TABLA DE RESPONSABILIDADES - PROYECTO BLACKJACK

Nombre del método	*Requerimiento asociado*	*Precondición*	*Postcondición*	*Modelo verbal*
Controlador(Vista& pVista, Jugador& pJugador, Crupier& pCrupier, Apuesta& pApuesta)	CTL-1	Existen instancias válidas de Vista, Jugador, Crupier y Apuesta	Controlador inicializado con referencias a todos los componentes del juego	Inicializar el controlador recibiendo referencias a la vista y los modelos principales para gestionar toda la interacción del sistema
getOpcionInt(int opcionMin, int opcionMax) const	CTL-2	Menú con opciones numéricas disponible en vista	Retorna opción válida dentro del rango especificado	Solicitar y validar entrada numérica del usuario dentro del rango, limpiando buffer y mostrando error si entrada inválida
getOpcionChar(const std::string& opciones, Menu menu) const	CTL-3	Menú activo con opciones de caracteres definidas	Retorna carácter válido convertido a mayúscula	Solicitar y validar carácter del usuario de entre opciones válidas, actualizar pantalla según menú activo en caso de error
esperarInput(const std::string& texto) const	CTL-4	Vista inicializada correctamente	Usuario presiona tecla y pantalla principal mostrada	Mostrar mensaje de error en pantalla completa, esperar input del usuario y retornar a pantalla principal
getNombreJugador(size_t charMin, size_t charMax) const	CTL-5	Vista inicializada y rangos válidos definidos	Retorna nombre válido del jugador	Solicitar y validar nickname verificando longitud esté entre mínimo y máximo, mostrar mensaje si inválido
limpiarBuffer() const	CTL-6	Buffer de entrada puede contener datos residuales	Buffer de entrada completamente limpio	Limpiar buffer de entrada ignorando caracteres hasta nueva línea para evitar lecturas erróneas
Apuesta()	AP-1	Ninguna	Objeto Apuesta inicializado con dineroInicial=1000, dineroTotal=1000, apuestaActual=0	Crear instancia de Apuesta con valores económicos por defecto del sistema de juego
aumentarApuesta(int cantidad)	AP-2	Cantidad positiva y dinero total	Apuesta incrementada y dinero	Incrementar apuesta actual en cantidad indicada descontando

		suficiente para cubrir apuesta	descontado, retorna true si exitoso o false si fondos insuficientes	de dinero total si hay saldo suficiente
resetearApuesta()	AP-3	Apuesta en estado modificable	Dinero total restaurado a 1000 inicial y apuesta actual en cero	Restablecer completamente el sistema de apuestas al estado inicial por defecto
ganar()	AP-4	Apuesta activa mayor a cero y resultado ganador	Ganancia del doble de apuesta sumada a dinero total, apuesta reiniciada a cero	Aplicar pago por victoria duplicando la apuesta y sumándola al dinero total del jugador
perder()	AP-5	Apuesta activa y resultado perdedor	Apuesta establecida en cero sin modificar dinero total	Registrar pérdida de apuesta sin devolver dinero (ya fue descontado al apostar)
empatar()	AP-6	Apuesta activa y resultado de empate	Apuesta devuelta sumándola al dinero total, apuesta reiniciada a cero	Devolver apuesta al jugador sumándola de vuelta al dinero total disponible
getDineroTotal() const	AP-7	Objeto Apuesta inicializado	Retorna valor entero del dinero total disponible	Devolver el saldo total actual del jugador para visualización y validaciones
getApuestaActual() const	AP-8	Apuesta definida o en cero	Retorna valor entero de la apuesta en curso	Retornar el monto específico que el jugador tiene apostado en la ronda actual
setDineroTotal(int cantidad)	AP-9	Objeto Apuesta inicializado y cantidad válida	Dinero total establecido al nuevo valor	Establecer manualmente dinero total del jugador, usado principalmente al cargar partida guardada
Carta(int pValorNominal, const std::string& pPalo, const Color& pColor)	C-1	Valores nominales entre 2-14, palo válido y color válido	Instancia Carta creada con atributos almacenados y cambioAUno=false	Crear carta con valor nominal (2-14 donde 11=As, 12=J, 13=Q, 14=K), palo Unicode y color

convertirAsAUno()	C-2	Carta es un As (valorNominal==11) y no ha sido convertida previamente	Valor nominal permanentemente cambiado a 1 y flag cambioAUno=true	Convertir valor del As de 11 a 1 para optimizar puntaje y marcar como convertido permanentemente
getValor() const	C-3	Carta inicializada correctamente	Retorna valor real de juego de la carta	Devolver valor según reglas: figuras (>11) retornan 10, demás cartas retornan su valor nominal
getPalo() const	C-4	Carta inicializada correctamente	Retorna string con símbolo Unicode del palo	Devolver símbolo del palo (♥ corazones, ♦ diamantes, ♣ tréboles, ♠ picas)
getColor() const	C-5	Carta inicializada correctamente	Retorna enum Color (NEGRO o ROJO)	Devolver color enumerado asociado al palo de la carta para renderizado visual
getEstado() const	C-6	Carta inicializada correctamente	Retorna booleano del estado de conversión	Devolver si el As ha sido convertido de 11 a 1, usado para validar ajustes de mano
getValorASCII() const	C-7	Carta inicializada correctamente	Retorna string con representación visual del valor	Devolver "A" para As, "J"/"Q"/"K" para figuras, número en string para cartas numéricas
Mazo()	M-1	Ninguna	Objeto Mazo creado con 52 cartas barajadas e índice en 0	Inicializar mazo llamando automáticamente a generarMazo para crear y mezclar baraja completa
generarMazo()	M-2	Vector de cartas puede estar vacío o poblado	Vector limpiado, 52 cartas generadas con colores correctos, barajadas e índice en 0	Generar baraja estándar (4 palos × 13 valores) asignando colores apropiados y barajar inmediatamente
barajar()	M-3	Mazo poblado con 52 cartas	Orden completamente aleatorio aplicado usando semilla temporal, índice reiniciado a 0	Aplicar std::shuffle con semilla basada en tiempo del sistema para mezclar cartas aleatoriamente

getIndiceCartaActual() const	M-4	Mazo inicializado	Retorna size_t con índice de próxima carta	Devolver índice actual que apunta a la siguiente carta disponible para repartir
getCartasRestantes() const	M-5	Mazo inicializado	Retorna size_t con cantidad de cartas no repartidas	Calcular y devolver cuántas cartas quedan por repartir (tamaño - índice actual)
avanzarIndice()	M-6	Mazo inicializado y cartas disponibles	Índice incrementado en 1	Avanzar índice de carta actual para preparar siguiente carta a repartir
Participante(Vista& pVista)	P-1	Existe instancia válida de Vista	Participante inicializado con referencia a vista y vector mano vacío	Crear participante base estableciendo referencia a vista para coordinación de renderizado visual
recibirCarta(const Carta& carta)	P-2	Carta válida proporcionada	Carta añadida a vector mano, mano ajustada si necesario, carta enviada a vista para renderizado	Agregar carta al vector mano, ejecutar ajuste automático de Ases y notificar vista para mostrar carta
limpiarMano()	P-3	Participante con mano existente	Vector mano vaciado y colas visuales de vista limpiadas	Vaciar completamente vector de cartas y limpiar todas las colas de renderizado de la vista
ajustarMano()	P-4	Mano con al menos una carta	Ases convertidos de 11 a 1 secuencialmente si valor total >21	Convertir Ases de 11 a 1 uno por uno hasta que mano valga ≤21 o se agoten Ases disponibles
getTipoDeMano() const	P-5	Método implementado en subclase concreta	Retorna enum Mano (JUGADOR o CRUPIER)	Método virtual puro que identifica tipo de participante para selección de cola visual correcta
getValorDeMano() const	P-6	Mano con cartas (puede estar vacía)	Retorna entero con suma total de valores de cartas	Calcular y devolver puntaje actual sumando valores reales de todas las cartas en mano

getConteoDeCartas() const	P-7	Mano inicializada	Retorna size_t con cantidad de cartas	Devolver tamaño del vector mano indicando cuántas cartas tiene el participante
getMano() const	P-8	Mano inicializada	Retorna referencia constante a vector de cartas	Devolver referencia al vector completo de cartas para inspección externa sin modificación
Jugador(Vista& pVista)	J-1	Instancia válida de Vista existe	Jugador instanciado con nombre vacío y mano vacía heredada	Crear objeto Jugador invocando constructor padre de Participante con referencia a vista
getTipoDeMano() const	J-2	Jugador inicializado	Retorna enum JUGADOR	Implementar método virtual para identificar este participante como jugador humano
getNombre() const	J-3	Jugador inicializado	Retorna string con nombre del jugador	Devolver nickname almacenado del jugador para mostrar en interfaz y guardar partida
setNombre(const std::string& pNombre)	J-4	Jugador inicializado y string válido	Nombre almacenado en atributo privado	Establecer nickname del jugador proporcionado por input del usuario o carga de partida
Crupier(Mazo& pMazo, Vista& pVista, Jugador& pJugador)	CR-1	Instancias válidas de Mazo, Vista y Jugador existen	Crupier inicializado con referencias a mazo, vista y jugador	Crear crupier estableciendo acceso al mazo para repartir y al jugador para coordinación de juego
empezarNuevaRonda()	CR-2	Crupier y jugador con manos previas o vacías	Ambas manos limpiadas completamente, 2 cartas iniciales repartidas a cada participante	Limpiar manos de ambos participantes y ejecutar reparto inicial estándar de 2 cartas cada uno
getValorDeManoParcial() const	CR-3	Crupier tiene al menos una carta en mano	Retorna valor entero de primera carta únicamente	Devolver solo valor de primera carta visible del crupier, ocultando resto durante turno del jugador

getSiguienteCarta()	CR-4	Mazo tiene cartas disponibles para repartir	Índice del mazo avanzado, carta retornada	Avanzar índice del mazo y retornar carta en nueva posición para distribución
darCartaAJugador(int cantidad)	CR-5	Mazo con cartas suficientes disponibles	Cantidad especificada de cartas añadidas a mano del jugador	Obtener cartas del mazo secuencialmente y entregarlas al jugador la cantidad de veces indicada
darCartaACrupier(int cantidad)	CR-6	Mazo con cartas suficientes disponibles	Cantidad especificada de cartas añadidas a mano propia del crupier	Obtener cartas del mazo secuencialmente y añadirlas a mano propia la cantidad de veces indicada
jugarTurno()	CR-7	Turno del crupier activo, jugador plantado	Crupier pidió cartas automáticamente hasta tener valor ≥17	Ejecutar regla automática obligatoria: pedir cartas repetidamente mientras valor sea menor a 17
evaluarEstado() const	CR-8	Ambos participantes tienen cartas repartidas	Retorna GameState con situación actual del juego	Evaluar situación en tiempo real y retornar BUST, BLACKJACK, GANAR, PERDER, EMPATE o NONE
decidirResultado()	CR-9	Jugador se plantó finalizando su turno	Crupier ejecutó turno completo, GameState final determinado	Ejecutar turno automático del crupier y determinar resultado final comparando valores de ambas manos
getTipoDeMano() const	CR-10	Crupier inicializado	Retorna enum CRUPIER	Implementar método virtual para identificar este participante como crupier/dealer
Vista()	V-1	Sistema de salida estándar disponible	Vista inicializada con 6 vectores de colas vacíos	Crear instancia de vista inicializando estructuras de colas para renderizado de cartas
mostrarTitulo() const	V-2	Vista inicializada correctamente	Pantalla limpiada y título ASCII amarillo mostrado	Limpiar pantalla completamente y mostrar logo grande del juego en arte ASCII con color
mostrarMenuPrincipal() const	V-3	Vista inicializada correctamente	Menú principal con 4 opciones mostrado	Presentar opciones principales: nueva partida, cargar, reglas, salir con formato visual

mostrarMenuApuesta() const	V-4	Vista inicializada correctamente	Menú de apuestas con fichas coloreadas mostrado	Presentar opciones de apuesta (100, 250, 500) con representación visual de fichas en colores
mostrarMenuAcciones(GameState estado) const	V-5	Estado del juego conocido y válido	Menú contextual apropiado mostrado	Mostrar "Tomar/Plantarse" si estado NONE, o "Jugar de nuevo S/N" si partida finalizada
mostrarMenuGuardado() const	V-6	Vista inicializada correctamente	Pantalla limpia con pregunta de guardado centrada	Mostrar pantalla dedicada preguntando si desea guardar partida con opciones S/N
mostrarGameData(const std::string& nombre, int dinero, int apuesta) const	V-7	Datos válidos del estado actual del juego	Línea informativa con datos del jugador mostrada	Mostrar en formato compacto: nombre jugador, dinero total (cian) y apuesta actual (amarillo)
mostrarEstado(GameState estado) const	V-8	Estado de juego válido (puede ser NONE)	Mensaje de resultado apropiado mostrado con color	Mostrar mensaje según estado: "GANASTE!"(cian), "Perdiste"(rojo), "BUST!"(rojo), "Empate"(negro), "BLACKJACK!"(amarillo) o vacío
mostrarReglas() const	V-9	Vista inicializada correctamente	Pantalla completa de reglas con bordes mostrada	Mostrar explicación detallada de objetivo, valores de cartas, acciones, reglas del crupier y condiciones de victoria
mostrarPantallaPrincipal() const	V-10	Vista inicializada correctamente	Pantalla completa de inicio renderizada	Mostrar título grande y menú principal juntos como pantalla de bienvenida
mostrarPantallaApuesta(const std::string& nombre, int dinero, int apuesta)	V-11	Datos válidos proporcionados	Pantalla completa de apuestas renderizada	Mostrar título, datos del juego actuales y menú de apuestas con fichas
mostrarPantallaJuego(const std::string& nombre, int dinero, int apuesta, const std::string& valorJugador, const std::string& valorCrupier, GameState estado)	V-12	Datos y estado válidos proporcionados	Pantalla completa de juego con todas las secciones renderizada	Mostrar datos del juego, mano del crupier (parcial o completa según estado), mensaje de estado, mano del jugador y menú de acciones

solicitarInput(const std::string& mensaje) const	V-13	Vista inicializada y mensaje preparado	Mensaje de solicitud mostrado con sangría	Imprimir mensaje solicitando input del usuario con formato y sangría estándar
mostrarTexto(const std::string& texto) const	V-14	Texto preparado	Texto mostrado con sangría y salto de línea	Imprimir texto simple con sangría estándar de 10 espacios y nueva línea
mostrarTexto(const std::string& texto, const std::string& color) const	V-15	Texto y código de color ANSI preparados	Texto coloreado mostrado con sangría y salto	Imprimir texto con color ANSI especificado, sangría estándar y reset de color
mostrarTexto(const std::string& prefijo, const std::string& texto, const std::string& sufijo) const	V-16	Strings preparados correctamente	Texto compuesto mostrado sin modificaciones	Imprimir concatenación directa de prefijo + texto + sufijo sin sangría ni saltos automáticos
mostrarErrorFullScreen(const std::string& texto) const	V-17	Texto de error preparado	Pantalla limpia con error centrado verticalmente mostrado	Limpiar pantalla, añadir espaciado vertical y mostrar mensaje de error en rojo centrado
limpiarPantalla() const	V-18	Interfaz puede tener contenido previo	Pantalla/terminal completamente limpiada	Ejecutar comando de sistema apropiado (cls en Windows, clear en Unix/Linux)
añadirCartaACola(Mano mano, const std::string& valor, const std::string& palo, Color colorEnum)	V-19	Parámetros válidos de carta proporcionados	Representación ASCII de carta añadida a 3 vectores de cola apropiada	Crear representación visual de carta en 3 partes (superior, medio, inferior) y añadir a cola según tipo de mano, gestionando carta oculta del crupier
imprimirCola(const std::vectorstd::string& cola) const	V-20	Cola contiene elementos de cartas	Todos elementos del vector impresos horizontalmente	Iterar vector imprimiendo cada elemento consecutivamente sin saltos para efecto de apilamiento horizontal
esColaVacia(std::vectorstd::string (&cola)[3]) const	V-21	Array de 3 vectores definido y pasado por referencia	Retorna true si los 3 vectores están vacíos	Verificar que los 3 vectores (superior, medio, inferior) del array estén completamente vacíos

limpiarColas()	V-22	Colas definidas (pueden tener contenido)	6 vectores de colas completamente vaciados	Limpiar todas las estructuras: 3 vectores del jugador, 3 del crupier incluyendo cola parcial
imprimirMano(Mano mano, GameState estado, const std::string& valor)	V-23	Mano válida, estado y valor calculado proporcionados	Mano completa impresa en 5 líneas con valor y cartas apiladas	Seleccionar cola apropiada según mano y estado, imprimir valor total con sangría, renderizar 3 líneas medias y línea inferior de cartas con efecto visual de profundidad
Serializador(const std::string& pNombreArchivo, const Vista& pVista, const Controlador& pControlador)	S-1	Referencias válidas a Vista y Controlador, nombre de archivo válido	Serializador inicializado con nombre de archivo y referencias	Crear serializador almacenando nombre de archivo y referencias para mostrar mensajes de error
serializarDatos(const Jugador& jugador, const Apuesta& apuesta)	S-2	Jugador con nombre válido y apuesta con dinero válido	Retorna string en formato CSV "nombre,dinero"	Convertir nombre del jugador y dinero total a formato CSV simple para almacenamiento
deserializarDatos()	S-3	Archivo puede existir o no, puede estar corrupto	Retorna par (nombre, dinero) si válido o ("", 0) si error	Abrir archivo, leer datos, validar formato CSV, convertir string a entero, manejar errores y retornar par con información o valores vacíos
guardarPartida(const Jugador& jugador, const Apuesta& apuesta)	S-4	Jugador y apuesta con datos válidos para serializar	Datos serializados escritos en archivo o mensaje de error mostrado	Abrir archivo en modo escritura, serializar datos del jugador, escribir en archivo y cerrar, mostrar error si falla apertura
empezarRonda(...)	F-1	Partida preparada con apuesta válida realizada	Ronda completada, dinero actualizado según resultado, retorna booleano para continuar	Ejecutar ciclo completo de ronda: iniciar ronda, bucle de acciones (tomar/plantar) hasta estado final, mostrar resultado, procesar pago, preguntar continuar con guardado opcional
prepararNuevaPartida(...)	F-2	Objetos de juego inicializados correctamente	Partida lista para jugar o cancelada, retorna true si	Solicitar nombre si es primera vez, gestionar menú de apuesta con validaciones de dinero suficiente y apuesta mínima,

			continuar o false si salir	permitir cancelación con opción de guardado
cicloJuego(...)	F-3	Todos componentes del sistema inicializados	Secuencia de partidas completada hasta que usuario decide salir	Ejecutar bucle alternando entre preparar nueva partida y jugar ronda hasta que usuario cancele apuesta o decida no continuar
main()	M-1	Programa ejecutado con todas dependencias	Aplicación finalizada normalmente o código de error	Inicializar todos objetos del sistema (Vista, Mazo, Jugador, Crupier, Apuesta, Controlador, Serializador), mostrar menú principal en bucle infinito y gestionar 4 opciones: nueva partida, cargar partida con validación, mostrar reglas, salir