# Sistema de Rodadas - Guia de Uso

## 📋 Resumo do Sistema

O jogo funciona em ciclos de **Sala → Rodada → Sala → Rodada...**:

1. Players se conectam ao servidor
2. Criam ou entram em salas (lobbies)
3. O host inicia uma rodada
4. Ao fim da rodada, todos voltam à sala
5. O ciclo pode se repetir infinitamente

## 🔄 Ciclo de Vida de uma Rodada

```
1. ServerManager._handle_start_round()
   ↓ Cria rodada via RoundRegistry.create_round()

2. ServerManager._server_instantiate_round()
   ↓ Spawna mapa e players

3. RoundRegistry.start_round()
   ↓ Rodada em andamento (GAMEPLAY)

4. [Evento do jogo chama fim de rodada]
   ↓ ServerManager.end_current_round()

5. ServerManager._complete_round_end()
   ↓ Adiciona ao histórico da sala
   ↓ Limpa objetos
   ↓ Notifica clientes: _client_return_to_room()

6. Players voltam à sala
```

## 🎮 Como Finalizar uma Rodada

### No Servidor (Script de Gameplay)

```gdscript
```

```gdscript
extends Node

# Exemplo: Fim de rodada quando um player atinge 100 pontos

func _process(_delta):
    if not multiplayer.is_server():
        return

    if not RoundRegistry.is_round_active():
        return

    # Verifica scores
    var scores = RoundRegistry.get_scores()
    for peer_id in scores:
        if scores[peer_id] >= 100:
            # Encontra dados do vencedor
            var winner_data = {
                "peer_id": peer_id,
                "name": _get_player_name(peer_id),
                "score": scores[peer_id]
            }

            # Finaliza a rodada
            ServerManager.end_current_round("victory", winner_data)
            return

func _get_player_name(peer_id: int) -> String:
    var player = PlayerRegistry.get_player(peer_id)
    return player.get("name", "Desconhecido")
```

## Exemplo: Timer de Rodada

```gdscript
```

```gdscript
extends Node
## GameplayManager - Controla lógica da rodada

@export var round_time_limit: float = 300.0  # 5 minutos

var round_timer: Timer

func _ready():
    if not multiplayer.is_server():
        return

    # Conecta sinal de início de rodada
    RoundRegistry.round_started.connect(_on_round_started)

func _on_round_started(round_id: int):
    _log_debug("Rodada %d iniciada, configurando timer" % round_id)

    # Cria timer
    round_timer = Timer.new()
    round_timer.wait_time = round_time_limit
    round_timer.one_shot = true
    round_timer.timeout.connect(_on_time_limit_reached)
    add_child(round_timer)
    round_timer.start()

func _on_time_limit_reached():
    _log_debug("Tempo limite atingido!")

    # Finaliza rodada por tempo
    ServerManager.end_current_round("time_limit")

    round_timer.queue_free()
    round_timer = null
```

## Exemplo: Último Player Vivo

gdscript

```gdscript
extends Node
## SurvivalManager - Battle Royale / Survival

func _process(_delta):
    if not multiplayer.is_server():
        return

    if not RoundRegistry.is_round_active():
        return

    # Conta players vivos
    var alive_count = 0
    var last_alive = null

    for player in RoundRegistry.get_all_spawned_players():
        if player and player.is_alive:
            alive_count += 1
            last_alive = player

    # Se só sobrou um, ele venceu
    if alive_count == 1 and last_alive:
        var winner_data = {
            "peer_id": last_alive.player_id,
            "name": last_alive.player_name,
            "score": RoundRegistry.get_score(last_alive.player_id)
        }

        ServerManager.end_current_round("last_alive", winner_data)
```

## Exemplo: Sistema de Objetivos

gdscript

```gdscript
extends Node3D
## ObjectiveZone - Zona de captura

@export var capture_time: float = 10.0
@export var points_on_capture: int = 50

var capturing_team: int = -1
var capture_progress: float = 0.0

func _process(delta):
    if not multiplayer.is_server():
        return

    # Lógica de captura
    if capturing_team >= 0:
        capture_progress += delta

        if capture_progress >= capture_time:
            _on_captured()

func _on_captured():
    _log_debug("Objetivo capturado pelo time %d" % capturing_team)

    # Adiciona pontos ao time vencedor
    for player in RoundRegistry.get_all_spawned_players():
        if player.team == capturing_team:
            RoundRegistry.add_score(player.player_id, points_on_capture)

    # Verifica se algum time atingiu pontos necessários
    var scores = RoundRegistry.get_scores()
    for peer_id in scores:
        if scores[peer_id] >= 100:
            var winner_data = {
                "peer_id": peer_id,
                "name": PlayerRegistry.get_player(peer_id)["name"],
                "score": scores[peer_id]
            }

            ServerManager.end_current_round("objective_complete", winner_data)
            break
```

## 🔧 Finalizações Automáticas

O sistema já possui finalizações automáticas:

### 1. Todos Desconectaram

gdscript

```
# Automático via RoundRegistry
# Verifica a cada 2 segundos (configurável)
# Chama ServerManager.end_current_round("all_disconnected")
```

### 2. Timeout de Duração

gdscript

```
# Configure no Inspector do RoundRegistry
max_round_duration = 600.0  # 10 minutos
# Finaliza automaticamente quando o tempo acaba
```

## 📊 Adicionando Pontos Durante a Rodada

gdscript

```
# Em qualquer script de gameplay no servidor:

func on_player_scored(peer_id: int, points: int):
    if multiplayer.is_server():
        RoundRegistry.add_score(peer_id, points)

        # Notifica todos os clientes (opcional)
        rpc("_update_scoreboard", RoundRegistry.get_scores())

@rpc("authority", "call_remote", "reliable")
func _update_scoreboard(scores: Dictionary):
    # Atualiza UI do scoreboard
    pass
```

## 🎯 Exemplo Completo: Modo Corrida

gdscript

```gdscript
extends Node3D
## FinishLine - Linha de chegada

func _ready():
    # Conecta sinal de área
    $Area3D.body_entered.connect(_on_body_entered)

func _on_body_entered(body: Node3D):
    if not multiplayer.is_server():
        return

    if not body.is_in_group("player"):
        return

    if not RoundRegistry.is_round_active():
        return

    var player = body

    # Player cruzou a linha de chegada!
    var winner_data = {
        "peer_id": player.player_id,
        "name": player.player_name,
        "score": 1000  # Score fixo por vencer
    }

    _log_debug("Player %s venceu a corrida!" % player.player_name)

    # Finaliza a rodada
    ServerManager.end_current_round("race_finished", winner_data)

func _log_debug(msg: String):
    print("[FinishLine] " + msg)
```

## 📝 Histórico de Rodadas

O histórico fica armazenado na sala:

```gdscript
```

```
# Obter histórico de rodadas da sala
var history = RoomRegistry.get_rounds_history(room_id)

for round_data in history:
    print("Rodada #%d" % round_data["round_id"])
    print("  Vencedor: %s" % round_data["winner"]["name"])
    print("  Duração: %.1fs" % round_data["duration"])
    print("  Razão: %s" % round_data["end_reason"])
```

## ⚙️ Configurações Importantes

### No Inspector do RoundRegistry:

- `max_round_duration`: Tempo máximo (0 = ilimitado)
- `disconnect_check_interval`: Frequência de verificação
- `auto_end_on_disconnect`: Finalizar se todos desconectarem

### No Inspector do RoomRegistry:

- `max_players_per_room`: Máximo de players por sala
- `min_players_to_start`: Mínimo para iniciar rodada

### No Inspector do ServerManager:

- `round_transition_time`: Tempo entre fim e volta à sala
- `round_preparation_time`: Tempo antes de próxima rodada

## 🐛 Debug

Para ver logs detalhados:

- Habilite `debug_mode = true` em todos os scripts
- Os logs mostram todo o ciclo de vida das rodadas

```
[ServerManager] HOST INICIANDO RODADA
[RoundRegistry] [SERVER] ✓ Rodada criada: ID 1, Sala 'Minha Sala', 4 players
[RoundRegistry] [SERVER] ▶ Rodada 1 INICIADA
[RoundRegistry] [SERVER] ■ Rodada 1 FINALIZANDO | Razão: victory | Duração: 123.4s
[ServerManager] ✓ Rodada 1 FINALIZADA | Vencedor: João
```

## 🎮 Fluxo Completo no Cliente

gdscript

```
# O cliente recebe os seguintes callbacks:

GameManager._client_round_started()     # Rodada iniciou
  ↓ Spawna mapa e players localmente
  ↓ Esconde menu

GameManager._client_round_ended()       # Rodada terminou
  ↓ Mostra tela de resultados
  ↓ Aguarda alguns segundos

GameManager._client_return_to_room()    # Volta à sala
  ↓ Limpa objetos
  ↓ Mostra menu da sala novamente
```

---

## 🚀 Próximos Passos

Após integrar este sistema:

1. Crie scripts de gameplay que chamem `ServerManager.end_current_round()`
2. Implemente lógica de vitória/derrota específica do seu jogo
3. Crie UI para mostrar resultados da rodada
4. Adicione sistema de pontuação/ranking persistente
5. Implemente diferentes modos de jogo com condições de vitória variadas