# Archivos (parte) de: C:\MiChinchonWeb

## Archivo: C:/MiChinchonWeb/scripts/autoload/audio_manager.gd

```
extends Node
# audio_manager.gd
# Gestor global de audio para el juego Chinchón
# Se encarga de reproducir música y efectos de sonido

# Buses de audio
const MASTER_BUS = "Master"
const MUSIC_BUS = "Music"
const SFX_BUS = "SFX"

# Rutas de recursos
const MUSIC_PATH = "res://assets/audio/music/"
const SFX_PATH = "res://assets/audio/sfx/"

# Nodos para reproducción de audio
var music_players: Array = []
var sfx_players: Array = []
var current_music_player: int = 0
var current_sfx_player: int = 0

# Configuración
var music_volume: float = 0.8
var sfx_volume: float = 1.0
var music_enabled: bool = true
var sfx_enabled: bool = true
var music_bus_idx: int = -1
var sfx_bus_idx: int = -1

# Precargar efectos de sonido comunes
var preloaded_sfx = {
□"card_draw": null,
□"card_discard": null,
□"card_shuffle": null,
□"button_click": null,
□"chinchon": null,
□"game_over": null,
□"game_start": null
}

# Función de inicialización
func _ready() -> void:
□# Obtener índices de buses de audio
□music_bus_idx = AudioServer.get_bus_index(MUSIC_BUS)
□sfx_bus_idx = AudioServer.get_bus_index(SFX_BUS)
□
□# Si los buses no existen, crearlos
□if music_bus_idx == -1:
□□music_bus_idx = AudioServer.bus_count
□□AudioServer.add_bus()
□□AudioServer.set_bus_name(music_bus_idx, MUSIC_BUS)
□□AudioServer.set_bus_send(music_bus_idx, MASTER_BUS)
□
□if sfx_bus_idx == -1:
□□sfx_bus_idx = AudioServer.bus_count
□□AudioServer.add_bus()
□□AudioServer.set_bus_name(sfx_bus_idx, SFX_BUS)
□□AudioServer.set_bus_send(sfx_bus_idx, MASTER_BUS)
□
□# Crear reproductores de música (para crossfade)
□for i in range(2):
□□var music_player = AudioStreamPlayer.new()
□□music_player.bus = MUSIC_BUS
□□music_player.volume_db = linear_to_db(music_volume)
□□music_player.name = "MusicPlayer" + str(i)
□□add_child(music_player)
□□music_players.append(music_player)
□
□# Crear pool de reproductores de efectos de sonido
□for i in range(8):  # 8 reproductores para permitir múltiples efectos simultáneos
□□var sfx_player = AudioStreamPlayer.new()
□□sfx_player.bus = SFX_BUS
□□sfx_player.volume_db = linear_to_db(sfx_volume)
□□sfx_player.name = "SFXPlayer" + str(i)
```

```
□□add_child(sfx_player)
□□sfx_players.append(sfx_player)
□
□# Precargar efectos de sonido comunes
□preload_common_sfx()
□
□# Cargar configuración
□load_settings()
□
□# Configurar volumen inicial
□set_music_volume(music_volume)
□set_sfx_volume(sfx_volume)

# Precargar efectos de sonido comunes
func preload_common_sfx() -> void:
□for sfx_name in preloaded_sfx.keys():
□□var sfx_resource = load(SFX_PATH + sfx_name + ".ogg")
□□if sfx_resource:
□□□preloaded_sfx[sfx_name] = sfx_resource
□□else:
□□□push_warning("No se pudo cargar el efecto de sonido: " + sfx_name)

# Reproducir música
func play_music(music_name: String, fade_duration: float = 1.0, loop: bool = true) -> void:
□if !music_enabled:
□□return
□
□var music_resource = load(MUSIC_PATH + music_name + ".ogg")
□if !music_resource:
□□push_warning("No se pudo cargar la música: " + music_name)
□□return
□
□# Obtener el siguiente reproductor (para crossfade)
□var next_player_idx = (current_music_player + 1) % music_players.size()
□var current_player = music_players[current_music_player]
□var next_player = music_players[next_player_idx]
□
□# Configurar el siguiente reproductor
□next_player.stream = music_resource
□next_player.volume_db = linear_to_db(0.0)  # Empezar en silencio
□next_player.loop = loop
□next_player.play()
□
□# Crear efecto de crossfade
□var fade_out_tween = create_tween()
□fade_out_tween.tween_property(current_player, "volume_db", linear_to_db(0.0), fade_duration)
□fade_out_tween.tween_callback(func(): current_player.stop())
□
□var fade_in_tween = create_tween()
□fade_in_tween.tween_property(next_player, "volume_db", linear_to_db(music_volume), fade_duration)
□
□# Actualizar el reproductor actual
□current_music_player = next_player_idx

# Detener música
func stop_music(fade_duration: float = 1.0) -> void:
□var current_player = music_players[current_music_player]
□
□if current_player.playing:
□□var tween = create_tween()
□□tween.tween_property(current_player, "volume_db", linear_to_db(0.0), fade_duration)
□□tween.tween_callback(func(): current_player.stop())

# Reproducir efecto de sonido
func play_sfx(sfx_name: String, volume: float = 1.0, pitch: float = 1.0) -> void:
□if !sfx_enabled:
□□return
□
□# Verificar si está precargado
□var sfx_resource = preloaded_sfx.get(sfx_name)
□
□# Si no está precargado, intentar cargarlo
□if !sfx_resource:
□□sfx_resource = load(SFX_PATH + sfx_name + ".ogg")
□
□if !sfx_resource:
□□push_warning("No se pudo cargar el efecto de sonido: " + sfx_name)
□□return
□
□# Buscar un reproductor de efectos disponible
□var found_player = false
```

```
	var start_idx = current_sfx_player
	
	for i in range(sfx_players.size()):
		var idx = (start_idx + i) % sfx_players.size()
		var player = sfx_players[idx]
		
		if !player.playing:
			player.stream = sfx_resource
			player.volume_db = linear_to_db(sfx_volume * volume)
			player.pitch_scale = pitch
			player.play()
			
			current_sfx_player = (idx + 1) % sfx_players.size()
			found_player = true
			break
	
	# Si todos los reproductores están ocupados, usar el siguiente en la lista
	if !found_player:
		var player = sfx_players[current_sfx_player]
		player.stream = sfx_resource
		player.volume_db = linear_to_db(sfx_volume * volume)
		player.pitch_scale = pitch
		player.play()
		
		current_sfx_player = (current_sfx_player + 1) % sfx_players.size()

# Establecer volumen de música
func set_music_volume(volume: float) -> void:
	music_volume = clamp(volume, 0.0, 1.0)
	
	if music_bus_idx >= 0:
		AudioServer.set_bus_volume_db(music_bus_idx, linear_to_db(music_volume))
	
	# Actualizar reproductores de música activos
	for player in music_players:
		if player.playing:
			player.volume_db = linear_to_db(music_volume)
	
	save_settings()

# Establecer volumen de efectos de sonido
func set_sfx_volume(volume: float) -> void:
	sfx_volume = clamp(volume, 0.0, 1.0)
	
	if sfx_bus_idx >= 0:
		AudioServer.set_bus_volume_db(sfx_bus_idx, linear_to_db(sfx_volume))
	
	save_settings()

# Activar/desactivar música
func toggle_music(enabled: bool) -> void:
	music_enabled = enabled
	
	if music_bus_idx >= 0:
		AudioServer.set_bus_mute(music_bus_idx, !music_enabled)
	
	if !music_enabled:
		for player in music_players:
			player.stop()
	
	save_settings()

# Activar/desactivar efectos de sonido
func toggle_sfx(enabled: bool) -> void:
	sfx_enabled = enabled
	
	if sfx_bus_idx >= 0:
		AudioServer.set_bus_mute(sfx_bus_idx, !sfx_enabled)
	
	save_settings()

# Guardar configuración de audio
func save_settings() -> void:
	var config = ConfigFile.new()
	
	config.set_value("audio", "music_volume", music_volume)
	config.set_value("audio", "sfx_volume", sfx_volume)
	config.set_value("audio", "music_enabled", music_enabled)
	config.set_value("audio", "sfx_enabled", sfx_enabled)
	
	var err = config.save("user://audio_settings.cfg")
```

```
□if err != OK:
□□push_error("Error al guardar configuración de audio: " + str(err))

# Cargar configuración de audio
func load_settings() -> void:
□var config = ConfigFile.new()
□var err = config.load("user://audio_settings.cfg")
□
□if err == OK:
□□music_volume = config.get_value("audio", "music_volume", 0.8)
□□sfx_volume = config.get_value("audio", "sfx_volume", 1.0)
□□music_enabled = config.get_value("audio", "music_enabled", true)
□□sfx_enabled = config.get_value("audio", "sfx_enabled", true)
□
□# Aplicar configuración
□if music_bus_idx >= 0:
□□AudioServer.set_bus_volume_db(music_bus_idx, linear_to_db(music_volume))
□□AudioServer.set_bus_mute(music_bus_idx, !music_enabled)
□
□if sfx_bus_idx >= 0:
□□AudioServer.set_bus_volume_db(sfx_bus_idx, linear_to_db(sfx_volume))
□□AudioServer.set_bus_mute(sfx_bus_idx, !sfx_enabled)
```

## Archivo: C:/MiChinchonWeb/scripts/autoload/audio_manager.gd.uid

```
uid://nku3avcuyhr2
```

# Archivo: C:/MiChinchonWeb/scripts/autoload/game_manager.gd

```
extends Node
# GameManager.gd
# Script singleton (autoload) que gestiona el estado global del juego Chinchón
# Controla la lógica del juego, reglas, puntuación y estado de la partida

# Señales
signal game_started              # Emitida cuando comienza una nueva partida
signal round_started(round_num)   # Emitida cuando comienza una nueva ronda
signal player_turn_started(player_id) # Emitida cuando comienza el turno de un jugador
signal player_turn_ended(player_id)   # Emitida cuando finaliza el turno de un jugador
signal card_drawn(player_id, from_pile) # Emitida cuando un jugador roba una carta
signal card_discarded(player_id, card)  # Emitida cuando un jugador descarta una carta
signal player_chinchon(player_id)  # Emitida cuando un jugador consigue Chinchón
signal player_closed(player_id)   # Emitida cuando un jugador cierra el juego
signal round_ended(scores)        # Emitida cuando finaliza una ronda
signal game_ended(final_scores)   # Emitida cuando finaliza la partida
signal game_paused               # Emitida cuando se pausa el juego
signal game_resumed               # Emitida cuando se reanuda el juego

# Enumeraciones
enum GameState {IDLE, DEALING, PLAYER_TURN, AI_TURN, ROUND_END, GAME_END}
enum DrawSource {DECK, DISCARD_PILE}
enum CardValue {AS = 1, DOS = 2, TRES = 3, CUATRO = 4, CINCO = 5, SEIS = 6, SIETE = 7, SOTA = 8, CABALLO = 9, REY = 10}
enum CardSuit {OROS, COPAS, ESPADAS, BASTOS}

# Constantes
const MAX_PLAYERS = 8          # Máximo número de jugadores
const MIN_PLAYERS = 2          # Mínimo número de jugadores
const CARDS_PER_PLAYER = 7     # Cartas por jugador
const MAX_ROUNDS = 5           # Número de rondas en una partida
const CHINCHON_BONUS = 10      # Puntos de bonificación por conseguir Chinchón
const MAX_SCORE = 100          # Puntuación máxima para eliminación
const GAME_VERSION = "1.0.0"   # Versión del juego

# Variables de estado del juego
var game_state = GameState.IDLE  # Estado actual del juego
var current_player_index = 0     # Índice del jugador actual
var players = []                 # Array de jugadores
var deck = []                    # Baraja de cartas
var discard_pile = []            # Pila de descarte
var current_round = 0            # Ronda actual
var is_game_active = false       # ¿Está el juego activo?
var is_game_paused = false       # ¿Está el juego pausado?
var use_48_card_deck = false     # True para baraja de 48 cartas, false para 40 cartas
var two_deck_mode = false        # True si se juega con dos barajas

# Variables de configuración
var game_mode = "singleplayer"  # "singleplayer" o "multiplayer"
var language = "es"             # Idioma predeterminado
var ai_difficulty = 1           # Dificultad de la IA (1-3)

# Funciones de inicialización
func _ready():
	randomize()  # Inicializar generador de números aleatorios
	load_config()
	print("GameManager inicializado - Versión: ", GAME_VERSION)

# Iniciar una nueva partida
func start_new_game(player_count: int, player_names: Array, use_two_decks: bool = false, use_48_cards: bool = false) -> void:
	if player_count < MIN_PLAYERS or player_count > MAX_PLAYERS:
		push_error("Número de jugadores inválido: " + str(player_count))
		return
	
	# Configurar opciones de juego
	two_deck_mode = use_two_decks
	use_48_card_deck = use_48_cards
	
	# Restablecer estado del juego
	is_game_active = true
	is_game_paused = false
	current_round = 1
	game_state = GameState.IDLE
	
	# Inicializar jugadores
	initialize_players(player_count, player_names)
	
	# Comenzar primera ronda
	start_round()
```

```
□# Señalizar inicio de juego
□emit_signal("game_started")
□print("Juego iniciado con " + str(player_count) + " jugadores")

# Inicializar jugadores
func initialize_players(count: int, names: Array) -> void:
□players.clear()
□
□for i in range(count):
□□var player_name = names[i] if i < names.size() else "Jugador " + str(i+1)
□□var is_ai = (i > 0) if game_mode == "singleplayer" else false
□□
□□players.append({
□□□"id": i,
□□□"name": player_name,
□□□"hand": [],
□□□"score": 0,
□□□"total_score": 0,
□□□"is_ai": is_ai,
□□□"has_drawn": false,
□□□"eliminated": false
□□})
□
□current_player_index = 0  # El primer jugador comienza

# Iniciar una nueva ronda
func start_round() -> void:
□game_state = GameState.DEALING
□
□# Inicializar baraja y repartir
□initialize_deck()
□shuffle_deck()
□deal_cards()
□
□# Comenzar con el juego
□game_state = GameState.PLAYER_TURN
□
□# Emitir señal de inicio de ronda
□emit_signal("round_started", current_round)
□start_player_turn()
□
□print("Ronda " + str(current_round) + " iniciada")

# Inicializar baraja de cartas
func initialize_deck() -> void:
□deck.clear()
□discard_pile.clear()
□
□var suits = [CardSuit.OROS, CardSuit.COPAS, CardSuit.ESPADAS, CardSuit.BASTOS]
□var min_value = 1
□var max_value = 12 if use_48_card_deck else 10  # 48 cartas (con 8,9) o 40 cartas
□
□# Crear baraja(s)
□var deck_count = 1
□if two_deck_mode:
□□deck_count = 2
□
□for _deck_num in range(deck_count):
□□for suit in suits:
□□□for value in range(min_value, max_value + 1):
□□□□# Ajustar valores para corresponder con la baraja española
□□□□var adjusted_value = value
□□□□if value > 7:
□□□□□adjusted_value = value + 2  # 8->10 (sota), 9->11 (caballo), 10->12 (rey)
□□□□
□□□□deck.append({
□□□□□"suit": suit,
□□□□□"value": adjusted_value,
□□□□□"score_value": get_card_score_value(adjusted_value)
□□□□})
□
□print("Baraja inicializada: " + str(deck.size()) + " cartas")

# Obtener el valor de puntuación de una carta
func get_card_score_value(card_value: int) -> int:
□if card_value == 1:  # As
□□return 1
□elif card_value == 10:  # Sota
□□return 8
□elif card_value == 11:  # Caballo
□□return 9
```

```
□elif card_value == 12:  # Rey
□□return 10
□else:
□□return card_value  # 2-7 valen lo mismo que su número

# Mezclar la baraja
func shuffle_deck() -> void:
□deck.shuffle()
□print("Baraja mezclada")

# Repartir cartas a los jugadores
func deal_cards() -> void:
□# Repartir cartas a cada jugador
□for player in players:
□□if player.eliminated:
□□□continue
□□□
□□player.hand.clear()
□□player.has_drawn = false
□□
□□for _i in range(CARDS_PER_PLAYER):
□□□if deck.size() > 0:
□□□□player.hand.append(deck.pop_back())
□
□# Colocar primera carta en la pila de descarte
□if deck.size() > 0:
□□discard_pile.append(deck.pop_back())
□
□print("Cartas repartidas")

# Comenzar turno del jugador actual
func start_player_turn() -> void:
□var current_player = players[current_player_index]
□
□if current_player.eliminated:
□□advance_to_next_player()
□□return
□
□current_player.has_drawn = false
□emit_signal("player_turn_started", current_player.id)
□
□if current_player.is_ai:
□□game_state = GameState.AI_TURN
□□# La lógica de la IA se implementará en otro script
□□# que llamará a process_ai_turn()
□else:
□□game_state = GameState.PLAYER_TURN
□
□print("Turno del jugador: " + current_player.name)

# Procesar turno de la IA
func process_ai_turn() -> void:
□# Esta función será llamada por la lógica de IA
□var current_player = players[current_player_index]
□
□if !current_player.is_ai:
□□push_error("process_ai_turn llamado para un jugador humano")
□□return
□
□# Implementar lógica de IA aquí o en otro script
□# Por ahora, solo pasamos el turno
□advance_to_next_player()

# Robar carta (del mazo o de la pila de descarte)
func draw_card(player_index: int, source: int) -> Dictionary:
□var player = players[player_index]
□
□if player.has_drawn:
□□push_error("El jugador ya ha robado una carta este turno")
□□return {}
□
□var card = {}
□
□if source == DrawSource.DECK:
□□# Verificar si hay cartas en el mazo
□□if deck.size() == 0:
□□□# Si no hay cartas, reciclar la pila de descarte
□□□recycle_discard_pile()
□□□
□□□if deck.size() == 0:
□□□□push_error("No hay cartas disponibles para robar")
```

```
        return {}
    
    card = deck.pop_back()
    print("Jugador " + player.name + " robó carta del mazo")
else:  # DrawSource.DISCARD_PILE
    if discard_pile.size() == 0:
        push_error("No hay cartas en la pila de descarte")
        return {}
    
    card = discard_pile.pop_back()
    print("Jugador " + player.name + " tomó carta de la pila de descarte")

player.hand.append(card)
player.has_drawn = true

emit_signal("card_drawn", player.id, source)
return card

# Reciclar pila de descarte como nuevo mazo
func recycle_discard_pile() -> void:
if discard_pile.size() <= 1:
    return

# Mantener la carta superior en la pila de descarte
var top_card = discard_pile.pop_back()

# Mover el resto al mazo y mezclar
deck = discard_pile.duplicate()
discard_pile.clear()
discard_pile.append(top_card)

shuffle_deck()
print("Pila de descarte reciclada como nuevo mazo")

# Descartar carta
func discard_card(player_index: int, card_index: int) -> Dictionary:
var player = players[player_index]

if !player.has_drawn:
    push_error("El jugador debe robar una carta antes de descartar")
    return {}

if card_index < 0 or card_index >= player.hand.size():
    push_error("Índice de carta inválido: " + str(card_index))
    return {}

var card = player.hand[card_index]
player.hand.remove_at(card_index)
discard_pile.append(card)

emit_signal("card_discarded", player.id, card)
print("Jugador " + player.name + " descartó una carta")

# Verificar si el jugador ganó la ronda
if check_for_round_win(player_index):
    end_round()
else:
    advance_to_next_player()

return card

# Verificar si un jugador ha ganado la ronda
func check_for_round_win(player_index: int) -> bool:
var player = players[player_index]

# El jugador debe tener exactamente 0 cartas para ganar
if player.hand.size() == 0:
    # El jugador ha cerrado el juego
    emit_signal("player_closed", player.id)
    print("¡Jugador " + player.name + " ha cerrado el juego!")
    return true

# Verificar si tiene Chinchón (todas las cartas combinadas)
var has_chinchon = check_for_chinchon(player.hand)
if has_chinchon:
    emit_signal("player_chinchon", player.id)
    print("¡Jugador " + player.name + " ha conseguido Chinchón!")
    return true

return false
# Verificar si una mano es Chinchón (todas las cartas en combinaciones)
```

```
func check_for_chinchon(hand: Array) -> bool:
    # Implementación simple - se expandirá en la versión completa
    # para verificar correctamente las combinaciones de cartas

    # Por ahora, consideramos que un jugador tiene Chinchón si tiene
    # exactamente 7 cartas (CARDS_PER_PLAYER) y puede formar grupos

    if hand.size() != CARDS_PER_PLAYER:
        return false

    # En la implementación real, verificaríamos si todas las cartas pueden
    # formar escaleras o grupos del mismo valor

    return false  # Por defecto, retornamos false hasta implementar la verificación completa

# Pasar al siguiente jugador
func advance_to_next_player() -> void:
    emit_signal("player_turn_ended", players[current_player_index].id)

    # Encontrar al siguiente jugador no eliminado
    var next_player_found = false
    var original_index = current_player_index

    while !next_player_found:
        current_player_index = (current_player_index + 1) % players.size()

        if !players[current_player_index].eliminated:
            next_player_found = true

        # Si hemos dado una vuelta completa, todos están eliminados
        if current_player_index == original_index:
            break

    start_player_turn()

# Finalizar la ronda actual
func end_round() -> void:
    game_state = GameState.ROUND_END

    # Calcular puntuaciones
    var round_scores = calculate_round_scores()

    # Actualizar puntuaciones totales y verificar eliminaciones
    for i in range(players.size()):
        players[i].total_score += players[i].score

        if players[i].total_score >= MAX_SCORE:
            players[i].eliminated = true
            print("Jugador " + players[i].name + " eliminado con " + str(players[i].total_score) + " puntos")

    emit_signal("round_ended", round_scores)
    print("Ronda " + str(current_round) + " finalizada")

    # Verificar si el juego ha terminado
    if current_round >= MAX_ROUNDS or check_game_end():
        end_game()
    else:
        current_round += 1
        start_round()

# Verificar si el juego ha terminado
func check_game_end() -> bool:
    # El juego termina si solo queda un jugador no eliminado
    var active_players = 0

    for player in players:
        if !player.eliminated:
            active_players += 1

    return active_players <= 1

# Calcular puntuaciones de la ronda
func calculate_round_scores() -> Dictionary:
    var scores = {}
    var winner_index = -1
    var chinchon_bonus = false

    # Encontrar al ganador (jugador que cerró)
    for i in range(players.size()):
        if players[i].hand.size() == 0 or check_for_chinchon(players[i].hand):
            winner_index = i
```

□□□chinchon_bonus = check_for_chinchon(players[i].hand)
□□□break
□
□# Calcular puntuación para cada jugador
□for i in range(players.size()):
□□var player = players[i]
□□var hand_value = 0
□□
□□# Sumar valores de las cartas en la mano
□□for card in player.hand:
□□□hand_value += card.score_value
□□
□□# El ganador recibe 0 puntos (o -CHINCHON_BONUS si hizo Chinchón)
□□if i == winner_index:
□□□player.score = -CHINCHON_BONUS if chinchon_bonus else 0
□□else:
□□□player.score = hand_value

□□
□□scores[i] = player.score
□
□return scores

# Finalizar el juego
func end_game() -> void:
□game_state = GameState.GAME_END
□is_game_active = false
□
□var final_scores = {}
□for i in range(players.size()):
□□final_scores[i] = players[i].total_score
□
□emit_signal("game_ended", final_scores)
□print("Juego finalizado")

# Pausar el juego
func pause_game() -> void:
□if is_game_active and !is_game_paused:
□□is_game_paused = true
□□emit_signal("game_paused")
□□print("Juego pausado")

# Reanudar el juego
func resume_game() -> void:
□if is_game_active and is_game_paused:
□□is_game_paused = false
□□emit_signal("game_resumed")
□□print("Juego reanudado")

# Guardar configuración
func save_config() -> void:
□var config = ConfigFile.new()
□config.set_value("settings", "language", language)
□config.set_value("settings", "game_mode", game_mode)
□config.set_value("settings", "ai_difficulty", ai_difficulty)
□
□var err = config.save("user://chinchon_config.cfg")
□if err != OK:
□□push_error("Error al guardar configuración: " + str(err))

# Cargar configuración
func load_config() -> void:
□var config = ConfigFile.new()
□var err = config.load("user://chinchon_config.cfg")
□
□if err == OK:
□□language = config.get_value("settings", "language", "es")
□□game_mode = config.get_value("settings", "game_mode", "singleplayer")
□□ai_difficulty = config.get_value("settings", "ai_difficulty", 1)
□else:
□□# Crear configuración predeterminada si no existe
□□save_config()

# Obtener el nombre formateado de un valor de carta
func get_card_value_name(value: int) -> String:
□match value:
□□1: return "As"
□□10: return "Sota"
□□11: return "Caballo"
□□12: return "Rey"
□□_: return str(value)

```
# Obtener el nombre de un palo
func get_suit_name(suit: int) -> String:
	match suit:
		CardSuit.OROS: return "Oros"
		CardSuit.COPAS: return "Copas"
		CardSuit.ESPADAS: return "Espadas"
		CardSuit.BASTOS: return "Bastos"
		_: return "Desconocido"

# Obtener una descripción de una carta
func get_card_description(card: Dictionary) -> String:
	return get_card_value_name(card.value) + " de " + get_suit_name(card.suit)
```

## Archivo: C:/MiChinchonWeb/scripts/autoload/game_manager.gd.uid

```
uid://dvlyv5cgwj2gj
```

## Archivo: C:/MiChinchonWeb/scripts/autoload/translation_manager.gd

```
extends Node
# translation_manager.gd
# Gestor de traducciones y localización para el juego Chinchón

# Señales
signal language_changed(locale)  # Emitida cuando se cambia el idioma

# Constantes
const TRANSLATIONS_PATH = "res://assets/i18n/"
const DEFAULT_LOCALE = "es"  # Español como idioma por defecto

# Idiomas soportados
var supported_locales = {
□"es": {
□□"name": "Español",
□□"flag": "es_flag"
□},
□"en": {
□□"name": "English",
□□"flag": "en_flag"
□}
□# Añadir más idiomas según sea necesario
}

# Estado
var current_locale: String = DEFAULT_LOCALE

# Función de inicialización
func _ready() -> void:
□# Cargar configuración de idioma guardada
□load_settings()
□
□# Establecer el idioma inicial
□set_locale(current_locale)
□
□# Cargar traducciones
□load_translations()

# Cargar archivos de traducción
func load_translations() -> void:
□# Buscar archivos de traducción en el directorio
□var translations_dir = DirAccess.open(TRANSLATIONS_PATH)
□
□if translations_dir:
□□translations_dir.list_dir_begin()
□□var file_name = translations_dir.get_next()
□□
□□while file_name != "":
□□□if !translations_dir.current_is_dir() and file_name.ends_with(".translation"):
□□□□var translation_path = TRANSLATIONS_PATH + file_name
□□□□var translation = load(translation_path)
□□□□
□□□□TranslationServer.add_translation(translation)
□□□□print("Traducción cargada desde: ", translation_path)

□□□
□□□file_name = translations_dir.get_next()
□□
□□translations_dir.list_dir_end()
□else:
□□push_error("No se pudo acceder al directorio de traducciones: " + TRANSLATIONS_PATH)

# Establecer el idioma
func set_locale(locale: String) -> void:
□if !supported_locales.has(locale):
□□push_warning("Idioma no soportado: " + locale + ". Usando el idioma por defecto.")
□□locale = DEFAULT_LOCALE
□
□current_locale = locale
□TranslationServer.set_locale(locale)
□save_settings()
□
□emit_signal("language_changed", locale)
□print("Idioma cambiado a: " + get_language_name(locale))

# Obtener el idioma actual
func get_current_locale() -> String:
□return current_locale
```

```
# Obtener el nombre del idioma
func get_language_name(locale: String) -> String:
	if supported_locales.has(locale):
		return supported_locales[locale].name
	return locale

# Obtener la ruta del icono de bandera
func get_flag_icon(locale: String) -> String:
	if supported_locales.has(locale):
		return "res://assets/images/ui/flags/" + supported_locales[locale].flag + ".png"
	return ""

# Obtener lista de idiomas soportados
func get_supported_locales() -> Array:
	var locales = []
	for locale in supported_locales.keys():
		locales.append(locale)
	return locales

# Obtener lista de nombres de idiomas soportados
func get_supported_language_names() -> Array:
	var names = []
	for locale in supported_locales.keys():
		names.append(supported_locales[locale].name)
	return names

# Traducir texto directamente (útil para textos dinámicos)
func translate(text_key: String) -> String:
	return tr(text_key)

# Guardar configuración
func save_settings() -> void:
	var config = ConfigFile.new()
	config.set_value("localization", "locale", current_locale)
	
	var err = config.save("user://language_settings.cfg")
	if err != OK:
		push_error("Error al guardar configuración de idioma: " + str(err))

# Cargar configuración
func load_settings() -> void:
	var config = ConfigFile.new()
	var err = config.load("user://language_settings.cfg")
	
	if err == OK:
		var saved_locale = config.get_value("localization", "locale", DEFAULT_LOCALE)
		if supported_locales.has(saved_locale):
			current_locale = saved_locale
	else:
		# Intentar detectar el idioma del sistema
		var system_locale = OS.get_locale().substr(0, 2).to_lower()
		if supported_locales.has(system_locale):
			current_locale = system_locale
		else:
			current_locale = DEFAULT_LOCALE

# Función para generar un archivo CSV de traducción base
func generate_translation_csv() -> void:
	if OS.is_debug_build():  # Solo disponible en modo debug
		var csv_content = "keys,es,en\n"  # Encabezado con idiomas
		
		# Añadir entradas del juego
		# Estas son solo ejemplos, en un juego real se extraerían de los textos del juego
		var translations = {
			"GAME_TITLE": ["Chinchón", "Chinchón"],
			"MENU_NEW_GAME": ["Nueva Partida", "New Game"],
			"MENU_SETTINGS": ["Configuración", "Settings"],
			"MENU_EXIT": ["Salir", "Exit"],
			"GAME_YOUR_TURN": ["Tu turno", "Your Turn"],
			"GAME_OPPONENT_TURN": ["Turno de %s", "%s's Turn"],
			"GAME_ROUND": ["Ronda %d", "Round %d"],
			"GAME_DRAW_CARD": ["Robar carta", "Draw Card"],
			"GAME_DISCARD": ["Descartar", "Discard"],
			"GAME_SORT_HAND": ["Ordenar mano", "Sort Hand"],
			"GAME_CHINCHON": ["¡Chinchón!", "Chinchón!"],
			"GAME_WINNER": ["¡%s gana!", "%s wins!"],
			"SETTINGS_MUSIC": ["Música", "Music"],
			"SETTINGS_SFX": ["Efectos", "SFX"],
			"SETTINGS_LANGUAGE": ["Idioma", "Language"],
			"DIALOG_CONFIRM": ["Aceptar", "OK"],
			"DIALOG_CANCEL": ["Cancelar", "Cancel"],
```

□□□"HELP_TITLE": ["Ayuda", "Help"],
□□□"HELP_RULES": ["Reglas del juego", "Game Rules"],
□□□# Reglas del juego - introducción
□□□"RULES_INTRO": ["El Chinchón es un juego de cartas en el que debes formar combinaciones antes que tus oponentes.",
□□□□□□□"Chinchón is a card game where you must form combinations before your opponents."],
□□□# Reglas del juego - objetivo
□□□"RULES_GOAL": ["El objetivo es formar grupos o escaleras con todas tus cartas y descartar la última.",
□□□□□□ "The goal is to form groups or runs with all your cards and discard the last one."],
□□□# Reglas - combinaciones
□□□"RULES_COMBINATIONS": ["Las combinaciones válidas son grupos de 3 o 4 cartas del mismo valor, o escaleras de 3 o más cartas consecutivas del misn
□□□□□□□□ "Valid combinations are groups of 3 or 4 cards of the same value, or runs of 3 or more consecutive cards of the same suit."],
□□□# Más entradas para todas las interfaces y mensajes del juego
□□}
□□
□□# Generar filas CSV
□□for key in translations.keys():
□□□csv_content += key + "," + translations[key][0] + "," + translations[key][1] + "\n"
□□
□□# Guardar archivo
□□var file = FileAccess.open(TRANSLATIONS_PATH + "translation_template.csv", FileAccess.WRITE)
□□if file:
□□□file.store_string(csv_content)
□□□print("Archivo de traducción generado en: " + TRANSLATIONS_PATH + "translation_template.csv")
□□else:
□□□push_error("No se pudo crear el archivo de traducción")

## Archivo: C:/MiChinchonWeb/scripts/autoload/translation_manager.gd.uid

```
uid://t7p7wqeyv1ae
```

# Archivo: C:/MiChinchonWeb/scripts/cards/card.gd

```
extends Node2D
# Card.gd
# Clase para representar una carta individual en el juego Chinchón
# Maneja la lógica visual e interactiva de las cartas

# Señales
signal card_clicked(card_node)  # Emitida cuando se hace clic en la carta
signal card_drag_started(card_node)  # Emitida cuando se comienza a arrastrar la carta
signal card_drag_ended(card_node)  # Emitida cuando se termina de arrastrar la carta

# Propiedades de la carta
var suit: int = -1  # Palo: oros, copas, espadas, bastos (valores de GameManager.CardSuit)
var value: int = -1  # Valor: 1(as) a 12(rey) (valores directos de carta española)
var score_value: int = 0  # Valor para puntuación
var card_id: String = ""  # Identificador único para la carta
var owner_id: int = -1  # ID del jugador propietario (-1 si está en el mazo/descarte)

# Estados visuales
var is_face_up: bool = true  # Si la carta muestra su cara (true) o su dorso (false)
var is_selected: bool = false  # Si la carta está seleccionada
var is_highlighted: bool = false  # Si la carta está resaltada (ej: posible jugada)
var is_draggable: bool = false  # Si la carta se puede arrastrar
var original_position: Vector2  # Posición original para retorno si se cancela arrastre
var original_z_index: int = 0  # Índice Z original

# Referencias a nodos
@onready var sprite: Sprite2D = $CardSprite
@onready var highlight: Sprite2D = $HighlightSprite
@onready var collision: CollisionShape2D = $CardCollision
@onready var animation_player: AnimationPlayer = $AnimationPlayer

# Constantes
const SELECTION_OFFSET: Vector2 = Vector2(0, -20)  # Desplazamiento cuando se selecciona
const HOVER_SCALE: Vector2 = Vector2(1.05, 1.05)  # Escala al pasar el cursor
const DRAG_Z_INDEX: int = 10  # Índice Z durante arrastre
const CARD_WIDTH: float = 140.0  # Ancho de la carta en píxeles
const CARD_HEIGHT: float = 190.0  # Alto de la carta en píxeles

# Variables para arrastre
var _dragging: bool = false
var _drag_offset: Vector2 = Vector2.ZERO
var _hover: bool = false

# Función de inicialización
func _ready() -> void:
    # Guardar posición original
    original_position = position
    original_z_index = z_index

    # Configurar estado visual inicial
    highlight.visible = false

    # Conectar señales de interacción
    var card_button = $CardButton
    card_button.connect("pressed", _on_card_pressed)
    card_button.connect("mouse_entered", _on_mouse_entered)
    card_button.connect("mouse_exited", _on_mouse_exited)

    # Generar ID único para la carta si no existe
    if card_id.is_empty():
        card_id = _generate_card_id()

    # Actualizar apariencia inicial
    _update_appearance()

# Configurar la carta con valores específicos
func setup(card_suit: int, card_value: int, show_face: bool = true) -> void:
    suit = card_suit
    value = card_value
    score_value = _calculate_score_value(card_value)
    is_face_up = show_face

    # Generar ID único para la carta
    card_id = _generate_card_id()

    # Actualizar visual
    _update_appearance()
# Generar un ID único para la carta
```

```
func _generate_card_id() -> String:
□return "card_" + str(suit) + "_" + str(value) + "_" + str(randi() % 10000)

# Calcular valor de puntuación para la carta
func _calculate_score_value(card_value: int) -> int:
□match card_value:
□□1: return 1  # As
□□10: return 8  # Sota
□□11: return 9  # Caballo
□□12: return 10  # Rey
□□_: return card_value  # 2-7 valen su número
□
# Actualizar apariencia de la carta según su estado
func _update_appearance() -> void:
□if is_face_up:
□□# Mostrar cara de la carta
□□_load_card_texture()
□else:
□□# Mostrar dorso
□□_load_back_texture()
□
□# Actualizar visibilidad del resaltado
□highlight.visible = is_highlighted or is_selected
□
□# Actualizar posición según selección
□if is_selected:
□□position = original_position + SELECTION_OFFSET
□else:
□□position = original_position

# Cargar textura correspondiente a la carta, usando el formato del repositorio de cartas españolas
func _load_card_texture() -> void:
□# Mapear valores de palo a nombres usados en el repositorio
□var suit_name = _get_suit_name(suit)
□
□# Mapear valores de carta a nombres usados en el repositorio
□var value_name = _get_value_file_name(value)
□
□# Construir la ruta del archivo según la estructura del repositorio
□# La estructura esperada es "baraja-española/{palo}/{valor}.png"
□var texture_path = "res://assets/sprites/cards/baraja-española/" + suit_name + "/" + value_name + ".png"
□
□var texture = load(texture_path)
□if texture:
□□sprite.texture = texture
□else:
□□# Si no encuentra la textura, intentar con una ruta alternativa (por si hay diferencias en nombres)
□□var alt_texture_path = "res://assets/sprites/cards/" + suit_name + "_" + value_name + ".png"
□□texture = load(alt_texture_path)
□□
□□if texture:
□□□sprite.texture = texture
□□else:
□□□# Textura de respaldo si no se encuentra la correcta
□□□push_warning("No se pudo cargar la textura: " + texture_path + " o " + alt_texture_path)
□□□_load_back_texture()

# Cargar textura del dorso de la carta
func _load_back_texture() -> void:
□# Ruta esperada del dorso según repositorio
□var back_texture = load("res://assets/sprites/cards/baraja-española/dorso.png")
□
□if not back_texture:
□□# Intentar ruta alternativa
□□back_texture = load("res://assets/sprites/cards/card_back.png")
□
□if back_texture:
□□sprite.texture = back_texture
□else:
□□push_warning("No se pudo cargar la textura del dorso de la carta")
□□# Crear un placeholder si no hay textura de dorso
□□sprite.texture = null
□□sprite.modulate = Color(0.2, 0.2, 0.7)

# Obtener nombre del palo para ruta de textura según el repositorio
func _get_suit_name(suit_value: int) -> String:
□match suit_value:
□□GameManager.CardSuit.OROS: return "oros"
□□GameManager.CardSuit.COPAS: return "copas"
□□GameManager.CardSuit.ESPADAS: return "espadas"
□□GameManager.CardSuit.BASTOS: return "bastos"
```

□□_: return "unknown"

# Obtener nombre del valor para ruta de textura según el repositorio
func _get_value_file_name(card_value: int) -> String:
□match card_value:
□□1: return "1"  # As
□□10: return "10" # Sota
□□11: return "11" # Caballo
□□12: return "12" # Rey
□□_: return str(card_value)

# Obtener nombre legible del valor para mostrar
func _get_value_name(card_value: int) -> String:
□match card_value:
□□1: return "as"
□□10: return "sota"
□□11: return "caballo"
□□12: return "rey"
□□_: return str(card_value)

# Obtener descripción de la carta
func get_description() -> String:
□var value_str = _get_value_name(value)
□var suit_str = _get_suit_name(suit)
□
□# Capitalizar primera letra
□value_str = value_str.substr(0, 1).to_upper() + value_str.substr(1)
□suit_str = suit_str.substr(0, 1).to_upper() + suit_str.substr(1)
□
□return value_str + " de " + suit_str

# Voltear la carta
func flip(face_up: bool = !is_face_up, animate: bool = true) -> void:
□is_face_up = face_up
□
□if animate and animation_player.has_animation("flip"):
□□animation_player.play("flip")
□else:
□□_update_appearance()

# Seleccionar/deseleccionar la carta
func select(selected: bool = true, animate: bool = true) -> void:
□if is_selected == selected:
□□return
□□
□is_selected = selected
□
□if animate and animation_player.has_animation("select"):
□□animation_player.play("select" if selected else "deselect")
□else:
□□_update_appearance()

# Resaltar/desresaltar la carta
func highlight(highlighted: bool = true) -> void:
□is_highlighted = highlighted
□_update_appearance()

# Habilitar/deshabilitar arrastre
func set_draggable(draggable: bool) -> void:
□is_draggable = draggable

# Mover a una posición con animación
func move_to(target_position: Vector2, duration: float = 0.3) -> void:
□if duration <= 0:
□□position = target_position
□□original_position = target_position
□□return
□
□var tween = create_tween()
□tween.tween_property(self, "position", target_position, duration).set_ease(Tween.EASE_OUT).set_trans(Tween.TRANS_CUBIC)
□tween.tween_callback(func(): original_position = position)

# Mover a un índice Z específico con animación
func move_to_z_index(target_z: int, duration: float = 0.2) -> void:
□if duration <= 0:
□□z_index = target_z
□□original_z_index = target_z
□□return
□
□var tween = create_tween()
□tween.tween_property(self, "z_index", target_z, duration)

```
□tween.tween_callback(func(): original_z_index = z_index)

# Procesamiento de entrada
func _input(event: InputEvent) -> void:
□if !is_draggable or !_dragging:
□□return
□
□if event is InputEventMouseMotion:
□□position = get_global_mouse_position() - _drag_offset
□
□if event is InputEventMouseButton:
□□if event.button_index == MOUSE_BUTTON_LEFT and !event.pressed:
□□□_end_drag()

# Iniciar arrastre
func _start_drag() -> void:
□if !is_draggable:
□□return
□
□_dragging = true
□_drag_offset = get_global_mouse_position() - position
□z_index = DRAG_Z_INDEX
□
□emit_signal("card_drag_started", self)

# Finalizar arrastre
func _end_drag() -> void:
□if !_dragging:
□□return
□
□_dragging = false
□z_index = original_z_index
□
□emit_signal("card_drag_ended", self)
□
□# La posición final se decidirá por quien reciba la señal card_drag_ended
□# Si no, la carta volverá a su posición original
□var tween = create_tween()
□tween.tween_property(self, "position", original_position, 0.2).set_ease(Tween.EASE_OUT)

# Manejadores de eventos
func _on_card_pressed() -> void:
□if is_draggable:
□□_start_drag()
□else:
□□is_selected = !is_selected
□□_update_appearance()
□
□emit_signal("card_clicked", self)

func _on_mouse_entered() -> void:
□_hover = true
□
□if !_dragging and !is_selected:
□□var tween = create_tween()
□□tween.tween_property(self, "scale", HOVER_SCALE, 0.1).set_ease(Tween.EASE_OUT)

func _on_mouse_exited() -> void:
□_hover = false
□
□if !_dragging and !is_selected:
□□var tween = create_tween()
□□tween.tween_property(self, "scale", Vector2.ONE, 0.1).set_ease(Tween.EASE_OUT)

# Para depuración
func _to_string() -> String:
□return get_description()
```

## Archivo: C:/MiChinchonWeb/scripts/cards/card.gd.uid

```
uid://bxcxto64ahwfq
```

# Archivo: C:/MiChinchonWeb/scripts/game/combination_area.gd

```
extends Node2D
# combination_area.gd
# Script para gestionar el área donde se muestran las combinaciones de cartas

# Señales
signal combination_added(combination_cards)        # Emitida cuando se añade una combinación
signal combination_removed(combination_index)      # Emitida cuando se elimina una combinación
signal combination_completed(combination_index)    # Emitida cuando una combinación está completa

# Referencias a nodos
@onready var combinations_container = $CombinationsContainer
@onready var instruction_label = $InstructionLabel
@onready var background = $Background

# Constantes
const MAX_COMBINATIONS = 3        # Número máximo de combinaciones visibles
const MIN_CARDS_PER_COMBO = 3     # Mínimo de cartas para una combinación válida
const MAX_CARDS_PER_COMBO = 7     # Máximo de cartas por combinación
const CARD_SCENE_PATH = "res://scenes/cards/card.tscn"

# Variables
var active_combinations = []       # Lista de combinaciones activas
var drag_target_combo = -1         # Índice de la combinación a la que se está arrastrando una carta
var is_accepting_combinations = false  # Si el área está actualmente aceptando combinaciones

# Función de inicialización
func _ready():
□# Ocultar inicialmente los paneles de grupos
□for i in range(MAX_COMBINATIONS):
□□var group_panel = get_group_panel(i)
□□if group_panel:
□□□group_panel.visible = false
□
□# Mostrar instrucciones iniciales
□instruction_label.visible = true
□
□# Configurar el área como no interactiva inicialmente
□set_active(false)

# Establecer si el área está activa
func set_active(active: bool):
□is_accepting_combinations = active
□
□# Ajustar visuales según estado
□background.modulate.a = 0.5 if active else 0.184
□instruction_label.visible = active and active_combinations.size() == 0
□
□# Habilitar/deshabilitar interacción
□set_process_input(active)
□set_process(active)

# Añadir una nueva combinación
func add_combination(cards: Array) -> bool:
□if active_combinations.size() >= MAX_COMBINATIONS:
□□push_warning("No se pueden añadir más combinaciones")
□□return false
□
□if cards.size() < MIN_CARDS_PER_COMBO:
□□push_warning("Se requieren al menos " + str(MIN_CARDS_PER_COMBO) + " cartas para formar una combinación")
□□return false
□
□if !is_valid_combination(cards):
□□push_warning("Las cartas no forman una combinación válida")
□□return false
□
□# Encontrar el primer panel disponible
□for i in range(MAX_COMBINATIONS):
□□var group_panel = get_group_panel(i)
□□if group_panel and !group_panel.visible:
□□□# Activar panel
□□□group_panel.visible = true
□□□
□□□# Añadir cartas a la combinación
□□□var card_container = group_panel.get_node("CardContainer")
□□□for child in card_container.get_children():
□□□□child.queue_free()
□□□
□□□var combination_data = {
```

```
□□□□"panel_index": i,
□□□□"cards": [],
□□□□"is_group": is_card_group(cards),
□□□□"suit": cards[0].suit if !is_card_group(cards) else -1,
□□□□"value": cards[0].value if is_card_group(cards) else -1
□□□}
□□□
□□□# Crear cartas visuales
□□□for card in cards:
□□□□add_card_to_combination(combination_data, card.suit, card.value)
□□□
□□□active_combinations.append(combination_data)
□□□
□□□# Ocultar instrucciones si hay al menos una combinación
□□□instruction_label.visible = false
□□□
□□□emit_signal("combination_added", cards)
□□□return true
□
□return false

# Añadir una carta a una combinación existente
func add_card_to_combination(combination, suit: int, value: int) -> bool:
□if combination.cards.size() >= MAX_CARDS_PER_COMBO:
□□return false
□
□# Verificar si la carta se puede añadir a esta combinación
□if combination.is_group:
□□# Para grupos, el valor debe coincidir
□□if combination.value != value:
□□□return false
□else:
□□# Para escaleras, debe ser del mismo palo y valor secuencial
□□if combination.suit != suit:
□□□return false
□□
□□# Ordenar cartas actuales por valor
□□var values = []
□□for card in combination.cards:
□□□values.append(card.value)
□□values.sort()
□□
□□# Verificar si la nueva carta extiende la secuencia
□□if value != values[0] - 1 and value != values[-1] + 1:
□□□return false
□
□# Crear instancia de carta visual
□var card_scene = load(CARD_SCENE_PATH)
□var card_instance = card_scene.instantiate()
□
□# Obtener contenedor de cartas del panel
□var panel_index = combination.panel_index
□var group_panel = get_group_panel(panel_index)
□var card_container = group_panel.get_node("CardContainer")
□
□# Añadir la carta visual
□card_container.add_child(card_instance)
□card_instance.setup(suit, value, true)
□card_instance.set_draggable(false)
□card_instance.scale = Vector2(0.7, 0.7) # Escala más pequeña para la combinación
□
□# Registrar la carta en la combinación
□combination.cards.append({
□□"suit": suit,
□□"value": value,
□□"node": card_instance
□})
□
□# Si la combinación está completa, emitir señal
□if combination.cards.size() >= MIN_CARDS_PER_COMBO:
□□emit_signal("combination_completed", panel_index)
□
□return true

# Eliminar una combinación
func remove_combination(index: int) -> bool:
□if index < 0 or index >= active_combinations.size():
□□return false
□
□var combination = active_combinations[index]
□var panel_index = combination.panel_index
```

```
□var group_panel = get_group_panel(panel_index)
□
□if group_panel:
□□group_panel.visible = false
□□var card_container = group_panel.get_node("CardContainer")
□□
□□# Eliminar todas las cartas
□□for child in card_container.get_children():
□□□child.queue_free()
□□
□□# Eliminar de la lista de combinaciones activas
□□active_combinations.erase(combination)
□□
□□# Mostrar instrucciones si no quedan combinaciones
□□instruction_label.visible = is_accepting_combinations and active_combinations.size() == 0
□□
□□emit_signal("combination_removed", index)
□□return true
□
□return false

# Verificar si una carta se puede añadir a una combinación existente
func can_add_to_combination(combination_index: int, card_node) -> bool:
□if combination_index < 0 or combination_index >= active_combinations.size():
□□return false
□
□var combination = active_combinations[combination_index]
□
□# Si la combinación ya está llena
□if combination.cards.size() >= MAX_CARDS_PER_COMBO:
□□return false
□
□# Verificar si la carta es compatible con la combinación
□if combination.is_group:
□□# Para grupos, el valor debe coincidir
□□return card_node.value == combination.value
□else:
□□# Para escaleras, debe ser del mismo palo y valor secuencial
□□if card_node.suit != combination.suit:
□□□return false
□□
□□# Ordenar cartas actuales por valor
□□var values = []
□□for card in combination.cards:
□□□values.append(card.value)
□□values.sort()
□□
□□# Verificar si la nueva carta extiende la secuencia
□□return card_node.value == values[0] - 1 or card_node.value == values[-1] + 1

# Obtener el panel de un grupo específico
func get_group_panel(index: int) -> Control:
□if index < 0 or index >= MAX_COMBINATIONS:
□□return null
□
□return combinations_container.get_node("Group" + str(index + 1))

# Verificar si un conjunto de cartas forma un grupo válido (mismo valor)
func is_card_group(cards: Array) -> bool:
□if cards.size() < MIN_CARDS_PER_COMBO:
□□return false
□
□var first_value = cards[0].value
□
□for card in cards:
□□if card.value != first_value:
□□□return false
□
□return true

# Verificar si un conjunto de cartas forma una escalera válida (secuencia del mismo palo)
func is_card_straight(cards: Array) -> bool:
□if cards.size() < MIN_CARDS_PER_COMBO:
□□return false
□
□# Todas las cartas deben ser del mismo palo
□var first_suit = cards[0].suit
□for card in cards:
□□if card.suit != first_suit:
□□□return false
□# Ordenar por valor
```

```
□var sorted_cards = cards.duplicate()
□sorted_cards.sort_custom(func(a, b): return a.value < b.value)
□
□# Verificar secuencia
□for i in range(1, sorted_cards.size()):
□□if sorted_cards[i].value != sorted_cards[i-1].value + 1:
□□□return false
□
□return true

# Verificar si un conjunto de cartas forma una combinación válida
func is_valid_combination(cards: Array) -> bool:
□return is_card_group(cards) or is_card_straight(cards)

# Limpiar todas las combinaciones
func clear_all_combinations():
□for i in range(active_combinations.size() - 1, -1, -1):
□□remove_combination(i)
□
□active_combinations.clear()
□instruction_label.visible = is_accepting_combinations

# Actualizar la visualización de las combinaciones
func update_combinations_display():
□for i in range(active_combinations.size()):
□□var combination = active_combinations[i]
□□var panel_index = combination.panel_index
□□var group_panel = get_group_panel(panel_index)
□□
□□if group_panel:
□□□group_panel.visible = true
□□□
□□□# Actualizar aspecto visual según el tipo de combinación
□□□if combination.is_group:
□□□□group_panel.self_modulate = Color(1.0, 0.9, 0.7, 1.0)  # Amarillo para grupos
□□□else:
□□□□group_panel.self_modulate = Color(0.7, 1.0, 0.8, 1.0)  # Verde para escaleras

# Procesar entrada para detectar arrastres de carta sobre el área
func _input(event):
□if !is_accepting_combinations:
□□return
□
□# Aquí se implementaría la lógica para detectar arrastre de cartas
□# sobre el área de combinaciones y resaltar la zona donde se puede soltar
```

## Archivo: C:/MiChinchonWeb/scripts/game/combination_area.gd.uid

```
uid://d3pi6k6a6fotj
```

# Archivo: C:/MiChinchonWeb/scripts/game/deck_manager.gd

```
extends Node2D
# deck_manager.gd
# Script para gestionar el mazo de cartas y la pila de descarte en el juego Chinchón

# Señales
signal deck_clicked         # Emitida cuando se hace clic en el mazo
signal discard_clicked      # Emitida cuando se hace clic en la pila de descarte
signal card_dealt(card_node, destination_position, player_id)  # Emitida cuando se reparte una carta
signal deck_exhausted       # Emitida cuando se agota el mazo
signal discard_recycled     # Emitida cuando la pila de descarte se recicla como nuevo mazo

# Referencias a nodos
@onready var deck_sprite: Sprite2D = $DeckSprite
@onready var discard_sprite: Sprite2D = $DiscardSprite
@onready var deck_button: TextureButton = $DeckButton
@onready var discard_button: TextureButton = $DiscardButton
@onready var card_container: Node2D = $CardContainer
@onready var animation_player: AnimationPlayer = $AnimationPlayer

# Constantes
const CARD_SCENE_PATH: String = "res://scenes/cards/card.tscn"
const DEAL_ANIMATION_SPEED: float = 0.3  # Duración de la animación al repartir
const CARD_SPACING: float = 0.5  # Espaciado entre cartas en el mazo (para efecto visual)

# Variables
var deck_position: Vector2
var discard_position: Vector2
var top_discard_card: Node2D = null  # Referencia a la carta superior de la pila de descarte
var deck_cards: Array = []  # Lista de nodos de cartas en el mazo
var discard_cards: Array = []  # Lista de nodos de cartas en la pila de descarte
var is_dealing: bool = false  # Indica si hay una animación de reparto en curso
var card_scene: PackedScene

# Función de inicialización
func _ready() -> void:
	# Guardar posiciones iniciales
	deck_position = deck_sprite.global_position
	discard_position = discard_sprite.global_position
	
	# Precargar escena de carta
	card_scene = load(CARD_SCENE_PATH)
	if card_scene == null:
		push_error("No se pudo cargar la escena de carta: " + CARD_SCENE_PATH)
	
	# Conectar señales de botones
	deck_button.connect("pressed", _on_deck_clicked)
	discard_button.connect("pressed", _on_discard_clicked)
	
	# Desactivar botón de pila de descarte inicialmente (hasta que haya una carta)
	discard_button.disabled = true
	
	# Actualizar el contador visual del mazo
	_update_deck_visuals()

# Inicializar el mazo con cartas de GameManager
func initialize_deck() -> void:
	# Limpiar mazo y pila de descarte anteriores
	clear_all_cards()
	
	# Crear carta visual para cada carta en el mazo de GameManager
	var deck_data = GameManager.deck.duplicate()
	
	for i in range(deck_data.size()):
		var card_data = deck_data[i]
		var card_instance = _create_card_from_data(card_data)
		
		# Colocar carta en el mazo (visualmente)
		card_instance.position = deck_position + Vector2(i * CARD_SPACING, i * CARD_SPACING)
		card_instance.is_face_up = false
		card_instance.z_index = i
		
		# Añadir a la lista de cartas del mazo
		deck_cards.append(card_instance)
	
	# Colocar primera carta en la pila de descarte
	if !GameManager.discard_pile.is_empty():
		var first_discard = GameManager.discard_pile[0]
		add_card_to_discard(first_discard)
```

```
□# Actualizar visuales
□_update_deck_visuals()

# Crear una carta visual a partir de datos
func _create_card_from_data(card_data: Dictionary) -> Node2D:
□var card_instance = card_scene.instantiate()
□card_container.add_child(card_instance)
□
□# Configurar valores de la carta
□card_instance.setup(card_data.suit, card_data.value, false)  # Inicialmente boca abajo
□
□return card_instance

# Agregar una carta a la pila de descarte
func add_card_to_discard(card_data: Dictionary, animate: bool = true) -> void:
□var card_instance = _create_card_from_data(card_data)
□
□# Configurar la carta
□card_instance.is_face_up = true
□card_instance.z_index = discard_cards.size()
□
□if animate:
□□# Animar desde posición actual a la pila de descarte
□□card_instance.position = card_instance.get_global_transform().origin
□□card_instance.flip(true, false)  # Voltear sin animación inicialmente
□□
□□var tween = create_tween()
□□tween.tween_property(card_instance, "position", discard_position, DEAL_ANIMATION_SPEED)
□□tween.tween_callback(func():
□□□card_instance.position = discard_position
□□□_update_discard_visuals()
□□)
□else:
□□# Colocar directamente
□□card_instance.position = discard_position
□□card_instance.flip(true, false)
□
□# Actualizar referencia a la carta superior
□top_discard_card = card_instance
□discard_cards.append(card_instance)
□
□# Activar el botón de pila de descarte
□discard_button.disabled = false
□
□# Actualizar visuales
□_update_discard_visuals()

# Repartir una carta a un jugador
func deal_card_to_player(player_id: int, destination_position: Vector2) -> void:
□if is_dealing:
□□push_warning("Ya hay una carta siendo repartida, se ignorará esta solicitud")
□□return
□
□if deck_cards.is_empty():
□□push_warning("El mazo está vacío, no se pueden repartir más cartas")
□□emit_signal("deck_exhausted")
□□return
□
□is_dealing = true
□
□# Obtener la carta superior del mazo
□var card_instance = deck_cards.pop_back()
□
□# Animar el movimiento de la carta
□var tween = create_tween()
□tween.tween_property(card_instance, "position", destination_position, DEAL_ANIMATION_SPEED)
□tween.tween_callback(func():
□□card_instance.flip(true)  # Voltear la carta al llegar
□□is_dealing = false
□□emit_signal("card_dealt", card_instance, destination_position, player_id)
□)
□
□# Actualizar visuales del mazo
□_update_deck_visuals()

# Tomar la carta superior de la pila de descarte
func take_top_discard_card() -> Node2D:
□if discard_cards.is_empty():
□□push_warning("La pila de descarte está vacía")
□□return null
□var card = discard_cards.pop_back()
```

```
    top_discard_card = discard_cards.back() if !discard_cards.is_empty() else null
    
    # Actualizar visuales
    _update_discard_visuals()
    
    return card

# Reciclar la pila de descarte como nuevo mazo
func recycle_discard_as_deck() -> void:
    if discard_cards.size() <= 1:
        push_warning("No hay suficientes cartas en la pila de descarte para reciclar")
        return
    
    # Mantener la carta superior en la pila de descarte
    var top_card = null
    if !discard_cards.is_empty():
        top_card = discard_cards.pop_back()
    
    # Mover el resto al mazo
    for card in discard_cards:
        card.flip(false)  # Voltear boca abajo
        deck_cards.append(card)
    
    # Limpiar la lista de descarte
    discard_cards.clear()
    
    # Restaurar la carta superior
    if top_card != null:
        discard_cards.append(top_card)
        top_discard_card = top_card
    
    # Mezclar visualmente las cartas del mazo
    _shuffle_deck_visually()
    
    # Actualizar visuales
    _update_deck_visuals()
    _update_discard_visuals()
    
    # Emitir señal
    emit_signal("discard_recycled")

# Mezclar visualmente las cartas del mazo
func _shuffle_deck_visually() -> void:
    deck_cards.shuffle()
    
    # Reposicionar las cartas en el mazo
    for i in range(deck_cards.size()):
        var card = deck_cards[i]
        card.z_index = i
        
        # Animar el reposicionamiento
        var target_pos = deck_position + Vector2(i * CARD_SPACING, i * CARD_SPACING)
        var tween = create_tween()
        tween.tween_property(card, "position", target_pos, 0.2)

# Limpiar todas las cartas
func clear_all_cards() -> void:
    # Eliminar todas las cartas de ambos mazo y pila de descarte
    for card in deck_cards:
        card.queue_free()
    
    for card in discard_cards:
        card.queue_free()
    
    deck_cards.clear()
    discard_cards.clear()
    top_discard_card = null
    
    # Actualizar visuales
    _update_deck_visuals()
    _update_discard_visuals()

# Actualizar visualización del mazo
func _update_deck_visuals() -> void:
    # Actualizar visibilidad del sprite del mazo
    deck_sprite.visible = !deck_cards.is_empty()
    
    # Actualizar interactividad del botón
    deck_button.disabled = deck_cards.is_empty()
    
    # Aquí se podría añadir un contador visual o efecto según cantidad de cartas
```

```
# Actualizar visualización de la pila de descarte
func _update_discard_visuals() -> void:
	# Actualizar visibilidad del sprite de la pila de descarte
	discard_sprite.visible = discard_cards.is_empty()
	
	# La carta superior es visible por sí misma, no necesita sprite adicional
	
	# Actualizar interactividad del botón
	discard_button.disabled = discard_cards.is_empty()

# Manejadores de eventos
func _on_deck_clicked() -> void:
	if !deck_cards.is_empty():
		emit_signal("deck_clicked")

func _on_discard_clicked() -> void:
	if !discard_cards.is_empty():
		emit_signal("discard_clicked")
```

## Archivo: C:/MiChinchonWeb/scripts/game/deck_manager.gd.uid

```
uid://byf22ko2okjsg
```

# Archivo: C:/MiChinchonWeb/scripts/game/main_game.gd

```
extends Node2D
# main_game.gd
# Script principal para la escena del juego Chinchón
# Coordina todos los componentes y gestiona el flujo de la partida

# Referencias a nodos
@onready var player_hand: Node2D = $PlayerHand
@onready var deck_manager: Node2D = $Table/DeckManager
@onready var ui_manager: CanvasLayer = $UILayer
@onready var opponent_container: Node2D = $OpponentsContainer
@onready var combination_area: Node2D = $CombinationArea
@onready var game_camera: Camera2D = $GameCamera
@onready var animation_player: AnimationPlayer = $AnimationPlayer
@onready var game_sound_player: AudioStreamPlayer = $GameSoundPlayer

# Variables del juego
var is_game_initialized: bool = false
var player_names: Dictionary = {}
var opponent_hands: Dictionary = {}
var current_dealer_id: int = -1
var current_round: int = 0
var active_players: Array = []
var ai_think_time: float = 1.0  # Tiempo de "pensamiento" de la IA en segundos

# Función de inicialización
func _ready() -> void:
	# Configurar cámara
	if game_camera:
		game_camera.make_current()

	# Conectar señales de GameManager
	GameManager.connect("game_started", _on_game_started)
	GameManager.connect("round_started", _on_round_started)
	GameManager.connect("player_turn_started", _on_player_turn_started)
	GameManager.connect("player_turn_ended", _on_player_turn_ended)
	GameManager.connect("card_drawn", _on_card_drawn)
	GameManager.connect("card_discarded", _on_card_discarded)
	GameManager.connect("player_chinchon", _on_player_chinchon)
	GameManager.connect("player_closed", _on_player_closed)
	GameManager.connect("round_ended", _on_round_ended)
	GameManager.connect("game_ended", _on_game_ended)
	GameManager.connect("game_paused", _on_game_paused)
	GameManager.connect("game_resumed", _on_game_resumed)

	# Conectar señales de UI
	ui_manager.connect("game_paused", _on_ui_pause_requested)
	ui_manager.connect("game_resumed", _on_ui_resume_requested)
	ui_manager.connect("new_game_requested", _on_new_game_requested)
	ui_manager.connect("main_menu_requested", _on_main_menu_requested)

	# Conectar señales del mazo
	deck_manager.connect("deck_clicked", _on_deck_clicked)
	deck_manager.connect("discard_clicked", _on_discard_clicked)

	# Conectar señales de la mano del jugador
	player_hand.connect("card_selected", _on_player_card_selected)
	player_hand.connect("card_played", _on_player_card_played)

	# Iniciar juego (para desarrollo, en producción esto se llamaría desde el menú)
	if OS.is_debug_build():
		# Solo para pruebas en desarrollo
		initialize_game()

# Inicializar un nuevo juego
func initialize_game() -> void:
	if is_game_initialized:
		return

	is_game_initialized = true

	# Crear lista de nombres según la configuración
	setup_player_names()

	# Configurar mano del jugador
	player_hand.player_id = 0
	player_hand.set_interactive(true)
	player_hand.set_player_turn(false)
	# Crear manos de oponentes
```

```
create_opponent_hands()

# Iniciar partida
GameManager.start_new_game(active_players.size(), get_player_names_array(),
  GameManager.two_deck_mode, GameManager.use_48_card_deck)

# Configurar nombres de jugadores
func setup_player_names() -> void:
player_names.clear()

# Añadir jugador humano
player_names[0] = "Jugador"
active_players = [0]

# Añadir oponentes (CPU en singleplayer, humanos en multiplayer)
var player_count = GameManager.players.size()

for i in range(1, player_count):
  if GameManager.game_mode == "singleplayer":
    player_names[i] = "CPU " + str(i)
  else:
    player_names[i] = "Jugador " + str(i + 1)

  active_players.append(i)

# Informar a UI de los nombres
ui_manager.set_player_names(player_names)

# Crear manos de oponentes
func create_opponent_hands() -> void:
# Limpiar oponentes anteriores
for child in opponent_container.get_children():
  child.queue_free()

opponent_hands.clear()

# Crear nuevos oponentes
var opponent_scene = load("res://scenes/game/opponent_hand.tscn")

for i in range(1, active_players.size()):
  var player_id = active_players[i]
  var opponent_instance = opponent_scene.instantiate()

  opponent_container.add_child(opponent_instance)
  opponent_instance.setup(player_id, player_names[player_id])

  # Posicionar según el número de oponentes
  position_opponent(opponent_instance, i, active_players.size() - 1)

  opponent_hands[player_id] = opponent_instance

# Posicionar oponente en la mesa
func position_opponent(opponent: Node2D, index: int, total: int) -> void:
var radius = 400  # Distancia desde el centro
var start_angle = -110
var end_angle = -70

# Calcular ángulo basado en posición
var angle_rad
if total > 1:
  var angle_step = (end_angle - start_angle) / (total - 1)
  angle_rad = deg_to_rad(start_angle + index * angle_step)
else:
  angle_rad = deg_to_rad((start_angle + end_angle) / 2)

# Establecer posición
var pos_x = cos(angle_rad) * radius
var pos_y = sin(angle_rad) * radius
opponent.position = Vector2(pos_x, pos_y)

# Obtener array con nombres de jugadores
func get_player_names_array() -> Array:
var names = []

for id in active_players:
  names.append(player_names[id])

return names

# Actualizar visualmente las manos después de repartir
func update_hands_visuals() -> void:
```

```
    # Actualizar mano del jugador con cartas del GameManager
    if GameManager.players.size() > 0:
        var player_cards = GameManager.players[0].hand
        player_hand.initialize_hand(player_cards)

    # Actualizar manos de oponentes
    for player_id in opponent_hands.keys():
        if player_id < GameManager.players.size():
            var opponent = opponent_hands[player_id]
            var cards = GameManager.players[player_id].hand
            opponent.update_hand(cards.size())

# Iniciar turno del jugador
func start_player_turn() -> void:
    player_hand.set_player_turn(true)
    player_hand.set_interactive(true)

    ui_manager.show_message("Tu turno - Roba una carta del mazo o pila de descarte")
    ui_manager.update_turn_indicator(0)

# Iniciar turno de oponente/CPU
func start_opponent_turn(player_id: int) -> void:
    player_hand.set_player_turn(false)
    player_hand.set_interactive(false)

    # Resaltar oponente activo
    if player_id in opponent_hands:
        opponent_hands[player_id].highlight_turn(true)

    ui_manager.update_turn_indicator(player_id)

    # Si es CPU, procesar su turno
    if GameManager.game_mode == "singleplayer":
        process_ai_turn(player_id)
    else:
        # En modo multijugador, mostrar interfaz para el jugador actual
        # (Implementar según sistema de turnos en multijugador)
        pass

# Procesar turno de la IA
func process_ai_turn(player_id: int) -> void:
    # Simular tiempo de "pensamiento" de la IA
    await get_tree().create_timer(ai_think_time).timeout

    # Determinar si tomar del mazo o pila de descarte
    var take_from_discard = false

    if !GameManager.discard_pile.is_empty():
        # Lógica simple de IA: 50% de probabilidad de tomar de la pila
        # En una implementación real, sería más complejo basado en estrategias
        take_from_discard = randf() > 0.5

    # Tomar carta
    if take_from_discard and !GameManager.discard_pile.is_empty():
        _on_discard_clicked_for_ai(player_id)
    else:
        _on_deck_clicked_for_ai(player_id)

    # Simular tiempo para "pensar" qué descartar
    await get_tree().create_timer(ai_think_time).timeout

    # Determinar qué carta descartar (lógica simple para ejemplo)
    var ai_player = GameManager.players[player_id]
    var hand = ai_player.hand

    # Por defecto, descartar una carta aleatoria
    var card_index = randi() % hand.size()

    # Descartar carta
    GameManager.discard_card(player_id, card_index)

# Procesar robo de carta del mazo para la IA
func _on_deck_clicked_for_ai(player_id: int) -> void:
    var card = GameManager.draw_card(player_id, GameManager.DrawSource.DECK)

    if card.size() > 0:
        # Animar visualmente
        if player_id in opponent_hands:
            opponent_hands[player_id].add_card_visual()

        # Sonido de robar carta
```

```
□□play_sound("card_draw")

# Procesar robo de carta de la pila para la IA
func _on_discard_clicked_for_ai(player_id: int) -> void:
□var card = GameManager.draw_card(player_id, GameManager.DrawSource.DISCARD_PILE)
□
□if card.size() > 0:
□□# Animar visualmente
□□if player_id in opponent_hands:
□□□opponent_hands[player_id].add_card_visual()
□□
□□# Actualizar visualización de la pila de descarte
□□deck_manager.take_top_discard_card()
□□
□□# Sonido de robar carta
□□play_sound("card_draw")

# Reproducir efectos de sonido
func play_sound(sound_name: String) -> void:
□var sound_path = "res://assets/audio/sfx/" + sound_name + ".ogg"
□var sound = load(sound_path)
□
□if sound and game_sound_player:
□□game_sound_player.stream = sound
□□game_sound_player.play()

# Volver al menú principal
func return_to_main_menu() -> void:
□# Cambiar a la escena del menú principal
□get_tree().change_scene_to_file("res://scenes/menus/main_menu.tscn")

# Manejadores de señales de GameManager
func _on_game_started() -> void:
□# Inicializar deck manager con las cartas del juego
□deck_manager.initialize_deck()
□
□# Actualizar visuales de las manos
□update_hands_visuals()
□
□# Actualizar UI
□ui_manager.update_round_indicator(1, GameManager.MAX_ROUNDS)
□
□# Animar inicio de partida si tenemos animación
□if animation_player and animation_player.has_animation("game_start"):
□□animation_player.play("game_start")
□else:
□□# Mostrar mensaje de inicio
□□ui_manager.show_message("¡Comienza la partida!", 2.0)

func _on_round_started(round_num: int) -> void:
□current_round = round_num
□
□# Actualizar UI
□ui_manager.update_round_indicator(round_num, GameManager.MAX_ROUNDS)
□ui_manager.show_message("Ronda " + str(round_num), 1.5)
□
□# Actualizar visuales de las manos
□update_hands_visuals()
□
□# Reiniciar mazo visual
□deck_manager.initialize_deck()

func _on_player_turn_started(player_id: int) -> void:
□# Desactivar resaltado del jugador anterior
□for opponent_id in opponent_hands.keys():
□□opponent_hands[opponent_id].highlight_turn(false)
□
□if player_id == 0:
□□# Turno del jugador humano
□□start_player_turn()
□else:
□□# Turno de un oponente
□□start_opponent_turn(player_id)

func _on_player_turn_ended(player_id: int) -> void:
□if player_id in opponent_hands:
□□opponent_hands[player_id].highlight_turn(false)

func _on_card_drawn(player_id: int, from_pile: int) -> void:
□# Actualizar visualización según quién robó y de dónde
□if player_id == 0:
```

```
□□player_hand.set_interactive(true)
□□
□□# Actualizar mensaje de instrucción
□□ui_manager.show_message("Selecciona una carta para descartar", 2.0)
□else:
□□if from_pile == GameManager.DrawSource.DISCARD_PILE:
□□□# Animar visualmente tomar carta de la pila
□□□deck_manager.take_top_discard_card()

func _on_card_discarded(player_id: int, card: Dictionary) -> void:
□# Visualizar descarte
□deck_manager.add_card_to_discard(card)
□
□# Sonido de descarte
□play_sound("card_discard")

func _on_player_chinchon(player_id: int) -> void:
□var player_name = player_names.get(player_id, "Jugador " + str(player_id + 1))
□
□# Mostrar mensaje de Chinchón
□ui_manager.show_message("¡" + player_name + " ha conseguido Chinchón!", 3.0)
□
□# Sonido de Chinchón
□play_sound("chinchon")

func _on_player_closed(player_id: int) -> void:
□var player_name = player_names.get(player_id, "Jugador " + str(player_id + 1))
□
□# Mostrar mensaje de cierre
□ui_manager.show_message("¡" + player_name + " ha cerrado el juego!", 3.0)
□
□# Sonido de cierre
□play_sound("game_close")

func _on_round_ended(scores: Dictionary) -> void:
□# Actualizar puntuaciones en la UI
□ui_manager.update_scores(scores)
□
□# Mostrar mensaje de fin de ronda
□ui_manager.show_message("Fin de la ronda " + str(current_round), 2.0)
□
□# Desactivar interactividad hasta la próxima ronda
□player_hand.set_interactive(false)
□player_hand.set_player_turn(false)

func _on_game_ended(final_scores: Dictionary) -> void:
□# Determinar ganador (menor puntuación)
□var winner_id = -1
□var min_score = 9999
□
□for player_id in final_scores.keys():
□□if final_scores[player_id] < min_score:
□□□min_score = final_scores[player_id]
□□□winner_id = player_id
□
□# Mostrar panel de fin de juego
□ui_manager.show_game_over(final_scores, winner_id)
□
□# Sonido de fin de partida
□play_sound("game_over")

func _on_game_paused() -> void:
□# Implementar pausa (como ralentizar animaciones, etc.)
□pass

func _on_game_resumed() -> void:
□# Implementar reanudación (restaurar velocidad normal, etc.)
□pass

# Manejadores de señales de UI
func _on_ui_pause_requested() -> void:
□GameManager.pause_game()

func _on_ui_resume_requested() -> void:
□GameManager.resume_game()

func _on_new_game_requested() -> void:
□# Reiniciar componentes visuales
□player_hand.clear_hand()
□deck_manager.clear_all_cards()
□for opponent_id in opponent_hands.keys():
```

```
□□opponent_hands[opponent_id].clear_hand()
□
□# Inicializar nuevo juego
□is_game_initialized = false
□initialize_game()

func _on_main_menu_requested() -> void:
□return_to_main_menu()

# Manejadores de señales del mazo
func _on_deck_clicked() -> void:
□if GameManager.game_state == GameManager.GameState.PLAYER_TURN and !GameManager.players[0].has_drawn:
□□var card = GameManager.draw_card(0, GameManager.DrawSource.DECK)
□□
□□if card.size() > 0:
□□□# Añadir la carta a la mano visual
□□□player_hand.add_card(card)
□□□
□□□# Sonido de robar carta
□□□play_sound("card_draw")

func _on_discard_clicked() -> void:
□if GameManager.game_state == GameManager.GameState.PLAYER_TURN and !GameManager.players[0].has_drawn:
□□var card = GameManager.draw_card(0, GameManager.DrawSource.DISCARD_PILE)
□□
□□if card.size() > 0:
□□□# Añadir la carta a la mano visual
□□□player_hand.add_card(card)
□□□
□□□# Actualizar visualización de la pila de descarte
□□□deck_manager.take_top_discard_card()
□□□
□□□# Sonido de robar carta
□□□play_sound("card_draw")

# Manejadores de señales de la mano del jugador
func _on_player_card_selected(card_node) -> void:
□# Implementar lógica cuando el jugador selecciona una carta
□pass

func _on_player_card_played(card_node) -> void:
□if GameManager.game_state == GameManager.GameState.PLAYER_TURN and GameManager.players[0].has_drawn:
□□var card_index = player_hand.get_card_index(card_node)
□□
□□if card_index >= 0:
□□□# Descartar la carta seleccionada
□□□var card = GameManager.discard_card(0, card_index)
□□□
□□□if card.size() > 0:
□□□□# Remover carta de la mano visual
□□□□player_hand.remove_card(card_node)
```

## Archivo: C:/MiChinchonWeb/scripts/game/main_game.gd.uid

```
uid://2mjh0d8nox1p
```

# Archivo: C:/MiChinchonWeb/scripts/game/opponent_hand.gd

```
[gd_scene load_steps=18 format=3 uid="uid://dvs3xgswrvbfj"]

[ext_resource type="Script" path="res://scripts/game/main_game.gd" id="1_2f6xt"]
[ext_resource type="PackedScene" uid="uid://cax76c32chyra" path="res://scenes/cards/deck_manager.tscn" id="2_k6g4l"]
[ext_resource type="PackedScene" uid="uid://c36jvhysosdkc" path="res://scenes/game/player_hand.tscn" id="3_e8v3q"]
[ext_resource type="Script" path="res://scripts/ui/ui_manager.gd" id="4_3fkst"]
[ext_resource type="Texture2D" uid="uid://dqmq5cuqd6s74" path="res://assets/images/ui/table_background.png" id="5_abdy5"]
[ext_resource type="PackedScene" uid="uid://bkuoxkowaq1lf" path="res://scenes/game/combination_area.tscn" id="6_ihnkl"]
[ext_resource type="Texture2D" uid="uid://c3qpbuo8q784v" path="res://assets/images/ui/pause_button.png" id="7_7hcoe"]
[ext_resource type="FontFile" uid="uid://cbrwe0wv4lsqf" path="res://assets/fonts/Montserrat-Medium.ttf" id="8_dnbrr"]
[ext_resource type="Texture2D" uid="uid://b8h8v8rahjkpx" path="res://assets/images/ui/help_button.png" id="9_0l5lv"]
[ext_resource type="AudioStream" uid="uid://d3xdlc8dko4q0" path="res://assets/audio/music/game_music.ogg" id="10_qfp3g"]

[sub_resource type="StyleBoxFlat" id="StyleBoxFlat_kbm52"]
bg_color = Color(0.0784314, 0.243137, 0.0784314, 0.870588)
border_width_left = 2
border_width_top = 2
border_width_right = 2
border_width_bottom = 2
border_color = Color(0.309804, 0.611765, 0.309804, 1)
corner_radius_top_left = 8
corner_radius_top_right = 8
corner_radius_bottom_right = 8
corner_radius_bottom_left = 8
shadow_color = Color(0, 0, 0, 0.2)
shadow_size = 4

[sub_resource type="StyleBoxFlat" id="StyleBoxFlat_wv6yy"]
bg_color = Color(0.0392157, 0.121569, 0.0392157, 0.921569)
border_width_left = 2
border_width_top = 2
border_width_right = 2
border_width_bottom = 2
border_color = Color(0.215686, 0.427451, 0.215686, 1)
corner_radius_top_left = 8
corner_radius_top_right = 8
corner_radius_bottom_right = 8
corner_radius_bottom_left = 8
shadow_color = Color(0, 0, 0, 0.2)
shadow_size = 4

[sub_resource type="StyleBoxFlat" id="StyleBoxFlat_q5xsd"]
bg_color = Color(0.196078, 0.380392, 0.196078, 0.792157)
border_width_left = 2
border_width_top = 2
border_width_right = 2
border_width_bottom = 2
border_color = Color(0.309804, 0.611765, 0.309804, 1)
corner_radius_top_left = 8
corner_radius_top_right = 8
corner_radius_bottom_right = 8
corner_radius_bottom_left = 8
shadow_color = Color(0, 0, 0, 0.2)
shadow_size = 4

[sub_resource type="StyleBoxFlat" id="StyleBoxFlat_f6j8t"]
bg_color = Color(0.0784314, 0.133333, 0.2, 0.870588)
border_width_left = 2
border_width_top = 2
border_width_right = 2
border_width_bottom = 2
border_color = Color(0.25098, 0.431373, 0.701961, 1)
corner_radius_top_left = 8
corner_radius_top_right = 8
corner_radius_bottom_right = 8
corner_radius_bottom_left = 8
shadow_color = Color(0, 0, 0, 0.2)
shadow_size = 4

[sub_resource type="StyleBoxFlat" id="StyleBoxFlat_txgje"]
bg_color = Color(0.0980392, 0.321569, 0.188235, 0.92549)
border_width_left = 2
border_width_top = 2
border_width_right = 2
border_width_bottom = 2
border_color = Color(0.196078, 0.643137, 0.376471, 1)
corner_radius_top_left = 12
corner_radius_top_right = 12
```

```
corner_radius_bottom_right = 12
corner_radius_bottom_left = 12
shadow_color = Color(0, 0, 0, 0.223529)
shadow_size = 6

[sub_resource type="Animation" id="Animation_jfnvp"]
resource_name = "game_start"
length = 2.0
tracks/0/type = "value"
tracks/0/imported = false
tracks/0/enabled = true
tracks/0/path = NodePath("Table:modulate")
tracks/0/interp = 1
tracks/0/loop_wrap = true
tracks/0/keys = {
"times": PackedFloat32Array(0, 1),
"transitions": PackedFloat32Array(0.5, 1),
"update": 0,
"values": [Color(0.5, 0.5, 0.5, 1), Color(1, 1, 1, 1)]
}
tracks/1/type = "value"
tracks/1/imported = false
tracks/1/enabled = true
tracks/1/path = NodePath("PlayerHand:position")
tracks/1/interp = 1
tracks/1/loop_wrap = true
tracks/1/keys = {
"times": PackedFloat32Array(0, 1, 1.5),
"transitions": PackedFloat32Array(0.5, 0.5, 1),
"update": 0,
"values": [Vector2(640, 800), Vector2(640, 600), Vector2(640, 620)]
}
tracks/2/type = "value"
tracks/2/imported = false
tracks/2/enabled = true
tracks/2/path = NodePath("OpponentsContainer:modulate")
tracks/2/interp = 1
tracks/2/loop_wrap = true
tracks/2/keys = {
"times": PackedFloat32Array(0, 0.7, 1.5),
"transitions": PackedFloat32Array(0.5, 0.5, 1),
"update": 0,
"values": [Color(0, 0, 0, 0), Color(0, 0, 0, 0), Color(1, 1, 1, 1)]
}
tracks/3/type = "value"
tracks/3/imported = false
tracks/3/enabled = true
tracks/3/path = NodePath("UILayer/GameUI:modulate")
tracks/3/interp = 1
tracks/3/loop_wrap = true
tracks/3/keys = {
"times": PackedFloat32Array(0, 1.5, 2),
"transitions": PackedFloat32Array(0.5, 0.5, 1),
"update": 0,
"values": [Color(1, 1, 1, 0), Color(1, 1, 1, 0), Color(1, 1, 1, 1)]
}

[sub_resource type="AnimationLibrary" id="AnimationLibrary_ihnvq"]
_data = {
"game_start": SubResource("Animation_jfnvp")
}

[node name="MainGame" type="Node2D"]
script = ExtResource("1_2f6xt")

[node name="Table" type="Sprite2D" parent="."]
position = Vector2(640, 360)
scale = Vector2(1.25, 1.25)
texture = ExtResource("5_abdy5")

[node name="DeckManager" parent="Table" instance=ExtResource("2_k6g4l")]
position = Vector2(0, 52)

[node name="CombinationArea" parent="." instance=ExtResource("6_ihnkl")]
position = Vector2(640, 360)

[node name="PlayerHand" parent="." instance=ExtResource("3_e8v3q")]
position = Vector2(640, 620)

[node name="OpponentsContainer" type="Node2D" parent="."]
position = Vector2(640, 360)
```

```
[node name="UILayer" type="CanvasLayer" parent="."]
script = ExtResource("4_3fkst")

[node name="GameUI" type="Control" parent="UILayer"]
layout_mode = 3
anchors_preset = 15
anchor_right = 1.0
anchor_bottom = 1.0
grow_horizontal = 2
grow_vertical = 2
mouse_filter = 2

[node name="TopBar" type="HBoxContainer" parent="UILayer/GameUI"]
layout_mode = 1
anchors_preset = 10
anchor_right = 1.0
offset_top = 10.0
offset_bottom = 60.0
grow_horizontal = 2
mouse_filter = 2

[node name="LeftSpacer" type="Control" parent="UILayer/GameUI/TopBar"]
layout_mode = 2
size_flags_horizontal = 3
mouse_filter = 2

[node name="RoundIndicator" type="Label" parent="UILayer/GameUI/TopBar"]
layout_mode = 2
size_flags_horizontal = 4
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 22
text = "Ronda 1 de 5"
horizontal_alignment = 1
vertical_alignment = 1

[node name="CenterSpacer" type="Control" parent="UILayer/GameUI/TopBar"]
layout_mode = 2
size_flags_horizontal = 3
mouse_filter = 2

[node name="TurnIndicator" type="Label" parent="UILayer/GameUI/TopBar"]
layout_mode = 2
size_flags_horizontal = 4
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 22
text = "¡Tu turno!"
horizontal_alignment = 1
vertical_alignment = 1

[node name="RightSpacer" type="Control" parent="UILayer/GameUI/TopBar"]
layout_mode = 2
size_flags_horizontal = 3
mouse_filter = 2

[node name="MessageLabel" type="Label" parent="UILayer/GameUI"]
visible = false
layout_mode = 1
anchors_preset = 8
anchor_left = 0.5
anchor_top = 0.5
anchor_right = 0.5
anchor_bottom = 0.5
offset_left = -250.0
offset_top = -25.0
offset_right = 250.0
offset_bottom = 25.0
grow_horizontal = 2
grow_vertical = 2
theme_override_colors/font_outline_color = Color(0, 0, 0, 1)
theme_override_constants/outline_size = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 24
text = "¡Comienza el juego!"
horizontal_alignment = 1
vertical_alignment = 1

[node name="PauseButton" type="TextureButton" parent="UILayer/GameUI"]
layout_mode = 1
anchors_preset = 1
anchor_left = 1.0
anchor_right = 1.0
```

```
offset_left = -60.0
offset_top = 10.0
offset_right = -10.0
offset_bottom = 60.0
grow_horizontal = 0
texture_normal = ExtResource("7_7hcoe")
ignore_texture_size = true
stretch_mode = 5

[node name="HelpButton" type="TextureButton" parent="UILayer/GameUI"]
layout_mode = 1
anchors_preset = 1
anchor_left = 1.0
anchor_right = 1.0
offset_left = -120.0
offset_top = 10.0
offset_right = -70.0
offset_bottom = 60.0
grow_horizontal = 0
texture_normal = ExtResource("9_0l5lv")
ignore_texture_size = true
stretch_mode = 5

[node name="ScorePanel" type="PanelContainer" parent="UILayer/GameUI"]
layout_mode = 0
offset_left = 20.0
offset_top = 20.0
offset_right = 220.0
offset_bottom = 200.0
theme_override_styles/panel = SubResource("StyleBoxFlat_kbm52")

[node name="VBoxContainer" type="VBoxContainer" parent="UILayer/GameUI/ScorePanel"]
layout_mode = 2

[node name="TitleLabel" type="Label" parent="UILayer/GameUI/ScorePanel/VBoxContainer"]
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 20
text = "Puntuaciones"
horizontal_alignment = 1

[node name="HSeparator" type="HSeparator" parent="UILayer/GameUI/ScorePanel/VBoxContainer"]
layout_mode = 2

[node name="ScoresContainer" type="VBoxContainer" parent="UILayer/GameUI/ScorePanel/VBoxContainer"]
layout_mode = 2
size_flags_vertical = 3

[node name="PauseMenu" type="PanelContainer" parent="UILayer"]
process_mode = 2
visible = false
anchors_preset = 8
anchor_left = 0.5
anchor_top = 0.5
anchor_right = 0.5
anchor_bottom = 0.5
offset_left = -150.0
offset_top = -175.0
offset_right = 150.0
offset_bottom = 175.0
grow_horizontal = 2
grow_vertical = 2
theme_override_styles/panel = SubResource("StyleBoxFlat_wv6yy")

[node name="VBoxContainer" type="VBoxContainer" parent="UILayer/PauseMenu"]
layout_mode = 2
theme_override_constants/separation = 20

[node name="TitleLabel" type="Label" parent="UILayer/PauseMenu/VBoxContainer"]
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 28
text = "Pausa"
horizontal_alignment = 1

[node name="HSeparator" type="HSeparator" parent="UILayer/PauseMenu/VBoxContainer"]
layout_mode = 2

[node name="ResumeButton" type="Button" parent="UILayer/PauseMenu/VBoxContainer"]
layout_mode = 2
size_flags_vertical = 3
```

```
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 20
theme_override_styles/normal = SubResource("StyleBoxFlat_q5xsd")
text = "Reanudar"

[node name="SettingsButton" type="Button" parent="UILayer/PauseMenu/VBoxContainer"]
layout_mode = 2
size_flags_vertical = 3
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 20
theme_override_styles/normal = SubResource("StyleBoxFlat_q5xsd")
text = "Configuración"

[node name="MainMenuButton" type="Button" parent="UILayer/PauseMenu/VBoxContainer"]
layout_mode = 2
size_flags_vertical = 3
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 20
theme_override_styles/normal = SubResource("StyleBoxFlat_f6j8t")
text = "Menú Principal"

[node name="GameOverPanel" type="PanelContainer" parent="UILayer"]
process_mode = 2
visible = false
anchors_preset = 8
anchor_left = 0.5
anchor_top = 0.5
anchor_right = 0.5
anchor_bottom = 0.5
offset_left = -250.0
offset_top = -200.0
offset_right = 250.0
offset_bottom = 200.0
grow_horizontal = 2
grow_vertical = 2
theme_override_styles/panel = SubResource("StyleBoxFlat_txgje")

[node name="VBoxContainer" type="VBoxContainer" parent="UILayer/GameOverPanel"]
layout_mode = 2
theme_override_constants/separation = 15

[node name="TitleLabel" type="Label" parent="UILayer/GameOverPanel/VBoxContainer"]
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 32
text = "Fin del Juego"
horizontal_alignment = 1

[node name="HSeparator" type="HSeparator" parent="UILayer/GameOverPanel/VBoxContainer"]
layout_mode = 2

[node name="WinnerLabel" type="Label" parent="UILayer/GameOverPanel/VBoxContainer"]
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 24
text = "¡Jugador 1 gana!"
horizontal_alignment = 1

[node name="ResultsLabel" type="Label" parent="UILayer/GameOverPanel/VBoxContainer"]
layout_mode = 2
size_flags_vertical = 3
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 20
text = "Puntuaciones finales:

Jugador 1: 45
CPU 1: 78
CPU 2: 105 (Eliminado)"
horizontal_alignment = 1
vertical_alignment = 1

[node name="HSeparator2" type="HSeparator" parent="UILayer/GameOverPanel/VBoxContainer"]
layout_mode = 2

[node name="PlayAgainButton" type="Button" parent="UILayer/GameOverPanel/VBoxContainer"]
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 22
theme_override_styles/normal = SubResource("StyleBoxFlat_q5xsd")
text = "Jugar de Nuevo"
[node name="ExitButton" type="Button" parent="UILayer/GameOverPanel/VBoxContainer"]
```

```
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 22
theme_override_styles/normal = SubResource("StyleBoxFlat_f6j8t")
text = "Menú Principal"

[node name="MessageTimer" type="Timer" parent="UILayer"]
one_shot = true

[node name="SettingsPanel" type="PanelContainer" parent="UILayer"]
process_mode = 2
visible = false
anchors_preset = 8
anchor_left = 0.5
anchor_top = 0.5
anchor_right = 0.5
anchor_bottom = 0.5
offset_left = -200.0
offset_top = -200.0
offset_right = 200.0
offset_bottom = 200.0
grow_horizontal = 2
grow_vertical = 2
theme_override_styles/panel = SubResource("StyleBoxFlat_wv6yy")

[node name="TooltipPanel" type="PanelContainer" parent="UILayer"]
visible = false
offset_right = 200.0
offset_bottom = 60.0
theme_override_styles/panel = SubResource("StyleBoxFlat_wv6yy")

[node name="TooltipLabel" type="Label" parent="UILayer/TooltipPanel"]
layout_mode = 2
theme_override_fonts/font = ExtResource("8_dnbrr")
theme_override_font_sizes/font_size = 16
text = "Tooltip text"
horizontal_alignment = 1
vertical_alignment = 1
autowrap_mode = 3

[node name="ConfirmationDialog" type="ConfirmationDialog" parent="UILayer"]
initial_position = 2
size = Vector2i(400, 150)
dialog_text = "¿Estás seguro de que quieres volver al menú principal?
Se perderá el progreso actual."

[node name="GameCamera" type="Camera2D" parent="."]
position = Vector2(640, 360)

[node name="AnimationPlayer" type="AnimationPlayer" parent="."]
libraries = {
"": SubResource("AnimationLibrary_ihnvq")
}

[node name="GameSoundPlayer" type="AudioStreamPlayer" parent="."]

[node name="BackgroundMusic" type="AudioStreamPlayer" parent="."]
stream = ExtResource("10_qfp3g")
volume_db = -10.0
bus = &"Music"
```

## Archivo: C:/MiChinchonWeb/scripts/game/opponent_hand.gd.uid

```
uid://53imy26n4oup
```

# Archivo: C:/MiChinchonWeb/scripts/game/player_hand.gd

```
extends Node2D
# player_hand.gd
# Script para gestionar la mano de cartas del jugador en el juego Chinchón

# Señales
signal card_selected(card_node)    # Emitida cuando se selecciona una carta
signal card_deselected(card_node)  # Emitida cuando se deselecciona una carta
signal card_played(card_node)      # Emitida cuando se juega una carta
signal hand_sorted()               # Emitida cuando se ordena la mano
signal combination_formed(cards)   # Emitida cuando se detecta una combinación válida

# Constantes
const CARD_SCENE_PATH: String = "res://scenes/cards/card.tscn"
const CARD_WIDTH: float = 140.0    # Ancho visual de una carta
const CARD_SPACING: float = 50.0   # Espacio entre cartas cuando están desplegadas
const CARD_OVERLAP: float = 30.0   # Espacio entre cartas cuando están agrupadas
const ANIMATION_SPEED: float = 0.2 # Duración de las animaciones de reorganización
const MAX_HAND_WIDTH: float = 900.0 # Ancho máximo para desplegar las cartas

# Variables
var player_id: int = 0             # ID del jugador propietario de esta mano
var cards: Array = []              # Array de nodos de cartas en la mano
var selected_cards: Array = []     # Array de cartas seleccionadas actualmente
var is_interactive: bool = true    # Si la mano permite interacción del usuario
var is_sorting: bool = false       # Si actualmente se está ordenando la mano
var card_scene: PackedScene        # Referencia a la escena de carta precargada
var hand_center: Vector2           # Centro de la mano para posicionamiento
var is_player_turn: bool = false   # Si es actualmente el turno del jugador
var max_cards: int = 7             # Número máximo de cartas en mano (por defecto 7)

# Referencias a nodos
@onready var sort_button: Button = $SortButton
@onready var play_button: Button = $PlayButton
@onready var cards_container: Node2D = $CardsContainer

# Función de inicialización
func _ready() -> void:
    # Precargar escena de carta
    card_scene = load(CARD_SCENE_PATH)
    if card_scene == null:
        push_error("No se pudo cargar la escena de carta: " + CARD_SCENE_PATH)

    # Establecer centro de la mano
    hand_center = global_position

    # Conectar señales
    sort_button.connect("pressed", _on_sort_button_pressed)
    play_button.connect("pressed", _on_play_button_pressed)

    # Actualizar estado de botones
    _update_buttons_state()

# Inicializar la mano con cartas
func initialize_hand(data_cards: Array) -> void:
    # Limpiar mano actual
    clear_hand()

    # Añadir las nuevas cartas
    for card_data in data_cards:
        add_card(card_data)

    # Ordenar y organizar
    sort_hand()

# Añadir una carta a la mano
func add_card(card_data: Dictionary, animate: bool = true) -> Node2D:
    if cards.size() >= max_cards + 1:
        push_warning("La mano ya tiene el máximo de cartas (" + str(max_cards + 1) + ")")
        return null

    # Crear instancia de carta
    var card_instance = card_scene.instantiate()
    cards_container.add_child(card_instance)

    # Configurar valores de la carta
    card_instance.setup(card_data.suit, card_data.value, true)
    card_instance.owner_id = player_id
    # Configurar interactividad
```

```
card_instance.set_draggable(is_interactive)
card_instance.connect("card_clicked", _on_card_clicked)
card_instance.connect("card_drag_ended", _on_card_drag_ended)

# Añadir a la lista de cartas
cards.append(card_instance)

# Posicionar la carta
if animate:
    _organize_cards()
else:
    _set_card_positions_immediately()

# Actualizar estado de botones
_update_buttons_state()

# Verificar automáticamente posibles combinaciones
_check_for_combinations()

return card_instance

# Eliminar una carta de la mano
func remove_card(card_node: Node2D) -> void:
if card_node in selected_cards:
    selected_cards.erase(card_node)

if card_node in cards:
    cards.erase(card_node)
    card_node.queue_free()

# Reorganizar las cartas restantes
_organize_cards()

# Actualizar estado de botones
_update_buttons_state()

# Verificar automáticamente posibles combinaciones
_check_for_combinations()

# Remover una carta por su índice
func remove_card_at(index: int) -> void:
if index < 0 or index >= cards.size():
    push_error("Índice fuera de rango: " + str(index))
    return

var card = cards[index]
remove_card(card)

# Ordenar las cartas en la mano
func sort_hand() -> void:
if is_sorting:
    return

is_sorting = true

# Ordenar primero por palo, luego por valor
cards.sort_custom(func(a, b):
    if a.suit == b.suit:
        return a.value < b.value
    return a.suit < b.suit
)

# Reorganizar visualmente
_organize_cards()

is_sorting = false
emit_signal("hand_sorted")

# Reorganizar visualmente las cartas
func _organize_cards() -> void:
if cards.is_empty():
    return

# Calcular el espacio entre cartas según el número de cartas
var total_width = min(CARD_WIDTH * cards.size() + CARD_SPACING * (cards.size() - 1), MAX_HAND_WIDTH)
var actual_spacing = (total_width - CARD_WIDTH) / max(cards.size() - 1, 1)

# Posición inicial (centrada)
var start_x = hand_center.x - total_width / 2

# Animar el movimiento de cada carta
```

```
□for i in range(cards.size()):
□□var card = cards[i]
□□var target_position = Vector2(start_x + i * (CARD_WIDTH + actual_spacing), hand_center.y)
□□var target_z_index = i
□□
□□# Si está seleccionada, ajustar posición vertical
□□if card in selected_cards:
□□□target_position.y -= 20
□□
□□# Animar el movimiento
□□card.move_to(target_position, ANIMATION_SPEED)
□□card.move_to_z_index(target_z_index, ANIMATION_SPEED)

# Posicionar cartas inmediatamente (sin animación)
func _set_card_positions_immediately() -> void:
□if cards.is_empty():
□□return
□
□# Calcular el espacio entre cartas según el número de cartas
□var total_width = min(CARD_WIDTH * cards.size() + CARD_SPACING * (cards.size() - 1), MAX_HAND_WIDTH)
□var actual_spacing = (total_width - CARD_WIDTH) / max(cards.size() - 1, 1)
□
□# Posición inicial (centrada)
□var start_x = hand_center.x - total_width / 2
□
□# Posicionar cada carta
□for i in range(cards.size()):
□□var card = cards[i]
□□var target_position = Vector2(start_x + i * (CARD_WIDTH + actual_spacing), hand_center.y)
□□
□□# Si está seleccionada, ajustar posición vertical
□□if card in selected_cards:
□□□target_position.y -= 20
□□
□□# Posicionar directamente
□□card.position = target_position
□□card.original_position = target_position
□□card.z_index = i
□□card.original_z_index = i

# Limpiar la mano
func clear_hand() -> void:
□for card in cards:
□□card.queue_free()
□
□cards.clear()
□selected_cards.clear()
□
□# Actualizar estado de botones
□_update_buttons_state()

# Establecer interactividad de la mano
func set_interactive(interactive: bool) -> void:
□is_interactive = interactive
□
□# Actualizar interactividad de cada carta
□for card in cards:
□□card.set_draggable(interactive)
□
□# Actualizar estado de botones
□_update_buttons_state()

# Establecer si es el turno del jugador
func set_player_turn(is_turn: bool) -> void:
□is_player_turn = is_turn
□
□# Actualizar estado de botones
□_update_buttons_state()

# Verificar posibles combinaciones en la mano
func _check_for_combinations() -> void:
□# Esta función identificará posibles combinaciones válidas en la mano
□# (grupos del mismo valor o escaleras del mismo palo)
□
□if cards.size() < 3:
□□return  # Se necesitan al menos 3 cartas para formar una combinación
□
□# Buscar grupos (3 o 4 cartas del mismo valor)
□var values_count = {}
□for card in cards:
□□if card.value not in values_count:
```

□□□values_count[card.value] = []
□□values_count[card.value].append(card)
□
□# Verificar si hay grupos de 3 o 4 cartas del mismo valor
□for value in values_count.keys():
□□if values_count[value].size() >= 3:
□□□var combo = values_count[value]
□□□emit_signal("combination_formed", combo)
□
□# Buscar escaleras (3 o más cartas consecutivas del mismo palo)
□var cards_by_suit = {}
□for card in cards:
□□if card.suit not in cards_by_suit:
□□□cards_by_suit[card.suit] = []
□□cards_by_suit[card.suit].append(card)
□
□for suit in cards_by_suit.keys():
□□if cards_by_suit[suit].size() < 3:
□□□continue
□□
□□# Ordenar cartas por valor
□□var suit_cards = cards_by_suit[suit]
□□suit_cards.sort_custom(func(a, b): return a.value < b.value)
□□
□□# Buscar secuencias consecutivas
□□var i = 0
□□while i < suit_cards.size() - 2:  # Necesitamos al menos 3 cartas
□□□var sequence = [suit_cards[i]]
□□□var current_value = suit_cards[i].value
□□□
□□□var j = i + 1
□□□while j < suit_cards.size() and suit_cards[j].value == current_value + 1:
□□□□sequence.append(suit_cards[j])
□□□□current_value = suit_cards[j].value
□□□□j += 1
□□□
□□□if sequence.size() >= 3:
□□□□emit_signal("combination_formed", sequence)
□□□
□□□i = j

# Obtener cartas seleccionadas
func get_selected_cards() -> Array:
□return selected_cards

# Obtener el índice de una carta en la mano
func get_card_index(card_node: Node2D) -> int:
□return cards.find(card_node)

# Verificar si la mano tiene una carta con valores específicos
func has_card_with_values(suit: int, value: int) -> bool:
□for card in cards:
□□if card.suit == suit and card.value == value:
□□□return true
□return false

# Jugar las cartas seleccionadas
func play_selected_cards() -> void:
□if selected_cards.is_empty():
□□return
□
□# Verificar si las cartas seleccionadas forman una combinación válida
□if _is_valid_combination(selected_cards):
□□for card in selected_cards:
□□□emit_signal("card_played", card)
□□selected_cards.clear()
□else:
□□# Mostrar mensaje de error o feedback visual
□□push_warning("Las cartas seleccionadas no forman una combinación válida")
□□
□□# Deseleccionar todas las cartas
□□for card in selected_cards:
□□□card.select(false)
□□selected_cards.clear()
□
□# Reorganizar las cartas
□_organize_cards()
□
□# Actualizar estado de botones
□_update_buttons_state()
# Verificar si las cartas forman una combinación válida

```
func _is_valid_combination(card_nodes: Array) -> bool:
    if card_nodes.size() < 3:
        return false  # Se necesitan al menos 3 cartas

    # Verificar si es un grupo (mismo valor)
    var is_group = true
    var first_value = card_nodes[0].value

    for card in card_nodes:
        if card.value != first_value:
            is_group = false
            break

    if is_group:
        return true

    # Verificar si es una escalera (mismo palo, valores consecutivos)
    var is_straight = true
    var first_suit = card_nodes[0].suit

    # Verificar que todas las cartas sean del mismo palo
    for card in card_nodes:
        if card.suit != first_suit:
            is_straight = false
            break

    if not is_straight:
        return false

    # Ordenar por valor
    var sorted_cards = card_nodes.duplicate()
    sorted_cards.sort_custom(func(a, b): return a.value < b.value)

    # Verificar que sean valores consecutivos
    for i in range(1, sorted_cards.size()):
        if sorted_cards[i].value != sorted_cards[i-1].value + 1:
            is_straight = false
            break

    return is_straight

# Actualizar el estado de los botones según la situación actual
func _update_buttons_state() -> void:
    sort_button.disabled = cards.size() < 2 or not is_interactive
    play_button.disabled = selected_cards.size() < 3 or not is_interactive or not is_player_turn

    # Mostrar/ocultar los botones según corresponda
    sort_button.visible = is_interactive
    play_button.visible = is_interactive and is_player_turn

# Manejadores de eventos
func _on_card_clicked(card_node) -> void:
    if not is_interactive:
        return

    # Alternar selección de la carta
    if card_node in selected_cards:
        selected_cards.erase(card_node)
        card_node.select(false)
        emit_signal("card_deselected", card_node)
    else:
        selected_cards.append(card_node)
        card_node.select(true)
        emit_signal("card_selected", card_node)

    # Reorganizar para reflejar la selección
    _organize_cards()

    # Actualizar estado de botones
    _update_buttons_state()

func _on_card_drag_ended(card_node) -> void:
    if not is_interactive or not is_player_turn:
        return

    # Aquí se podría implementar lógica para detectar si la carta
    # fue arrastrada a una zona de juego o a la pila de descarte

    # Por ahora, solo reorganizamos las cartas
    _organize_cards()
func _on_sort_button_pressed() -> void:
```

□sort_hand()

func _on_play_button_pressed() -> void:
□play_selected_cards()

## Archivo: C:/MiChinchonWeb/scripts/game/player_hand.gd.uid

```
uid://xmjwvny56pra
```

# Archivo: C:/MiChinchonWeb/scripts/menus/main_menu.gd

```
extends Control
# main_menu.gd
# Script para el menú principal del juego Chinchón

# Señales
signal start_singleplayer  # Emitida cuando se inicia el modo un jugador
signal start_multiplayer   # Emitida cuando se inicia el modo multijugador
signal open_settings       # Emitida cuando se abren las configuraciones
signal exit_game           # Emitida cuando se solicita salir del juego

# Referencias a nodos
@onready var single_player_button: Button = $MainContainer/ButtonsContainer/SinglePlayerButton
@onready var multiplayer_button: Button = $MainContainer/ButtonsContainer/MultiplayerButton
@onready var settings_button: Button = $MainContainer/ButtonsContainer/SettingsButton
@onready var exit_button: Button = $MainContainer/ButtonsContainer/ExitButton
@onready var version_label: Label = $VersionLabel
@onready var player_setup_panel: Control = $PlayerSetupPanel
@onready var settings_panel: Control = $SettingsPanel
@onready var animation_player: AnimationPlayer = $AnimationPlayer
@onready var background_music: AudioStreamPlayer = $BackgroundMusic
@onready var logo_animation: AnimatedSprite2D = $LogoAnimation

# Variables
var player_count: int = 1  # Número de jugadores (por defecto 1 jugador)
var player_names: Array = ["Jugador"]  # Nombres de los jugadores
var use_48_card_deck: bool = true  # Usar baraja de 48 cartas por defecto
var two_deck_mode: bool = false  # Modo de una o dos barajas

# Función de inicialización
func _ready() -> void:
	# Mostrar versión del juego
	version_label.text = "v" + GameManager.GAME_VERSION
	
	# Conectar señales de botones
	single_player_button.connect("pressed", _on_single_player_button_pressed)
	multiplayer_button.connect("pressed", _on_multiplayer_button_pressed)
	settings_button.connect("pressed", _on_settings_button_pressed)
	exit_button.connect("pressed", _on_exit_button_pressed)
	
	# Configurar paneles
	player_setup_panel.visible = false
	settings_panel.visible = false
	
	# Conectar botones del panel de configuración de jugadores
	var start_game_button = player_setup_panel.get_node("VBoxContainer/StartGameButton")
	var back_button = player_setup_panel.get_node("VBoxContainer/BackButton")
	var player_count_slider = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer/PlayerCountSlider")
	var deck_toggle = player_setup_panel.get_node("VBoxContainer/DeckOptionsContainer/DeckTypeToggle")
	var two_deck_toggle = player_setup_panel.get_node("VBoxContainer/DeckOptionsContainer/TwoDeckToggle")
	
	start_game_button.connect("pressed", _on_start_game_button_pressed)
	back_button.connect("pressed", _on_back_button_pressed)
	player_count_slider.connect("value_changed", _on_player_count_slider_changed)
	deck_toggle.connect("toggled", _on_deck_toggle_toggled)
	two_deck_toggle.connect("toggled", _on_two_deck_toggle_toggled)
	
	# Configurar panel de configuraciones
	var settings_back_button = settings_panel.get_node("VBoxContainer/BackButton")
	settings_back_button.connect("pressed", _on_settings_back_button_pressed)
	
	# Iniciar animaciones y música
	if animation_player.has_animation("menu_intro"):
		animation_player.play("menu_intro")
	
	if background_music.stream != null:
		background_music.play()
	
	if logo_animation != null:
		logo_animation.play("default")

# Mostrar panel de configuración de jugadores para modo un jugador
func show_singleplayer_setup() -> void:
	player_count = 1
	player_names = ["Jugador"]
	
	# Configurar panel para modo un jugador
	var player_count_container = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer")
	var player_count_slider = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer/PlayerCountSlider")
```

```
□var player_count_label = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer/PlayerCountValue")
□var title_label = player_setup_panel.get_node("VBoxContainer/TitleLabel")
□
□title_label.text = "Configuración de Juego Solitario"
□player_count_container.visible = true  # Mostrar selección de número de oponentes
□player_count_slider.min_value = 1
□player_count_slider.max_value = 7
□player_count_slider.value = 2  # Jugador + 1 oponente por defecto
□player_count_label.text = "Oponentes: 1"
□
□# Actualizar opciones de baraja
□update_deck_options()
□
□# Mostrar panel con animación
□_transition_to_panel(player_setup_panel)

# Mostrar panel de configuración de jugadores para modo multijugador
func show_multiplayer_setup() -> void:
□player_count = 2
□player_names = ["Jugador 1", "Jugador 2"]
□
□# Configurar panel para modo multijugador
□var player_count_container = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer")
□var player_count_slider = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer/PlayerCountSlider")
□var player_count_label = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer/PlayerCountValue")
□var title_label = player_setup_panel.get_node("VBoxContainer/TitleLabel")
□
□title_label.text = "Configuración de Multijugador"
□player_count_container.visible = true  # Mostrar selección de número de jugadores
□player_count_slider.min_value = 2
□player_count_slider.max_value = 8
□player_count_slider.value = 2  # 2 jugadores por defecto
□player_count_label.text = "Jugadores: 2"
□
□# Actualizar opciones de baraja
□update_deck_options()
□
□# Mostrar panel con animación
□_transition_to_panel(player_setup_panel)

# Actualizar opciones de baraja según el número de jugadores
func update_deck_options() -> void:
□var two_deck_toggle = player_setup_panel.get_node("VBoxContainer/DeckOptionsContainer/TwoDeckToggle")
□var two_deck_container = player_setup_panel.get_node("VBoxContainer/DeckOptionsContainer/TwoDeckContainer")
□
□# Habilitar opción de dos barajas solo si hay 5+ jugadores
□two_deck_container.visible = player_count >= 5
□
□if player_count >= 5 and !two_deck_mode:
□□two_deck_toggle.button_pressed = true
□□two_deck_mode = true
□elif player_count < 5 and two_deck_mode:
□□two_deck_toggle.button_pressed = false
□□two_deck_mode = false

# Mostrar panel de configuraciones
func show_settings_panel() -> void:
□_transition_to_panel(settings_panel)

# Volver al menú principal
func back_to_main_menu() -> void:
□_transition_to_panel(null)

# Función para transicionar entre paneles
func _transition_to_panel(target_panel: Control) -> void:
□# Ocultar todos los paneles excepto el menú principal
□if player_setup_panel.visible:
□□player_setup_panel.visible = false
□
□if settings_panel.visible:
□□settings_panel.visible = false
□
□# Mostrar el panel objetivo
□if target_panel != null:
□□target_panel.visible = true
□□
□□# Animar transición si es posible
□□if animation_player.has_animation("panel_transition"):
□□□animation_player.play("panel_transition")
□else:
□□# Mostrar menú principal
```

```
    if animation_player.has_animation("menu_return"):
        animation_player.play("menu_return")

# Iniciar juego con configuración actual
func start_game() -> void:
    var actual_player_count = player_count

    if GameManager.game_mode == "singleplayer":
        actual_player_count += 1  # Añadir al jugador humano a los oponentes

    # Configurar GameManager
    GameManager.use_48_card_deck = use_48_card_deck
    GameManager.two_deck_mode = two_deck_mode

    # Crear la lista de nombres de jugadores para la partida
    var game_player_names = []

    # Modo un jugador (jugador + CPU)
    if GameManager.game_mode == "singleplayer":
        game_player_names.append("Jugador")

        # Añadir oponentes CPU
        for i in range(1, actual_player_count):
            game_player_names.append("CPU " + str(i))
    else:
        # Modo multijugador
        for i in range(actual_player_count):
            game_player_names.append("Jugador " + str(i + 1))

    # Iniciar el juego
    if GameManager.game_mode == "singleplayer":
        emit_signal("start_singleplayer", actual_player_count, game_player_names, two_deck_mode, use_48_card_deck)
    else:
        emit_signal("start_multiplayer", actual_player_count, game_player_names, two_deck_mode, use_48_card_deck)

# Manejadores de eventos
func _on_single_player_button_pressed() -> void:
    GameManager.game_mode = "singleplayer"
    show_singleplayer_setup()

func _on_multiplayer_button_pressed() -> void:
    GameManager.game_mode = "multiplayer"
    show_multiplayer_setup()

func _on_settings_button_pressed() -> void:
    show_settings_panel()
    emit_signal("open_settings")

func _on_exit_button_pressed() -> void:
    emit_signal("exit_game")

func _on_start_game_button_pressed() -> void:
    start_game()

func _on_back_button_pressed() -> void:
    back_to_main_menu()

func _on_settings_back_button_pressed() -> void:
    back_to_main_menu()

func _on_player_count_slider_changed(value: float) -> void:
    var count_value = int(value)
    player_count = count_value

    var label = player_setup_panel.get_node("VBoxContainer/PlayerCountContainer/PlayerCountValue")

    if GameManager.game_mode == "singleplayer":
        label.text = "Oponentes: " + str(count_value - 1)
    else:
        label.text = "Jugadores: " + str(count_value)

    # Actualizar opciones de baraja
    update_deck_options()

func _on_deck_toggle_toggled(button_pressed: bool) -> void:
    use_48_card_deck = button_pressed

    var label = player_setup_panel.get_node("VBoxContainer/DeckOptionsContainer/DeckTypeLabel")

    if button_pressed:
        label.text = "Baraja de 48 cartas"
```

```
□else:
□□label.text = "Baraja de 40 cartas"

func _on_two_deck_toggle_toggled(button_pressed: bool) -> void:
□two_deck_mode = button_pressed
□
□var label = player_setup_panel.get_node("VBoxContainer/DeckOptionsContainer/TwoDeckContainer/TwoDeckLabel")
□
□if button_pressed:
□□label.text = "Usando dos barajas"
□else:
□□label.text = "Usando una baraja"
```

## Archivo: C:/MiChinchonWeb/scripts/menus/main_menu.gd.uid

```
uid://d1w3qis11ue7e
```