ChatGPT

Gmod How To – Building a Single-Player DarkRP-Like Gamemode

Introduction

Garry's Mod (GMod) is a physics sandbox built on the Source engine. Players can build contraptions, spawn NPCs and weapons, and create custom game modes using Lua. **DarkRP** is one of the most popular role-play gamemodes; it simulates urban life where players take on jobs like police officer, gangster or shop owner [1]. In a single-player environment you will need to implement basic systems (money, jobs, properties and NPC interactions) yourself. This guide shows how to structure a custom gamemode folder, introduce core systems and provide a starting codebase for a **"LonerRP"** – a lightweight single-player DarkRP-style experience.

1. Gamemode Structure & Setup

1.1 Folder layout

Garry's Mod looks for gamemodes in the `garrysmod/gamemodes/` directory. Create a folder named after your gamemode (e.g., `loner_rp`). Inside it create a `gamemode/` folder and place three key Lua files:

File	Purpose
`init.lua`	**Server** realm. Runs only on the server and contains game rules, player management, entity spawning and server-side hooks. It must call `AddCSLuaFile()` to tell the client to download client files [2].
`cl_init.lua`	**Client** realm. Runs only on clients and defines HUD, client-side UI and visual hooks. Use `include()` to load shared code and `--` to comment out server-only sections [3].
`shared.lua`	**Shared** realm. Code accessible from both client and server, such as entity definitions and networked variables. Use `AddCSLuaFile()` at the top so it gets sent to clients.

To register the gamemode you must also create an `info.txt` at the root of your gamemode folder. This key-value file defines the title, author, version and category. Valid categories include `rp`, `pvp`, `pve` and `other` [4]. Example:

```
"name"       "LonerRP"
"author"     "YourName"
"version"    "0.1"
```

```
"description" "Single-player DarkRP-style gamemode"
"category"    "rp"
```

1.2 Deriving from Sandbox

Most custom gamemodes derive from the default Sandbox gamemode to inherit basic spawning, toolgun and physics behaviour. In `init.lua` and `cl_init.lua` call `DeriveGamemode("sandbox")` [5] to inherit Sandbox functions before defining your custom hooks.

1.3 Loading files – `include` and `AddCSLuaFile`

- `include("file.lua")` executes another Lua file in the current realm. For example, the server's `init.lua` might include a shared configuration file so both server and client share variables [2].
- `AddCSLuaFile("file.lua")` should be called from the server realm to tell clients to download a clientside file. Without `AddCSLuaFile`, clients will not run your script [2].

1.4 Realms & Networking

Garry's Mod runs separate Lua states for the client and server. Global variables do not cross these realms; the only way to communicate is via the `net` library or networked variables [6]. Always check the realm using the `CLIENT` or `SERVER` boolean before running code. For events triggered on both realms, you can place code in `shared.lua` or call `AddCSLuaFile` for clientside scripts.

1.5 Hooks and the hook system

The hook system lets you run code when certain events occur. Use `hook.Add(eventName, identifier, function)` to register a function. The first argument is the name of the event (hook), the second is a unique string (used to remove or override the hook), and the third is the function to run [7]. Hooks are used for actions such as players spawning, using entities or dying. For example, `hook.Add("Think", "PrintEveryTick", function() print("Tick") end)` will print a message every tick [8]. Overriding a `GM:*` method inside your gamemode (e.g., `function GM:PlayerSpawn(ply)`) will also handle those events, but using hooks is safer when adding multiple functions.

1.6 Important hooks

- `GM:PlayerInitialSpawn(ply, transition)` – Called once when a player first spawns. It runs before the player fully loads; sending `net` messages here is unreliable, so perform heavy setup in later hooks [9]. Use this to initialise data tables or assign default money.
- `GM:PlayerSpawn(ply)` – Called every time the player spawns. Use it to give weapons or models.
- `GM:PlayerUse(ply, ent)` – Called when a player presses the USE key on an entity. You can return `false` to prevent them from interacting [10].

2. Basic Systems for a Single-Player RP

A role-play gamemode needs systems for money, jobs and NPC interactions. In a single-player environment you can store this data as networked variables on the player; these variables replicate automatically from server to client.

2.1 Player money

Networked variables allow the server to store data on an entity that is automatically available to all clients. Use `SetNWInt` and `GetNWInt` on the player entity to manage an integer field such as money. `SetNWInt` sets a networked integer and clients can read it using `GetNWInt`[11]. Example server-side function:

```lua
-- In shared.lua or a server file
local function SetMoney(ply, amt)
    ply:SetNWInt("money", amt) -- store integer across network
end
local function AddMoney(ply, amt)
    local current = ply:GetNWInt("money", 0)
    SetMoney(ply, current + amt)
end
local function GetMoney(ply)
    return ply:GetNWInt("money", 0)
end
```

To persist money across sessions, avoid storing on the client. Use file-based storage on the server.

2.2 Jobs system

DarkRP's job system is built using Lua tables that define properties such as name, model, salary and access. In a custom gamemode you can implement a simplified version by storing the current job as a string networked variable on the player. For example:

```lua
-- Server: define available jobs
g_JOBS = {
    Citizen = { salary = 45, model = "models/player/Group01/male_01.mdl" },
    Police  = { salary = 65, model = "models/player/police.mdl", weapon =
"weapon_pistol" }
}

function SetJob(ply, jobName)
    local job = g_JOBS[jobName]
    if not job then return end
    ply:SetNWString("job", jobName)
    ply:StripWeapons()
    ply:SetModel(job.model)
    if job.weapon then ply:Give(job.weapon) end
end
```

Create a simple console command or Derma menu to let players choose their job. For example, in `cl_init.lua` you can open a frame with buttons for each job. When a button is clicked, send a net message to the server requesting the job change. On the server, handle the net message and call `SetJob`.

DarkRP uses `DarkRP.createJob` with fields like `color`, `model`, `description`, `weapons`, `command`, `max`, `salary`, `admin` and `vote` [12]. For inspiration, study those fields; for instance, `color` sets the job color on scoreboard and `max` limits the number of players per job [13]. In your single-player environment these restrictions can be simplified.

2.3 Simple interactive NPC

To simulate shopkeepers or quest givers, create a custom entity under `gamemode/entities/loner_rp_npc/`. An entity consists of `init.lua` (server), `cl_init.lua` (client) and `shared.lua` (shared). In the server file you set the model, define physics and handle the `Use` interaction. Example skeleton:

```lua
-- gamemode/entities/loner_rp_npc/init.lua
AddCSLuaFile("cl_init.lua")
AddCSLuaFile("shared.lua")
include("shared.lua")

function ENT:Initialize()
    self:SetModel("models/Humans/Group01/Female_01.mdl")
    self:SetHullType(HULL_HUMAN)
    self:SetHullSizeNormal()
    self:SetNPCState(NPC_STATE_SCRIPT)
    self:CapabilitiesAdd(CAP_ANIMATEDFACE + CAP_TURN_HEAD)
    self:SetUseType(SIMPLE_USE)
end

function ENT:AcceptInput(name, activator)
    if name == "Use" and activator:IsPlayer() then
        net.Start("open_npc_menu")
        net.Send(activator)
    end
end
```

On the client you receive `open_npc_menu` and build a Derma frame. Derma provides panels (e.g., `DPanel`, `DButton`) for UI elements. Create panels using `vgui.Create` and adjust size, position and visibility [14]. For example:

```
-- gamemode/entities/loner_rp_npc/cl_init.lua
include("shared.lua")
net.Receive("open_npc_menu", function()
    local frame = vgui.Create("DFrame")
    frame:SetSize(300, 200)
    frame:Center()
    frame:SetTitle("NPC Shop")
    frame:MakePopup()
    -- add buttons or labels here
end)
```

Remember Derma code runs only on the client; to trigger it from the server you must send a `net` message. Do not run Derma inside HUDPaint; instead, create a function to open the menu and call it via `net.Receive` [15].

2.4 Persisting data across sessions

Single-player gamemodes often need to save progress. Garry's Mod provides the `file` library for reading and writing files. You should never trust the client; always save data server-side. Convert tables to JSON using `util.TableToJSON`, write them to the data folder with `file.Write`, and read them back using `file.Read` and `util.JSONToTable` [16]. For example:

```
local DATA_FILE = "loner_rp/players.txt"

function SavePlayerData(ply)
    local t = {
        steamID = ply:SteamID64(),
        money   = ply:GetNWInt("money", 0),
        job     = ply:GetNWString("job", "Citizen")
    }
    local json = util.TableToJSON(t)
    file.Write(DATA_FILE .. "/" .. t.steamID .. ".txt", json)
end

function LoadPlayerData(ply)
    local path = DATA_FILE .. "/" .. ply:SteamID64() .. ".txt"
    if not file.Exists(path, "DATA") then return end
    local json = file.Read(path, "DATA")
    local t    = util.JSONToTable(json)
    if t then
        ply:SetNWInt("money", t.money)
        ply:SetNWString("job", t.job)
    end
end
```

Call `SavePlayerData` when the player quits (`PlayerDisconnected`) and `LoadPlayerData` when they spawn for the first time.

3. Advanced Concepts (optional)

3.1 Nextbot AI

Nextbot NPCs use Valve's navigation mesh system to provide advanced AI. Creating a police bot that patrols, detects crime and arrests the player is complex. You would need to implement pathfinding (using navmesh), crime detection (detecting illegal actions like damage or unauthorized entry), and pursuit behaviour. The official GMod wiki notes that Nextbots were originally created for Team Fortress 2 and Left 4 Dead, and building such AI is beyond a basic tutorial [17]. You may start by reading the Nextbot API to spawn a simple chasing bot and gradually add states for patrolling and apprehension.

3.2 Civilians with schedules and personalities

Simulating an entire city of civilians requires behaviour trees or state machines. Each NPC would need a daily routine (walking, sitting, buying items), a personality (fearful, aggressive, calm) and reactions to player actions. This is more advanced than a simple single-player mode. If you attempt this, consider using a finite state machine for each NPC and randomising traits. For example, assign each NPC a "bravery" value that influences whether they flee or confront the player.

3.3 Property buying system

A DarkRP property system allows players to purchase houses or rooms and restrict door access to owners. Implementing this in a single-player gamemode involves:

- **Property definitions:** Create a table of property zones (vectors or trigger areas) with price and door list.
- **Ownership tracking:** Store ownership in a persistent file or networked variable. When the player purchases a property, mark them as the owner and prevent other players (in multiplayer) from opening the doors.
- **UI:** Present available properties via Derma. When the player selects one, check they have enough money and deduct the price.
- **Door linking:** Doors are entities. Override `GM:PlayerUse` or set door key values to ensure only the owner may open them. The `properties` library also lets you add context menu options to entities [18].

3.4 Furniture system

Allowing players to buy and place furniture requires a shop NPC or menu and a placement system. Create custom furniture entities that players can spawn only within their property zone. Provide a preview model that can be rotated and placed via ghosting (similar to Prop Hunt). On placement, spawn the real entity, deduct money and save the furniture to persistent data.

4. DarkRP-specific Modifications

If you eventually decide to build on top of DarkRP rather than writing your own gamemode, follow the DarkRP modification guidelines:

- **Do not edit core files.** Editing DarkRP's base files prevents updates and breaks support. Instead, use the DarkRP modification addon located in `addons/darkrpmodification` [19]. Place custom jobs, entities, shipments and ammo in `lua/darkrp_customthings` and configuration in `lua/darkrp_config`.
- **Custom jobs:** Use `DarkRP.createJob` with fields like `color`, `model`, `description`, `weapons`, `command`, `max`, `salary`, `admin` and `vote` [12]. The `max` field can be a number limiting the total number of players or a fraction of the population [13]. Use `customCheck` for conditions restricting access (e.g., only admins).
- **Custom entities:** Create shipments or custom entities with fields `ent`, `model`, `price`, `max` and `cmd`. Optional fields like `allowed`, `customCheck`, `getPrice` and `getMax` enable dynamic pricing and restrictions [20].
- **Modules:** For deeper changes, write modules in `lua/darkrp_modules`. File names should be prefixed with `sv_`, `cl_` or `sh_` depending on the realm [21].

5. Putting It All Together – Example LonerRP Base Code

Below is a simplified scaffold for a single-player DarkRP-like gamemode. It includes the folder structure and fundamental files. Adjust names and add features as you expand.

```
loner_rp/
  info.txt          – metadata (see §1.1)
  gamemode/
    init.lua        – server logic
    cl_init.lua     – client logic (HUD & menus)
    shared.lua      – shared functions & variables
    entities/
      loner_rp_npc/
        init.lua    – custom NPC server code
        cl_init.lua – NPC client UI
        shared.lua  – NPC shared settings
```

info.txt

```
"name"        "LonerRP"
"author"      "YourName"
"version"     "0.1"
"description" "Single-player DarkRP-style gamemode"
"category"    "rp"
```

gamemode/shared.lua

```
DeriveGamemode("sandbox")

AddCSLuaFile() -- send this shared file to clients

-- Job table accessible on both server and client
g_JOBS = {
    Citizen = { salary = 45, model = "models/player/Group01/male_02.mdl" },
    Police  = { salary = 65, model = "models/player/police.mdl", weapon =
"weapon_pistol" }
}

-- Shared accessor functions
function GM:GetPlayerMoney(ply)
    return ply:GetNWInt("money", 0)
end

function GM:GetPlayerJob(ply)
    return ply:GetNWString("job", "Citizen")
end
```

gamemode/init.lua (server)

```
AddCSLuaFile("shared.lua")
AddCSLuaFile("cl_init.lua")
include("shared.lua")

util.AddNetworkString("change_job")
util.AddNetworkString("open_npc_menu")

function GM:PlayerInitialSpawn(ply)
    -- Give default money and job
    ply:SetNWInt("money", 100)
    ply:SetNWString("job", "Citizen")
    -- Load persisted data if exists
    if file.Exists("loner_rp/" .. ply:SteamID64() .. ".txt", "DATA") then
        local tbl = util.JSONToTable(file.Read("loner_rp/" ..
ply:SteamID64() .. ".txt", "DATA"))
        ply:SetNWInt("money", tbl.money or 100)
        ply:SetNWString("job", tbl.job or "Citizen")
    end
end

function GM:PlayerSpawn(ply)
```

```lua
    -- Set model and give weapons based on job
    local job = g_JOBS[ply:GetNWString("job", "Citizen")]
    if job then
        ply:SetModel(job.model)
        ply:StripWeapons()
        if job.weapon then ply:Give(job.weapon) end
    end
end

-- Handle job change requests from client
net.Receive("change_job", function(_, ply)
    local newJob = net.ReadString()
    if g_JOBS[newJob] then
        ply:SetNWString("job", newJob)
        ply:ChatPrint("Your job is now " .. newJob)
        -- respawn to apply model and weapons
        ply:Spawn()
    end
end)

-- Persist data when player disconnects
function GM:PlayerDisconnected(ply)
    local tbl = {
        money = ply:GetNWInt("money", 0),
        job   = ply:GetNWString("job", "Citizen")
    }
    file.CreateDir("loner_rp")
    file.Write("loner_rp/" .. ply:SteamID64() .. ".txt", util.TableToJSON(tbl))
end
```

gamemode/cl_init.lua (client)

```lua
include("shared.lua")

-- Basic HUD that shows money and job
hook.Add("HUDPaint", "LonerRP_HUD", function()
    local ply = LocalPlayer()
    draw.SimpleText("Money: $" .. ply:GetNWInt("money", 0), "Trebuchet24", 10,
10, color_white)
    draw.SimpleText("Job: " .. ply:GetNWString("job", "Citizen"),
"Trebuchet24", 10, 30, color_white)
end)

-- Derma menu to change job
local function OpenJobMenu()
    local frame = vgui.Create("DFrame")
    frame:SetTitle("Choose Job")
    frame:SetSize(200, 100)
    frame:Center()
    frame:MakePopup()

    local y = 30
    for name,_ in pairs(g_JOBS) do
        local btn = vgui.Create("DButton", frame)
        btn:SetText(name)
        btn:SetPos(10, y)
        btn:SetSize(180, 20)
        btn.DoClick = function()
            net.Start("change_job")
            net.WriteString(name)
            net.SendToServer()
            frame:Close()
        end
        y = y + 25
    end
end

concommand.Add("lrp_job", OpenJobMenu)

-- Receive NPC menu net message
net.Receive("open_npc_menu", function()
    -- open NPC shop UI; for now just print
    chat.AddText("You talked to the NPC. (Implement shop here)")
end)
```

gamemode/entities/loner_rp_npc/shared.lua

```lua
ENT.Base = "base_ai"
ENT.Type = "ai"
ENT.PrintName = "Shopkeeper"
ENT.Category = "LonerRP"
ENT.AutomaticFrameAdvance = true

function ENT:SetAutomaticFrameAdvance( bUsingAnim )
    self.AutomaticFrameAdvance = bUsingAnim
end
```

gamemode/entities/loner_rp_npc/init.lua (server)

```lua
AddCSLuaFile("cl_init.lua")
AddCSLuaFile("shared.lua")
```

```
include("shared.lua")

function ENT:Initialize()
    self:SetModel("models/Humans/Group01/Female_02.mdl")
    self:SetHullType(HULL_HUMAN)
    self:SetHullSizeNormal()
    self:SetNPCState(NPC_STATE_SCRIPT)
    self:CapabilitiesAdd(CAP_ANIMATEDFACE + CAP_TURN_HEAD)
    self:SetUseType(SIMPLE_USE)
end

function ENT:AcceptInput(name, activator)
    if name == "Use" and activator:IsPlayer() then
        net.Start("open_npc_menu")
        net.Send(activator)
    end
end
```

gamemode/entities/loner_rp_npc/cl_init.lua (client)

```
include("shared.lua")

function ENT:Draw()
    self:DrawModel()
    -- Optionally draw overhead text here
end
```

Using the gamemode

1. Place the `loner_rp` folder inside `garrysmod/gamemodes/`.
2. Launch GMod and start a single-player game. Choose **LonerRP** from the gamemode list.
3. Use console command `lrp_job` to change jobs. Spawn your custom NPC via the *spawnmenu* (Entities → LonerRP). Press USE on the NPC to open the placeholder shop menu.

Conclusion

This guide summarises the fundamental building blocks needed to create a single-player DarkRP-like gamemode. You learned how to structure your gamemode folder, manage money and jobs using networked variables [11], create interactive NPCs using hooks like `GM:PlayerUse` [10], persist data via JSON files [16] and design simple Derma UIs [14]. More advanced features, such as Nextbot police or property systems, require deeper knowledge of AI, pathfinding and networking [17]. The official Garry's Mod wiki and DarkRP pages provide further reference; avoid modifying core DarkRP files and instead use the `darkrpmodification` addon [19]. Continue experimenting, iteratively building features, and you'll create a compelling single-player role-play experience.

---

[1] DarkRP
https://darkrp.miraheze.org/wiki/Main_Page

[2] [3] Understanding AddCSLuaFile and include - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/Understanding_AddCSLuaFile_and_include

[4] [5] Gamemode Creation - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/Gamemode_Creation

[6] States / Realms - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/States

[7] [8] hook.Add
https://gmodwiki.com/hook.Add

[9] GM:PlayerInitialSpawn - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/GM:PlayerInitialSpawn

[10] GM:PlayerUse - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/GM:PlayerUse

[11] Entity:SetNWInt - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/Entity:SetNWInt

[12] [13] DarkRP:CustomJobFields - DarkRP
https://darkrp.miraheze.org/wiki/DarkRP:CustomJobFields

[14] Introduction To Using Derma - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/Introduction_To_Using_Derma

[15] Derma Basic Guide - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/Derma_Basic_Guide

[16] File Based Storage - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/File_Based_Storage

[17] NextBot NPC Creation - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/NextBot_NPC_Creation

[18] properties - Garry's Mod Wiki
https://wiki.facepunch.com/gmod/properties

[19] DarkRP:ModifyingDarkRP - DarkRP
https://darkrp.miraheze.org/wiki/DarkRP:ModifyingDarkRP

[20] DarkRP:CustomEntityFields - DarkRP
https://darkrp.miraheze.org/wiki/DarkRP:CustomEntityFields

[21] Creating modules - DarkRP
https://darkrp.miraheze.org/wiki/Creating_modules