📡 Sistema de Sincronização Customizada

🎯 Arquitetura do Sistema

```
Cliente Local → NetworkManager → Servidor → NetworkManager → Clientes Remotos
     ↓              (RPC)           ↓          (RPC)            ↓
  Envia Estado  Valida/Processa  Propaga   Recebe Estado  Aplica Estado
```

🔧 Como Sincronizar Qualquer Coisa

Passo 1: Adicionar RPC no NetworkManager

gdscript

```gdscript
# ===== EXEMPLO: Sincronizar Animação =====

# Cliente envia ao servidor
func send_player_animation(p_id: int, anim_name: String, anim_speed: float):
  if not is_connected:
    return
  rpc_id(1, "_server_player_animation", p_id, anim_name, anim_speed)

# Servidor recebe
@rpc("any_peer", "unreliable")
func _server_player_animation(p_id: int, anim_name: String, anim_speed: float):
  if not multiplayer.is_server():
    return

  var sender_id = multiplayer.get_remote_sender_id()
  if sender_id != p_id:
    return

  # Propaga para outros clientes
  var room = RoomRegistry.get_room_by_player(p_id)
  if room.is_empty():
    return

  for player in room["players"]:
    if player["id"] != p_id:
      rpc_id(player["id"], "_client_player_animation", p_id, anim_name, anim_speed)

# Cliente recebe
@rpc("authority", "unreliable")
func _client_player_animation(p_id: int, anim_name: String, anim_speed: float):
  if multiplayer.is_server():
    return

  var player = get_tree().root.get_node_or_null(str(p_id))
  if player and player.has_node("AnimationPlayer"):
    var anim = player.get_node("AnimationPlayer")
    anim.play(anim_name, -1, anim_speed)
```

Passo 2: Chamar do Player

gdscript

```gdscript
# No player.gd
func play_animation(anim_name: String, speed: float = 1.0):
 # Toca localmente
 if $AnimationPlayer:
  $AnimationPlayer.play(anim_name, -1, speed)

 # Sincroniza
 if is_local_player:
  NetworkManager.send_player_animation(player_id, anim_name, speed)
```

📦 Exemplos Práticos

1. Sincronizar Vida do Jogador

NetworkManager.gd:

gdscript

```gdscript
# Cliente informa mudança de vida
func send_player_health(p_id: int, health: float, max_health: float):
  if not is_connected:
    return
  rpc_id(1, "_server_player_health", p_id, health, max_health)

@rpc("any_peer", "reliable")  # reliable para dados críticos
func _server_player_health(p_id: int, health: float, max_health: float):
  if not multiplayer.is_server():
    return

  var sender_id = multiplayer.get_remote_sender_id()
  if sender_id != p_id:
    return

  # Atualiza no servidor (se tiver lógica autoritativa)
  ServerManager.update_player_health(p_id, health, max_health)

  # Propaga
  var room = RoomRegistry.get_room_by_player(p_id)
  if room.is_empty():
    return

  for player in room["players"]:
    rpc_id(player["id"], "_client_player_health", p_id, health, max_health)

@rpc("authority", "reliable")
func _client_player_health(p_id: int, health: float, max_health: float):
  if multiplayer.is_server():
    return

  var player = get_tree().root.get_node_or_null(str(p_id))
  if player and player.has_method("update_health_display"):
    player.update_health_display(health, max_health)
```

player.gd:

✓

gdscript

```gdscript
var health: float = 100.0
var max_health: float = 100.0

func take_damage(damage: float):
  if not is_local_player:
    return  # Só o local processa dano

  health = max(0, health - damage)

  # Atualiza UI local
  update_health_display(health, max_health)

  # Sincroniza
  NetworkManager.send_player_health(player_id, health, max_health)

  if health <= 0:
    die()

func update_health_display(hp: float, max_hp: float):
  # Atualiza barra de vida sobre o personagem
  if has_node("HealthBar"):
    $HealthBar.value = (hp / max_hp) * 100.0
```

---

2. Sincronizar Objetos Coletáveis

NetworkManager.gd:

gdscript

```
func send_item_collected(item_id: int, collector_id: int):
  if not is_connected:
    return
  rpc_id(1, "_server_item_collected", item_id, collector_id)

@rpc("any_peer", "reliable")
func _server_item_collected(item_id: int, collector_id: int):
  if not multiplayer.is_server():
    return

  var sender_id = multiplayer.get_remote_sender_id()
  if sender_id != collector_id:
    return

  # Valida coleta no servidor
  if not ServerManager.is_item_available(item_id):
    return  # Item já foi coletado

  # Marca como coletado
  ServerManager.mark_item_collected(item_id, collector_id)

  # Adiciona score
  RoundRegistry.add_score(collector_id, 10)

  # Notifica TODOS os clientes
  var room = RoomRegistry.get_room_by_player(collector_id)
  if room.is_empty():
    return

  for player in room["players"]:
    rpc_id(player["id"], "_client_item_collected", item_id, collector_id)

@rpc("authority", "reliable")
func _client_item_collected(item_id: int, collector_id: int):
  if multiplayer.is_server():
    return

  # Remove item da cena
  var item = get_tree().root.get_node_or_null("Items/Item_%d" % item_id)
  if item:
    item.queue_free()
  # Toca efeito sonoro
  AudioManager.play_sound("item_collected")
```

CollectableItem.gd:

gdscript

```
extends Area3D

@export var item_id: int = 0
@export var points: int = 10

func _ready():
  body_entered.connect(_on_body_entered)

func _on_body_entered(body: Node3D):
  # Verifica se é um player local
  if not body.is_in_group("local_player"):
    return

  var player = body as CharacterBody3D
  if not player:
    return

  # Coleta o item
  collect(player.player_id)

func collect(collector_id: int):
  # Desabilita para não coletar duas vezes
  set_deferred("monitoring", false)

  # Envia ao servidor
  NetworkManager.send_item_collected(item_id, collector_id)

  # Efeito visual local
  play_collect_effect()
```

---

3. Sincronizar Ações/Habilidades

NetworkManager.gd:

gdscript

```
func send_player_action(p_id: int, action_name: String, action_data: Dictionary):
  if not is_connected:
    return
  rpc_id(1, "_server_player_action", p_id, action_name, action_data)

@rpc("any_peer", "reliable")
func _server_player_action(p_id: int, action_name: String, action_data: Dictionary):
  if not multiplayer.is_server():
    return

  var sender_id = multiplayer.get_remote_sender_id()
  if sender_id != p_id:
    return

  # Valida ação no servidor
  if not ServerManager.validate_action(p_id, action_name, action_data):
    return

  # Processa efeitos da ação
  ServerManager.process_action(p_id, action_name, action_data)

  # Propaga
  var room = RoomRegistry.get_room_by_player(p_id)
  if room.is_empty():
    return

  for player in room["players"]:
    rpc_id(player["id"], "_client_player_action", p_id, action_name, action_data)

@rpc("authority", "reliable")
func _client_player_action(p_id: int, action_name: String, action_data: Dictionary):
  if multiplayer.is_server():
    return

  var player = get_tree().root.get_node_or_null(str(p_id))
  if not player:
    return

  # Executa ação visual
  match action_name:
    "shoot":
      player.play_shoot_animation()
      player.spawn_projectile(action_data["direction"], action_data["position"])
    "dash":
      player.play_dash_effect(action_data["direction"])
    "emote":
      player.play_emote(action_data["emote_id"])
```

player.gd:

gdscript

```
func perform_action(action_name: String, data: Dictionary = {}):
  if not is_local_player:
    return

  # Executa localmente
  _execute_action(action_name, data)

  # Sincroniza
  NetworkManager.send_player_action(player_id, action_name, data)

func _execute_action(action_name: String, data: Dictionary):
  match action_name:
    "shoot":
      play_shoot_animation()
      spawn_projectile(data["direction"], data["position"])
    "dash":
      apply_dash_impulse(data["direction"])
      play_dash_effect(data["direction"])
```

---

4. Sincronizar Estado do Mundo (Portas, Interruptores)

NetworkManager.gd:

gdscript

```
func send_world_object_state(object_id: int, new_state: String, data: Dictionary = {}):
  if not is_connected:
    return
  rpc_id(1, "_server_world_object_state", object_id, new_state, data)

@rpc("any_peer", "reliable")
func _server_world_object_state(object_id: int, new_state: String, data: Dictionary):
  if not multiplayer.is_server():
    return

  # Atualiza estado no servidor
  ServerManager.set_world_object_state(object_id, new_state, data)

  # Propaga para TODOS na rodada
  var round_data = RoundRegistry.get_current_round()
  if round_data.is_empty():
    return

  for player in round_data["players"]:
    rpc_id(player["id"], "_client_world_object_state", object_id, new_state, data)

@rpc("authority", "reliable")
func _client_world_object_state(object_id: int, new_state: String, data: Dictionary):
  if multiplayer.is_server():
    return

  # Encontra objeto no mundo
  var obj = get_tree().root.get_node_or_null("World/Objects/Object_%d" % object_id)
  if obj and obj.has_method("set_state"):
    obj.set_state(new_state, data)
```

Door.gd (exemplo):

gdscript

```
extends Node3D

@export var object_id: int = 0
var is_open: bool = false

func interact(interactor_id: int):
  # Só o player local pode interagir
  var player = get_tree().get_first_node_in_group("local_player")
  if not player or player.player_id != interactor_id:
    return

  # Toggle estado
  var new_state = "open" if not is_open else "closed"

  # Aplica localmente
  set_state(new_state, {})

  # Sincroniza
  NetworkManager.send_world_object_state(object_id, new_state, {})

func set_state(state: String, data: Dictionary):
  match state:
    "open":
      is_open = true
      $AnimationPlayer.play("open")
      $CollisionShape3D.disabled = true
    "closed":
      is_open = false
      $AnimationPlayer.play("close")
      $CollisionShape3D.disabled = false
```

📊 Sistema de Score Sincronizado

Já existe no RoundRegistry! Veja como usar:

gdscript

```
# Adicionar pontos (qualquer lugar do código)
RoundRegistry.add_score(player_id, 100)

# Definir score direto
RoundRegistry.set_score(player_id, 500)

# Obter score
var score = RoundRegistry.get_score(player_id)

# Obter todos scores
var all_scores = RoundRegistry.get_scores()  # {player_id: score}

# O vencedor é calculado automaticamente ao finalizar rodada
```

Exemplo: Sistema de Kills

ServerManager.gd:

gdscript

```gdscript
func register_kill(killer_id: int, victim_id: int):
	"""Registra um kill e atualiza scores"""
	
	# Adiciona pontos ao killer
	RoundRegistry.add_score(killer_id, 100)
	
	# Remove pontos da vítima (opcional)
	RoundRegistry.add_score(victim_id, -50)
	
	# Notifica todos
	var room = RoomRegistry.get_room_by_player(killer_id)
	if room.is_empty():
		return
	
	for player in room["players"]:
		NetworkManager.rpc_id(player["id"], "_client_kill_notification", killer_id, victim_id)
```

🎮 Tipos de RPC e Quando Usar

`@rpc("any_peer", "unreliable")`

Uso: Estados de posição/movimento (alta frequência)

- ✅ Rápido, baixa latência
- ❌ Pode perder pacotes
- 📦 20-30 vezes por segundo

`@rpc("any_peer", "reliable")`

Uso: Eventos importantes (coleta, dano, ações)

- ✅ Garantido chegar
- ⚠️ Pode ter delay
- 📦 Quando necessário

`@rpc("authority", "reliable")`

Uso: Comandos do servidor para clientes

- ✅ Autoritativo
- ✅ Garantido
- 📦 Spawn, morte, fim de rodada

🔒 Validação no Servidor (Anti-Cheat)

SEMPRE valide ações críticas no servidor:

📋 ✓

gdscript

```
@rpc("any_peer", "reliable")
func _server_player_shoot(p_id: int, target_pos: Vector3):
  if not multiplayer.is_server():
    return

  var sender_id = multiplayer.get_remote_sender_id()

  # 1. Valida sender
  if sender_id != p_id:
    _kick_player(sender_id, "Tentativa de spoofing")
    return

  # 2. Valida se player existe
  if not PlayerRegistry.is_player_registered(p_id):
    return

  # 3. Valida se está na rodada
  if not RoundRegistry.is_round_active():
    return

  # 4. Valida cooldown
  if not ServerManager.can_player_shoot(p_id):
    push_warning("Player %d tentou atirar antes do cooldown" % p_id)
    return

  # 5. Valida distância (anti-cheat)
  var player_pos = ServerManager.get_player_position(p_id)
  var distance = player_pos.distance_to(target_pos)

  if distance > 100.0:  # Máximo de alcance
    push_warning("Player %d tentou atirar muito longe" % p_id)
    return

  # ✅ Validação passou - processa tiro
  ServerManager.process_shot(p_id, target_pos)

  # Propaga para clientes
  _propagate_shot(p_id, target_pos)
```

📝 Checklist de Sincronização

Para sincronizar qualquer coisa nova:

- [ ] **1. Criar RPC no NetworkManager.gd**
  - [ ] `send_X()` - Cliente → Servidor
  - [ ] `_server_X()` - Servidor recebe
  - [ ] `_client_X()` - Clientes recebem
- [ ] **2. Validar no servidor**
  - [ ] Verificar sender_id
  - [ ] Validar dados
  - [ ] Aplicar lógica autoritativa
- [ ] **3. Propagar para clientes**
  - [ ] Encontrar room/rodada
  - [ ] Loop nos players
  - [ ] Enviar via `rpc_id()`
- [ ] **4. Aplicar nos clientes**
  - [ ] Encontrar objeto
  - [ ] Atualizar visual/estado
  - [ ] Tocar efeitos
- [ ] **5. Testar**
  - [ ] Funciona com 2 clientes?
  - [ ] Sincroniza corretamente?
  - [ ] Sem lag visível?

🚀 Template Rápido

Copie e adapte este template para qualquer sincronização:

gdscript

```gdscript
# ===== NetworkManager.gd =====

# Cliente envia
func send_custom_event(p_id: int, event_data: Dictionary):
  if not is_connected:
    return
  rpc_id(1, "_server_custom_event", p_id, event_data)

# Servidor processa
@rpc("any_peer", "reliable")
func _server_custom_event(p_id: int, event_data: Dictionary):
  if not multiplayer.is_server():
    return

  var sender_id = multiplayer.get_remote_sender_id()
  if sender_id != p_id:
    return

  # Validação aqui
  # ...

  # Propagação
  var room = RoomRegistry.get_room_by_player(p_id)
  if room.is_empty():
    return

  for player in room["players"]:
    rpc_id(player["id"], "_client_custom_event", p_id, event_data)

# Cliente recebe
@rpc("authority", "reliable")
func _client_custom_event(p_id: int, event_data: Dictionary):
  if multiplayer.is_server():
    return

  # Aplicar efeito visual/sonoro
  # ...
```

Use este template e substitua "custom_event" pelo que você quer sincronizar!