# Manual Técnico

A continuación, se presenta una descripción de cada uno de los métodos y funciones que se utilizaron dentro del proyecto.

El siguiente método es el método main y este se invoca cuando se inicia a la ejecución del programa, se presenta un menú con opciones y el usuario puede seleccionar si desea iniciar un nuevo juego o si desea salir del programa, además tiene una validación en caso el número este fuera del rango.

```
int main() {
    int opcion = 0;

    while (opcion != 2) {
        cout << endl;
        cout << "Menu Principal" << endl;
        cout << "   1. Iniciar Partida" << endl;
        cout << "   2. Salir" << endl;
        cin >> opcion;

        if (opcion == 1) {
            iniciarPartida();
        } else if (opcion != 2) {
            cout << "\n----------------------------------------" << endl;
            cout << "Numero fuera de Rango, Intentalo Nuevamente" << endl;
            cout << "----------------------------------------" << endl;
        }
    }
}
```

El siguiente método fue necesario para tener un código más limpio y para evitar estar copiando y pegando la misma información en este caso es sobre el código que utilizamos para obtener la lista correspondiente al momento de realizar un cambio o simplemente a la hora de mostrar las listas en pantalla, y con este método solo requiere mandarle el índice correspondiente para obtener la lista requerida.

```
Lista* obtenerLista(int indice, Lista *lista_1, Lista *lista_2, Lista *lista_3, Lista *lista_4,
    Lista *listaAEnviar = new Lista();

    if (indice == 1) {
        listaAEnviar = lista_1;
    } else if (indice == 2) {
        listaAEnviar = lista_2;
    } else if (indice == 3) {
        listaAEnviar = lista_3;
    } else if (indice == 4) {
        listaAEnviar = lista_4;
    } else if (indice == 5) {
        listaAEnviar = lista_5;
    } else if (indice == 6) {
        listaAEnviar = lista_6;
    } else if (indice == 7) {
        listaAEnviar = lista_7;
    }
    return listaAEnviar;
}
```

El siguiente método es utilizado al iniciar un nuevo juego y su función es distribuir cada una de las cartas dentro de las siete filas del juego y el resto depositarlas dentro del montón de cartas que el usuario puede ir pasando, y la forma de realizar esta operación es primero haciendo un ciclo while en cual vamos a ir agregando las cartas a cada una de las filas del juego, primero verificamos que la carta no haya sido utilizada anteriormente, luego comparamos que el tamaño de la fila no haya sobrepasado el tamaño permitido para esa fila, por ultimo si todo está bien se procede a realizar la inserción, sumarle uno al contador de cartas, obtener la siguiente fila y aumentar el contador del arreglo que nos permite saber cuánto es el límite de cartas que podemos ingresar dentro de una fila y que cuando llegue al final se sale del ciclo.

Cuando solo queden 24 cartas sin colocar significa que las 7 filas ya están llenas por lo tanto el resto de cartas se agregan a la fila de cartas de donde el usuario va poder tomar cartas para ir completando su juego.

```
void insertarCartas(Lista *lista_1, Lista *lista_2, Lista *lista_3, Lista *lista_4, Lista *lista_5, Lista *lista_6,
    Lista *listaAAgregar = new Lista();
    int dimensiones[7] = {1,2,3,4,5,6,7};
    int contador = 52, indice = 1;
    srand(time(NULL));
    string cartasTotales[52] = {"A<3R", "2<3R", "3<3R", "4<3R", "5<3R", "6<3R", "7<3R", "8<3R", "9<3R", "I<3R", "J<3
                                "A)(N", "2)(N", "3)(N", "4)(N", "5)(N", "6)(N", "7)(N", "8)(N", "9)(N", "I)(N", "J)(
                                "A<>R", "2<>R", "3<>R", "4<>R", "5<>R", "6<>R", "7<>R", "8<>R", "9<>R", "I<>R", "J<>
                                "A!!N", "2!!N", "3!!N", "4!!N", "5!!N", "6!!N", "7!!N", "8!!N", "9!!N", "I!!N", "J!!

    listaAAgregar = obtenerLista(indice, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);

    while (contador != 24) {
        int num=rand()%52;

        if (cartasTotales[num] != "####") {
            if (listaAAgregar->obtenerLongitud() != dimensiones[indice - 1]) {
                listaAAgregar->ingresarDato(cartasTotales[num]);
                cartasTotales[num] = "####";
                contador--;
            } else {
                indice++;
                listaAAgregar->obtenerNodoFinal()->contenido->mostrar = true;
                listaAAgregar = obtenerLista(indice, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);
            }
        }
    }
    listaAAgregar->obtenerNodoFinal()->contenido->mostrar = true;
    bool salir = false;
    contador = 0;

    while (!salir) {
        if (cartasTotales[contador] != "####") {
            cola_2->ingresarDato(cartasTotales[contador]);
        }
        if (contador == 51) {
            break;
        }
        contador++;
    }
}
```

El siguiente método es utilizado al momento de realizar una inserción en cualquiera de las 7 filas del juego, como primer paso realiza una búsqueda comparando la lista a recibir el valor con los elementos del arreglo que dan el orden correcto de inserción, luego que los dos valores sean iguales se procede a revisar que el valor no sea el último en la cadena de cartas ya que eso significa que ya no hay otra carta después de esta, caso contrario obtenemos el valor siguiente del arreglo, luego obtenemos el color de la última carta y guardamos el contrario y luego de hacer esto volvemos a la ejecución del programa.

```
void verificarInsercionDeCarta(Lista *listaARecibir, string& numeroSiguiente, string& colorSiguiente, bool& salir) {
    string ordenDeLasCartas[12] = {"K", "Q", "J", "I", "9", "8", "7", "6", "5", "4", "3", "2"};

    for (int i = 0; i < 12; i++) {
        if (listaARecibir->obtenerNodoFinal()->contenido->numero == ordenDeLasCartas[i]) {
            if (i == 11) {
                salir = true;
            } else {
                numeroSiguiente = ordenDeLasCartas[i + 1];

                if (listaARecibir->obtenerNodoFinal()->contenido->color == "R") {
                    colorSiguiente = "N";
                } else {
                    colorSiguiente = "R";
                }
            }
            break;
        }
    }
}
```

El siguiente método es utilizado para verificar que las cartas que se encuentran en el tope de las pilas no sean ases, si se encontrará una se procede a colocarla en el espacio destinado dentro de la vista para que el usuario sepa que ya fue localizada y se procede a eliminar el elemento de la pila correspondiente, y por último le da la instrucción al nuevo elemento del tope que muestre su cara.

```
string verificarAses(Lista *lista_1, Lista *lista_2, Lista *lista_3, Lista *lista_4, Lista *lista_5, Lista
    Lista *listaAAgregar = new Lista();
    string a_Verificar = "";

    for (int i = 1; i <= 8; i++) {
        if (i < 8) {
            listaAAgregar = obtenerLista(i, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);
        } else {
            listaAAgregar = cola_2;
        }
        if (!listaAAgregar->estaVacia()) {
            if (listaAAgregar->obtenerNodoFinal()->contenido->numero == "A") {
                a_Verificar = listaAAgregar->obtenerNodoFinal()->contenido->cara;
                listaAAgregar->eliminar();

                if (!listaAAgregar->estaVacia()) {
                    listaAAgregar->obtenerNodoFinal()->contenido->mostrar = true;
                }
                break;
            }
        }
    }
    return a_Verificar;
}
```

El siguiente método es utilizado para toda la parte grafica de las cartas que son presentadas al usuario, primero se imprimen las ases, seguido de las dos pilas de cartas que el usuario valla seleccionando y por último se muestran las cartas de las filas que el usuario tiene para el juego, y por medio de validaciones se mira si la lista a mostrar esta vacía o no y en cualquiera de los casos se busca se busca cubrir cualquier posible error que se pueda presentar así como garantizar que la presentación al jugador sea lo más agradable posible.

```
void mostrarJuego(Lista *lista_1, Lista *lista_2, Lista *lista_3, Lista *lista_4, Lista *lista_5, Lista *lista_6, Lista *lista_7,
    cout << endl;
    cout << "-------------------SOLITARIO-------------------" << endl << endl;
    cout << "A(s):" << endl;
    cout << " " << a_1 << "  " << a_2 << "  " << a_3 << "  " << a_4 << " ";
    cout << endl << endl;
    cout << "Cartas Restantes:" << endl;

    for (int i = 1; i <= 2; i++) {
        Lista *colaAMostrar = new Lista();

        if (i == 1) {
            if (cola_2->estaVacia()) {
                colaAMostrar = new Lista();
            } else {
                colaAMostrar = cola_1;
            }
        } else if (i == 2) {
            if (cola_2->estaVacia()) {
                int longitud = cola_1->obtenerLongitud();

                for (int i = 0; i < longitud; i++) {
                    cola_2->ingresarDato(cola_1->obtenerNodoFinal()->contenido->cara);
                    cola_1->eliminar();
                }
                colaAMostrar = cola_2;
            } else {
                colaAMostrar = cola_2;
            }
        }
    if (!colaAMostrar->estaVacia()) {

        if (i == 1) {
            cout << " C(" << colaAMostrar->obtenerLongitud()  << ")" << ": ####";
        } else {
            cout << " C(" << colaAMostrar->obtenerLongitud()  << ")" << ": " << colaAMostrar->obtenerNodoFinal()->contenido->cara;
        }
    } else {
        cout << " C(0)" << ": ----   ";
    }
}

cout << endl << endl;
cout << "Cartas:" << endl;

Lista *listaAMostrar = new Lista();
for (int i = 1; i < 8; i++) {
    Nodo *actual = new Nodo();
    listaAMostrar = obtenerLista(i, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);

    if (!listaAMostrar->estaVacia()) {
        actual = listaAMostrar->obtenerNodoInicial();
        cout << " - F(" << i << "): ";

        for (int i = 0; i < listaAMostrar->obtenerLongitud(); i++) {
            string elemento = "####";

            if (actual->contenido->mostrar == true) {
                elemento = actual->contenido->cara;
            }
```

```
                cout << elemento << " ";
                actual = actual->nodoSiguiente;
            }
        } else {
            cout << " - F(" << i << "): ";
        }
        cout << endl;
        listaAMostrar = NULL;
    }
}
```

Este método es utilizado para iniciar el juego y es donde se encuentra el menú principal del juego y donde el usuario va poder seleccionar entre 4 opciones y que según la opción que elija se va redireccionar a una acción para que pueda completarse el ciclo del programa, se tiene validaciones para evitar que se ingrese un número fuera de rango, así como validaciones de para verificar que las pilas y colas no estén vacías cuando se trate de manipular sus datos.

En este apartado es también donde se inicializan los datos y donde se declaran las variables por ejemplo las letras, además dentro de este método se llama a la mayoría de métodos de otros como parte de su completo por lo tanto este método se convierte en el motor principal del juego.

```
void iniciarPartida() {
    string a_1 = "----", a_2 = "----", a_3 = "----", a_4 = "----";
    bool salir = false;
    int opcion = 0;
    Lista *lista_1 = new Lista(), *lista_2 = new Lista(), *lista_3 = new Lista(), *lista_4 = new Lista(), *lista_5 = new Lista(),
    Lista *cola_1 = new Lista(), *cola_2 = new Lista();
    insertarCartas(lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7, cola_2);

    while (!salir) {
        string a_Verificar = verificarAses(lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7, cola_2);

        if (a_Verificar != "") {
            if (a_1 == "----") {
                a_1 = a_Verificar;
            } else if (a_2 == "----") {
                a_2 = a_Verificar;
            } else if (a_3 == "----") {
                a_3 = a_Verificar;
            } else if (a_4 == "----") {
                a_4 = a_Verificar;
            }
        }
        mostrarJuego(lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7, cola_1, cola_2, a_1, a_2, a_3, a_4);
        cout << endl;
        cout << "Seleccionar una Opcion:" << endl;
        cout << "   1. Voltear una Carta" << endl;
        cout << "   2. Tomar Carta(s)" << endl;
        cout << "   3. Regresar una Carta" << endl;
        cout << "   4. Salir" << endl;
        cin >> opcion;
        cout << endl;

        if (opcion == 1) {
            cola_1->ingresarDato(cola_2->obtenerNodoFinal()->contenido->cara);
            cola_2->eliminar();
        } else if (opcion == 2) {
            int opcion_1 = 0;

            cout << "Seleccionar de donde desea tomar la(s) Carta(s):" << endl;
            cout << "   1. Del Monton" << endl;
            cout << "   2. De las Filas" << endl;
            cin >> opcion_1;
            cout << endl;
```

```
    if (opcion_1 == 1) {
        int fila_1 = 0;
        cout << "-------------------------------------------------" << endl;
        cout << "Escriba la Fila donde desea Depositar la Carta: "; cin >> fila_1;
        cout << "-------------------------------------------------" << endl;

        if (fila_1 > 0 && fila_1 < 8) {
            Lista *listaARecibir = new Lista();
            string numeroSiguiente = "", colorSiguiente = "";
            listaARecibir = obtenerLista(fila_1, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);

            if (!listaARecibir->estaVacia()) {
                verificarInsercionDeCarta(listaARecibir, numeroSiguiente, colorSiguiente, salir);

                if (cola_2->obtenerNodoFinal()->contenido->numero == numeroSiguiente && cola_2->obtenerNodoFinal()->contenido->color
                    listaARecibir->ingresarDato(cola_2->obtenerNodoFinal()->contenido->cara);
                    listaARecibir->obtenerNodoFinal()->contenido->mostrar = true;
                    cola_2->eliminar();
                    if (!cola_2->estaVacia()) {
                        cola_2->obtenerNodoFinal()->contenido->mostrar = true;
                    }
                } else {
                    cout << "Error" << endl;
                }
            } else {
                if (cola_2->obtenerNodoFinal()->contenido->numero == "K") {
                    listaARecibir->ingresarDato(cola_2->obtenerNodoFinal()->contenido->cara);
                    listaARecibir->obtenerNodoFinal()->contenido->mostrar = true;
                    cola_2->eliminar();

                    if (!cola_2->estaVacia()) {
                        cola_2->obtenerNodoFinal()->contenido->mostrar = true;
                    }
                } else {
                    cout << "Esta Vacia" << endl;
                }
            }
        } else {
            cout << "Numero Fuera de Rango" << endl;
        }
    } else if (opcion_1 == 2) {
        int filaEmisor = 0, filaReceptor = 0, largo = 0;
        Lista *listaAMandar = new Lista(), *listaARecibir = new Lista();
        bool salir = false;

        cout << "-------------------------------------------------------------------------" << endl;
        cout << "Escriba un Numero en las siguiente Peticiones" << endl;
        cout << endl;
        cout << "Fila para tomar la(s) Carta(s): "; cin >> filaEmisor;
        cout << "Cantidad de Cartas a Tomar: "; cin >> largo;
        cout << "Fila para depositar la(s) Carta(s): "; cin >> filaReceptor;
        cout << "-------------------------------------------------------------------------" << endl;

        if (filaEmisor != filaReceptor) {
            if (filaEmisor > 0 && filaEmisor < 8 && filaReceptor > 0 && filaReceptor < 8) {
                listaAMandar = obtenerLista(filaEmisor, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);
                listaARecibir = obtenerLista(filaReceptor, lista_1, lista_2, lista_3, lista_4, lista_5, lista_6, lista_7);

                if (listaAMandar->obtenerLongitud() >= largo) {
                    if (!listaAMandar->estaVacia()) {
                        int indice = listaAMandar->obtenerLongitud() - largo, contador = 0;
                        Nodo *actual = listaAMandar->obtenerNodoInicial();

                        for (int i = 0; i < indice; i++) {
                            actual = actual->nodoSiguiente;
                        }
                        if (actual->contenido->mostrar) {
                            string numeroSiguiente = "", colorSiguiente = "";

                            for (int i = 0; i < largo; i++) {
                                //
                                verificarInsercionDeCarta(listaARecibir, numeroSiguiente, colorSiguiente, salir);
                                //
```

```
                        if (actual->contenido->numero == numeroSiguiente && actual->contenido->color == colorSiguiente) {
                            listaARecibir->ingresarDato(actual->contenido->cara);
                            listaARecibir->obtenerNodoFinal()->contenido->mostrar = true;
                            actual = actual->nodoSiguiente;
                        } else {
                            cout << "Error" << endl;
                            salir = true;
                            break;
                        }
                    }

                    if (!salir) {
                        for (int i = 0; i < largo; i++) {
                            listaAMandar->eliminar();
                        }
                        if (!listaAMandar->estaVacia()) {
                            listaAMandar->obtenerNodoFinal()->contenido->mostrar = true;
                        }
                    }
                } else {
                    cout << "No se puede trasladar Cartas Volteadas" << endl;
                }
            }
        } else {
            cout << "No es posible tomar esa cantidad de Cartas de esta Fila" << endl;
        }
    } else {
        cout << "Numero fuera de Rango" << endl;
    }
} else {
    cout << "No puede Seleccionar la Misma Fila" << endl;
}
    }
} else if (opcion == 3) {
    if (!cola_1->estaVacia()) {
        cola_2->ingresarDato(cola_1->obtenerNodoFinal()->contenido->cara);
        cola_1->eliminar();
    }
} else if (opcion == 3) {
    salir = true;
} else {
    cout << "\n-----------------------------------------" << endl;
    cout << "Numero fuera de Rango, Intentalo Nuevamente" << endl;
    cout << "-----------------------------------------" << endl;
}
}
```

Como otra parte esencial del programa está la estructura Nodo la cual guarda en su interior un objeto carta si como su nodo anterior y su nodo siguiente que son utilizados por las listas para obtener y guardar información del juego.

```
struct Nodo {
    Carta *contenido;
    Nodo *nodoSiguiente;
    Nodo *nodoAnterior;
};
```

La segunda estructura que utilizamos es el de las cartas en las cuales guardamos toda la información importante acerca de ellas siendo está el número de la carta, su color, su cara es decir la combinación entre el numero el color y la forma y por último tiene un valor booleano que nos va servir para saber si tenemos que mostrar la carta al usuario o no.

```
struct Carta {
    string numero;
    string color;
    string cara;
    bool mostrar = false;
};
```

Como la última parte del código tenemos la lista que se utilizó para el control y modificación de los nodos dentro del programa su funcionamiento se manejó tanto en el ingreso de datos como en la eliminación de los mismos, además de contar con un método para poder saber si la lista utilizada está vacía.

```
class Lista {
    private:
        Nodo *nodoInicial;
        Nodo *nodoFinal;
        int longitud = 0;

    public:
        Lista();
        void ingresarDato(string contenido);
        void eliminar();
        Nodo* obtenerNodoInicial();
        Nodo* obtenerNodoFinal();
        int obtenerLongitud();
        bool estaVacia();
};
```

El método ingresar separa en caracteres la carta seleccionada para poder guardar el número de la carta como su color para poder utilizarlo en el cambio de cartas

```
Lista::Lista() {}

void Lista::ingresarDato(string contenido) {
    Carta *carta = new Carta();
    carta->cara = contenido;

    for (int i = 0; i < contenido.size(); i++) {

        if (i == 0) {
            carta->numero = contenido[i];
        } else if (i == 3) {
            carta->color = contenido[i];
        }
    }

    if (longitud == 0) {
        nodoFinal = new Nodo();
        nodoFinal->contenido = carta;
        nodoInicial = nodoFinal;
    } else {
        Nodo *nuevo = new Nodo();
        nuevo->contenido = carta;
        nodoFinal->nodoSiguiente = nuevo;
        nuevo->nodoAnterior = nodoFinal;
        nodoFinal = nuevo;
    }
    longitud++;
}
```

El método eliminar borra el último elemento de la lista, además de verificar el estado de la lista y ver si esta se encuentra vacía.

```
void Lista::eliminar() {
    if (estaVacia()) {
        cout << "La Lista esta Vacia" << endl;
    } else if (longitud == 1) {
        nodoInicial = nodoFinal = NULL;
    } else {
        nodoFinal = nodoFinal->nodoAnterior;
        nodoFinal->nodoSiguiente = NULL;
    }
    longitud--;
}
```

Los siguientes métodos solo sirven para retornar valores de la lista.

```
Nodo* Lista::obtenerNodoInicial() {
    return nodoInicial;
}

Nodo* Lista::obtenerNodoFinal() {
    return nodoFinal;
}

int Lista::obtenerLongitud() {
    return longitud;
}

bool Lista::estaVacia() {
    return longitud == 0;
}
```