the second card for data like games, saves, etc. You can get 16GB SD cards pretty cheap. Get reputable brands like SanDisk, and if Amazon's too risky for you, go run over to Best Buy to grab some. These sugar wafer SD cards can serve as a temporary solution while you mess around and learn your system, but they shouldn't be permanent.

Second, **the firmware the system comes with probably sucks.** More often than not, they're out of date or an unsupported firmware for Portmaster. It's best to install one that's better maintained. You'll want to do some of your own research, but the four firmware Portmaster tends to focus on are:

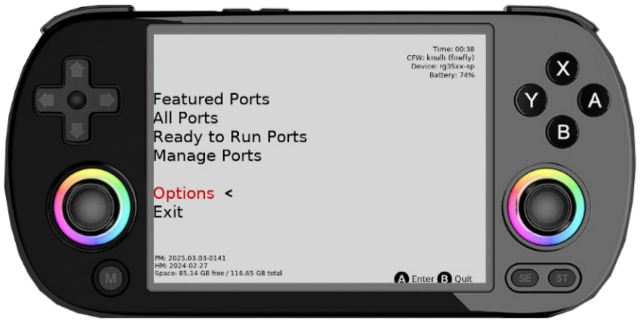
- AmberELEC
- ArkOS
- ROCKNIX
- muOS

Any one of these should be safe bets for a beginner. Their respective sites all contain guides on how to set up for your device. You'll typically be downloading their firmware and flashing it to an SD card. If set up correctly, your device will boot into it and you'll be free to set it up however you'd like. If your device has wifi, you'll also be able to set it up so you can access your device's files without having to remove the SD card every time, and you can even SSH into it for some automated testing and tooling.

Portmaster Setup

Portmaster is the repository and kit that contains a bunch of pre-configured games to run on these handhelds. Each firmware likely has its own way to install Portmaster, but it should be simple enough to find how to do from the respective firmware websites or the device itself. Ensure you run it, so it can do its own setup.

As a game developer, it's worth knowing how to grab the packages needed to run Godot games. The standard Godot runtime scripts automatically download these, but knowing how to do it manually doesn't hurt. From Portmaster's main menu, go to Options → Runtime Manager and select any Godot versions you'll be using. You should also download Westonpack, which contains compatibility tools to make Godot games run consistently across supported systems.

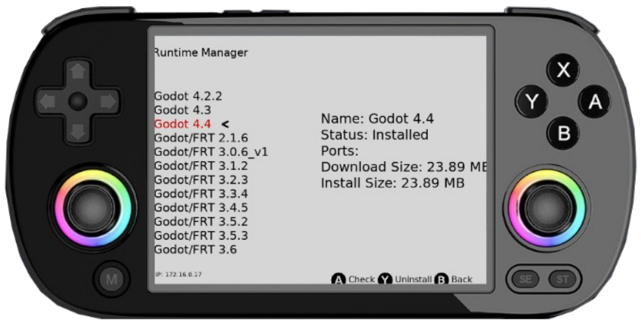


```
                                   Time: 00:38
                            CFW: knulli (firefly)
                              Device: rg35xx-sp
                                   Battery: 74%

Featured Ports
All Ports
Ready to Run Ports
Manage Ports

Options  <
Exit

PM: 2025.03.03-0141
HM: 2024-02-27
Space: 85.14 GB free / 116.65 GB total
```



```
Runtime Manager


Godot 4.2.2
Godot 4.3
Godot 4.4  <             Name: Godot 4.4
Godot/FRT 2.1.6          Status: Installed
Godot/FRT 3.0.6_v1       Ports:
Godot/FRT 3.1.2          Download Size: 23.89 MB
Godot/FRT 3.2.3          Install Size: 23.89 MB
Godot/FRT 3.3.4
Godot/FRT 3.4.5
Godot/FRT 3.5.2
Godot/FRT 3.5.3
Godot/FRT 3.6

IP: 172.16.0.17
```

Prepare Your Game PCK

Exporting your project is simple enough. To ensure a smooth runtime test, export as an ARM64 release build. You'll only need the PCK and whatever external files you need. Note that your build must…

- Not be a Mono C# build.
- Have ETC2/ASTC texture compression enabled.
- Not require GDExtension libraries that lack ARM64 support.

You won't need the runtime itself, just the PCK and related files.

Your Game Setup

Portmaster should be ready now. Depending on your OS, the exact location of where to place games vary, but usually it'll be in a subdirectory of **roms/ports/**. The easiest way would be to access it over the network after connecting to wifi and enabling sambda and finding the folder in particular. You now need a boot script and a directory containing your game data. Create a subfolder with some kind of title, ideally without spaces or weird characters.

You can get an example script for running Godot games here: https://github.com/binarycounter/Westonpack/wiki/Godot-4-Example

Copy and rename the script to your game title, saving it to the root of your ports folder as **Whatever Your Game Is.sh**. Edit the following lines:

```
# Adjust these to your paths and desired godot version
GAMEDIR=/$directory/ports/your_port_folder
godot_runtime="godot_4.2.2"
godot_executable="godot422.$DEVICE_ARCH"
pck_filename="YourGame.pck"
gptk_filename="YourControls.gptk"
```

Note that some maintenance releases may not be available, such as godot441 for 4.4.1, but 4.4 should work for such a case.

GAMEDIR should point to your subdirectory. YourGame.pck should be changed to the main pck (typically the only one you export for smaller projects.)

In your subdirectory, you should also create a basic text file matching gptk_filename. This allows you to map controls to buttons on Godot. Sometimes button mapping is weird and different across firmwares, so this may need some tinkering, or your game may want to support input remapping (which is good for PC games as well.) See https://portmaster.games/gptokeyb-documentation.html for information on writing a gptk file.

For Devices Without Wifi

If you have something like the G350 or R36S, where there is no built-in wifi, you'll need to set Portmaster up with the SD card directly. There's instructions for how to do this at https://portmaster.games/installation.html. You can manually download and add runtimes for Godot and Westonpack from https://github.com/PortsMaster/PortMaster-New/tree/main/runtimes, where you will want to add them to Portmaster's internal **libs/** subdirectory. This will vary based on your firmware.

Running Your Game

After adding your subdirectory and boot script, restart your system so it can appear in the game explorer. If all went well, and it never does on the first try, then you should see your game as one of the entries. Select it and see if it runs. If it doesn't, then your game's subdirectory should have a log.txt which you can search for issues. Often, it could be something like you tried using a non-ARM64 library, you didn't name or place something right, or you didn't export compressed textures correctly (not enabling ETC2/ASTC on export will throw missing image asset errors.) You can CTRL+F to see if "Godot" is booting and what errors spit out. The Portmaster Discord can also typically look through logs to clarify something's misplaced or unsupported.

If your game boots and your input's working well enough to play, then congrats. This isn't the end, but the beginning.

Optimizations and Etiquette

If your game runs like molten caramel, then it may need optimizations. When you develop for a PC, you have a lot of space and power to waste, but on this hardware, you're going to be squeezing for power if you do anything crazy.

While these might seem like a pain to learn and do, they are excellent practices and ensure your game will run on just about any modern, more powerful hardware that can run a Godot game. You'll also become a better software engineer as you learn how to max out weak, cheap hardware. "Lateral thinking with withered technology" and all that.

My game pauses briefly during gameplay/scene loading

Modern PCs tend to be using a fast SSD to load, which is near instant. These devices are reading your game from an SD card and have slow IO/loading. If you are loading assets and packed scene instantiation on demand, or making sudden allocations for a new resource, then you may see this happen. Depending on what you're doing, there's a few optimizations you can do.

For loading new levels, you can fade out. Ideally, you can use **async loading** to make a loading screen, if it takes more than a second. Godot has resources on how to set up async loading. See https://docs.godotengine.org/en/latest/tutorials/io/background_loading.html for details.

If you're having realtime hitches, such as firing bullets for the first time or explosions spawning, then there's a few ways to improve that. A common pattern is **object pooling**, where you can preallocate inactive instances before they're needed and reset/activate them when they're needed. This ensures more assets that you need are preloaded into RAM and individual nodes don’t have to be reallocated.

My game crashes when loading scenes/big stuff happens/on boot

It’s possible you’re exhausting all your RAM. You can estimate you’ve got about 700MB of RAM on a 1GB device, possibly more or less. Since you may want to pool and preload assets to prevent mid-game hitching, it’s possible you may also load too much and crash the game. To reduce this chance, start by cutting down your assets in a given scene or moment. If you’re making a PS1-style horror game, you don’t need textures that are 4096x4096. Use smaller textures. Pick your audio formats based on your needs. Importantly, plug any memory leaks. Godot will hold onto memory for a while and manages its own internal stuff pretty well, but you’ll want to look at practices to ensure you don’t have assets dangling in memory, taking up space.

Transferring PCK updates takes forever because my game is 100MB

Godot lets you split PCK files via separate export configurations. If you have a bunch of heavy assets that aren’t changing much, you can set Godot up to exclude the subdirectory they’re in for the main game and then set up a separate export config that contains only the assets themselves. This way, you can send a PCK of 100MB of textures and audio once (until you add more) and then export your main game PCK, which might just be scenes, scripts, and some common assets.

This does require some manual setup, and your game must load these extra PCK files separately on boot before accessing anything, but this is doable with a simple pre-boot scene or whatever you wish to use. See https://docs.godotengine.org/en/stable/tutorials/export/exporting_pcks.html for more info on how to do this.

SD Card Wear and Tear

SD cards have limited writing capabilities before they start to burn out and corrupt. This is why it’s advised to **replace cheap, offbrand SD cards with reliable ones.** If you’re doing a lot of writes, consider getting a high-endurance SD card, which is more expensive but better fit for tasks like embedded hardware systems (Raspberry Pi drives, for example.)

Because you never know what the player’s SD card is, it is wise to consider this if your game uses save data. If you save your game using a Godot resource, try saving as **.res** instead of .**tres**. You can also use tricks like a separate config file that saves MD5 checksums for a save file, to ensure integrity. Or you can copy and compress backups to save your assets.

Finally, if your game writes a ton of different stuff, you can use compression and separate files for different large-scale save data and page them out, letting you save writes by writing to small, separate files that are loaded in rather than one monolithic file that’s always loaded. If you don’t need all your save data loaded in at once, this can also save on RAM. These techniques are for games that save and load especially large amounts of data, but is unlikely to matter much for the average small-scale Godot game.

Also, when combined with the PCK splitting, you can save wear and tear on your system’s SD card, as you won’t have to update nearly as much data as you would if you had transferred tens or hundreds of megabytes multiple times just to change and test small adjustments.

3D Graphics

3D is very limited. As said, you’re on GLES2 and don’t have access to Vulkan APIs. Some 3D features of Godot may not be supported as a result. The editor will often warn you of such cases when you edit the project in Compatibility mode.

Shaders will need to be hand-written and optimized for their purpose. It won’t do well to just grab the first thing you see on GodotShaders. An H700-CPU is probably the bare minimum you want to use for developing 3D games. With the right hands and optimizations, I believe you can get graphics on par with Gamecube or 3DS at 30FPS in Godot, but even this is a very generous guesstimate.

The best 3D games would be ones that make careful use of materials, preload all the assets they need, page out what it doesn’t, and especially are focused on close-range scenes. Even an H700 unit can handle decent polygon counts. Your bottlenecks will most likely come from shaders, physics, and running complex animations. Consider all of Godot’s tools to make use of multi-threading and profile your game’s performance to the best of your ability.

For Godot 4.x games, you will only want to really do 3D with 4.4 and later. 4.0 had removed vertex lighting, and it was not restored until 4.4. Pixel lighting is too intense for good frametime on these devices, especially when needing to optimize so many other things. Jolt physics was also made native to Godot 4.4, which may help with physics-based performance issues.

Screens and Resolutions

Consider optimizing for aspect ratios and know the common screen sizes. Most screens are 4:3 or 16:9, with some cases of 16:10 and a rare 3:2 like the RG34XX, which imitates the design of the Gameboy Advance. There’s even a few 1:1 aspect ratio devices like the Cube series from Anbernic. Most 4:3 screens are 640x480. See Display → Stretch Mode in Godot’s Project Settings for the different ways you can scale a game. A game that can change its resolution and aspect ratio, especially a pixel art game, can scale to many of these screens very well if you put some research into this.

Major resolutions include 480x320, 640x480, 720x720, and high resolutions like 1280x720. Be aware that HDMI output can raise a device’s internal resolution, which may reduce performance on games that are already pushing the device to its limits.

This Is Hard

Maybe, but it’s funny.

```
PU OpenGL ES 3.2 v1.r20p0-01rel0.8ec337de4ddab3365a32cff58063a9da
1EM 973M/(30M/30M)
PS 40 (C0ms G0ms)
```

Outro

This isn’t the most well put together guide, but it’s one of the only ones that exists in the first place for the specific niche topic of Godot games running on retro gaming handhelds. This should be more seen as a jumping off point for better writers and supporters to do and explain how to do this, as well as for more specific hardware + firmware setups. I’d say developing for these devices is extremely rewarding. There’s nothing quite like playing your own game on something that feels like a super-GBA SP.

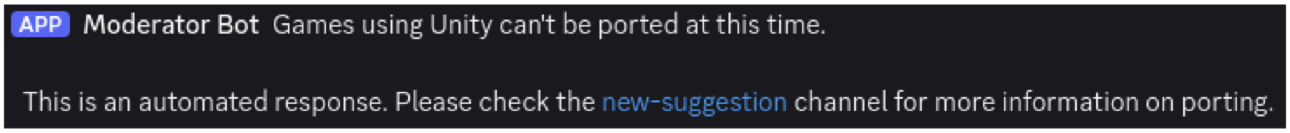
Westonpack also allows for porting other game runtimes and engines that previously weren’t possible, but to do so will require specifically tailoring example scripts and setups that go far out from the scope of this Godot hackjob guide. Portmaster also generally supports a number of runtimes and game engines (Game Maker, when not using YYC, is another popular choice.)

If you’d like to help spread awareness of this being something now feasibly possible to do with Godot, feel free to share this document with potential gamedev circles, Godot dev circles, etc that might be interested. I mostly wrote it because I posted to a few places showing this was doable and got a surprisingly large amount of interest and questions.

Credit Where Due

- **Dinoleaf** which is my company. I wrote this because it’s a necessary starter doc right now. You’re welcome. See our stuff at https://dinoleaf.com. Regards, @pyral@dinoleaf.com.
- **Anbernic, Retroid, TrimUI, etc** for digging through their random Chinese warehouses and turning all this excess junk into funny game handhelds.
- **Firmware devs** who for some reason felt the need to jam Linux onto the weirdest of hardware for running old games.
- **Portmaster Crew** for getting the infrastructure and process figured out for running ports on these cheap handhelds. Throw them a bone for their efforts. They’re way underfunded.

- **BinaryCounter** for ignoring Portmaster's autobot telling everyone constantly that Unity games can't be ported, as his Westonpack compatibility layer is what makes this possible for many devices.


APP Moderator Bot Games using Unity can't be ported at this time.

This is an automated response. Please check the new-suggestion channel for more information on porting.