Universidad de Costa Rica

Escuela de ingeniería eléctrica

Estructuras abstractas de datos y algoritmos para ingeniería

Simulador de Cola para Parque de Atracciones

Autores:
Joseph Álvarez Sandí C20440
Joseph Castillo Torres C11795
Luis Fernando Rojas Morua B86941
Alejandro García Rucavado C03187

2 de Marzo de 2025

1. Implementación

Link del proyecto: https://github.com/J0sph/Queue-Simulator/tree/main

1.1. Manejo de Colas

Para administrar la llegada, espera y salida de los clientes del parque de atracciones, se implementaron simultáneamente estructuras de datos de tipo cola y lista enlazada. A nivel más básico, la clase `Queue` maneja la entrada y salida de personas siguiendo el principio FIFO. Adicionalmente, se creó la clase `VIPQueue`, la cual hereda a `Queue` y funciona como cola de prioridad. El propósito de estas dos clases es definir dos tipos de visitantes: los VIP y los regulares; se establece que la cola regular no puede avanzar hasta que la cola VIP esté vacía. Adicionalmente, se implementa la capacidad de manejar grupos de visitantes, que llegan y se van juntos. Para ello, cada grupo se almacena como una lista enlazada donde cada nodo corresponde al nombre de un integrante.

A continuación, se presentan las clases, atributos y métodos utilizados para la implementación

1.1.1. Nodo

La clase `Node.h` define la estructura de un visitante, sus atributos son:

- `string name`: Nombre del integrante.
- `Node* next`: Puntero que permite conectar los integrantes de un grupo en la lista enlazada.

Lo importante de esta clase son estos atributos, ya que el objetivo de un visitante es almacenar su nombre y unirse a un grupo. Para más información sobre el código de esta clase se puede consultar el archivo *Node.h* en la carpeta *Estructuras* del proyecto, el cual tiene comentarios explicando el funcionamiento del código.

1.1.2. Lista enlazada

La clase `LinkedList.h` define la estructura de los grupos de visitantes; sus atributos son:

- `int groupID`: Identificador del grupo.
- `int priority`: Prioridad del grupo.
- `Node* head`: Puntero que apunta al primer elemento de la lista.
- `Node* tail`: Puntero que apunta al último elemento de la lista.
- `LinkedList* next`: Puntero que permite conectar grupos (Listas enlazadas) en la cola.
- `int length`: Longitud de la lista enlazada (número de integrantes del grupo)

Respecto a las funciones que tiene el un grupo, se agregaron los siguientes métodos:

- `void append(const string& name)`: Este método permite agregar un integrante al grupo, recibiendo como entrada su nombre. Para esto crea un nuevo nodo con el nombre del nuevo integrante, y lo agrega al final de la lista enlazada.

- `vector<string> getMembers()`: Está función recorre todos los nodos de la lista enlazada y devuelve un vector con sus nombres. Es útil guardar los nombres de los integrantes en un vector, para usarlo luego como entrada en las funciones de la cola. Por ejemplo, al cambiar la posición de un grupo existente.
- `void showGroup()` : Recorre todos los nodos de la lista enlazada e imprime el identificador de grupo, y el nombre de cada uno de sus integrantes.

Estos métodos y atributos, muestran de manera general cómo se implementó el manejo de grupos en las colas, ya que da información sobre los datos que almacena el grupo y sus funciones, las cuáles son de utilidad para los métodos de las colas, que se muestran a continuación. Para más información sobre el código utilizado para esta clase, se puede consultar el archivo *LinkedList.h* en la carpeta *Estructuras* del proyecto, el cual contiene comentarios explicando el funcionamiento del código.

1.1.3. Cola Regular

La clase `Queue.h` define la estructura de la fila del parque, con todos los grupos de visitantes. Tiene los siguientes atributos

- `LinkedList* first`: Puntero al primer nodo de la cola.
- `LinkedList* last`: Puntero al último nodo de la cola.
- `int length`: Cantidad de grupos en la cola
- `int nextGroupID`: Identificador a asignar a un nuevo grupo. Cada vez que se agrega un grupo, este valor cambia para que todos tengan un identificador único.

Asimismo, la clase tiene los siguientes métodos:

- `bool isEmpty()`: Determina si su longitud es igual a 0; devuelve `true` o `false` según corresponda.
- `virtual void enqueue(const vector<string>& members)`: Agrega un nuevo grupo al final de la cola. Toma como entrada la lista de nombres de los integrantes del mismo. Crea una instancia de `LinkedList` para cada grupo, y aplica `append` para cada uno de los integrantes. Debe establecerse como virtual porque la clase `VIPQueue` sobrecarga la función.
- `bool joinGroup(const vector<string>& members, int groupID)`: Añade integrantes a un grupo existente. Toma como entrada los nombres de los nuevos integrantes, y el ID del grupo al que se van a añadir. Este método permite que algunos integrantes "guarden campo" para los demás miembros de su grupo.
- `void dequeue()`: Elimina el primer grupo de la cola, siguiendo el principio FIFO.
- `vector<string>deleteGroup(int groupID)`: Elimina cualquier grupo de la cola, identificado por su ID, y devuelve una lista con los nombres de sus integrantes. Aunque este método no sigue el principio FIFO, se utiliza para que los usuarios regulares tengan la posibilidad de pasarse a la cola VIP mientras hacen la fila. También se utiliza para actualizar la prioridad de los miembros de la cola VIP.

- `LinkedList* getGroup(int groupID)`: Devuelve la lista enlazada correspondiente al ID ingresado.
- `void showQueue()`: Imprime todos los integrantes de la cola.
- `LinkedList* PrimerElemento()`: Devuelve el primer grupo de la cola.
- `int getLength()`: Devuelve el número de grupos en la cola.

Estos métodos y atributos muestran de forma general el funcionamiento de la cola regular. En caso de requerir más información se puede consultar el archivo *Queue.h* en la carpeta *Estructuras* del proyecto, el cual tiene comentarios explicando el funcionamiento del código.

1.1.4. Cola VIP

La clase `VIPQueue.h` hereda a la clase `Queue` y funciona como cola de prioridad. No tiene atributos adicionales, pero tiene los siguientes métodos:

- `void enqueue(const vector<string>& members, int priority)`: Sobrecarga del método `enqueue`. Agrega un nuevo grupo y lo coloca en la posición correspondiente a la prioridad dada.
- `void updatePriority(int groupID, int newPriority)`: Elimina el grupo correspondiente y lo vuelve a agregar con la nueva prioridad.

Se puede consultar el código de esta clase en el archivo *VIPQueue.h* en la carpeta *Estructuras* del proyecto, el cual tiene comentarios explicando cada parte del código.

1.2. Interfaz Gráfica

Para la implementación de la interfaz donde será posible visualizar tanto la cola VIP, como la cola Regular, su avance y los botones agregar, eliminar y mover, se utilizó la biblioteca SFML la cual es una biblioteca multimedia que nos permite dibujar e interactuar con objetos en una ventana mediante eventos los cuales van desde presionar una tecla hasta dar click sobre una parte de la ventana para realizar una acción.

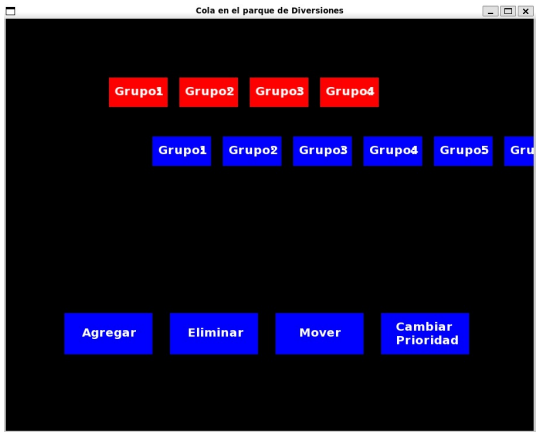

Figura 1: Demostración de la interfaz implementada.

En el siguiente apartado se darán más detalles sobre la implementación y funcionamiento de la interfaz gráfica.

1.2.1. Estructura Botón

Esta estructura nos permite crear los diferentes botones que se implementarán, con la biblioteca SFML se pueden crear rectángulos a los cuales se les pueden modificar las dimensiones con las funciones que incorpora la biblioteca como .setSize(), posición en la ventana con .setPosition(), color con .setFillColor(), y se les puede agregar texto, a este se le puede agregar una fuente con .setFont(), el tamaño de letra con .setCaracterSize(), la posición con .setPosition(), el color con .setFillColor() y el string() con .SetString(). Esta estructura incorpora dos funciones las cuales son: Draw e isClicked, la primera permite dibujar los botones en la ventana principal y la segunda nos indica si un botón fue presionado, realmente indica si se hizo click en la posición en la que se encuentra el botón.

```
// Estructura para realizar los botones
struct Boton {
    sf::RectangleShape shape; //Se hace la forma rectangular de los botones utilizando la biblioteca SFML
    sf::Text text; //Se hace el texto que tendrán los botones utilizando la biblioteca SFML

    Boton(float x, float y, const std::string &texto, sf::Font &font, float textposX, float textposY) : text(texto, font) { //Constructor de la estructura Botón
        shape.setSize({150, 70}); //Se asignan las dimensiones de los botones
        shape.setPosition({x, y}); //Se asigna la posición de cada botón
        shape.setFillColor(sf::Color::Blue); //Se asigna el color del botón

        text.setFont(font); //Se le asigna la fuente al texto de los botones
        text.setString(texto); //Texto que se va a mostrar
        text.setCharacterSize(20);  //Tamaño del texto
        text.setFillColor(sf::Color::White); //Color del texto
        text.setPosition({x + textposX, y + textposY}); //Se asigna la posición que tendrá el texto dentro del botón
    }

    void draw(sf::RenderWindow &window) { //Función para dibujar la forma y texto de los botones en la interfaz
        window.draw(shape);
        window.draw(text);
    }

    bool isClicked(sf::Vector2f mousePos) { //Función que indica si el botón fue presionado
        return shape.getGlobalBounds().contains(mousePos);
    }
};
```

Figura 2: Struct Boton

1.2.2. enum Estado

Este tipo de dato nos permite llevar un control del flujo de ejecución del programa, para realizar diferentes ejecuciones dependiendo de las acciones del usuario, INICIAL es cuando no se ha realizado ninguna acción y se está a la espera de que el usuario realice algún movimiento, SELECCIONAR_TIPO es cuando el usuario presionó el botón de agregar y se le muestran los botones VIP y Regular para que seleccione que tipo de grupo a agregar, INGRESAR_NOMBRE es cuando el usuario está digitando los nombres de los integrantes del grupo que va a añadir, ELIMINAR_GRUPO es cuando el usuario dijitó el botón de eliminar, INGRESAR_GRUPO es cuando el usuario va a eliminar o mover un grupo de la cola Regular a la VIP, INGRESAR_PRIORIDAD es cuando el usuario va a agregar un nuevo grupo o cambia la prioridad de un grupo VIP ya existente y se le pregunta por la prioridad que tendrá el nuevo grupo o la nueva prioridad del grupo ya

existente, y por último MOVER_GRUPO es cuando el usuario presionó en botón de mover para mover un grupo de la cola Regular a la VIP.

```
enum Estado { //Estados para controlar el flujo de ejecución
    INICIAL, //Para volver al inicio cuando no se ha seleccionado ningún botón
    SELECCIONAR_TIPO,  //Se seleccionó el botón agregar
    INGRESAR_NOMBRE,   //Introduciendo el nombre de los miembros del grupo
    ELIMINAR_GRUPO, //Se seleccionó el botón eliminar
    INGRESAR_GRUPO, //Se seleccionó el botón eliminar y se pidió al usuario que dijite el grupo al que se le desea realizar la acción
    INGRESAR_PRIORIDAD, //Se seleccionó agregar Vip y se le pide al usuario que dijite la prioridad
    MOVER_GRUPO, //Se seleccionó el botón mover
    CAMBIAR_PRIORIDAD // Se seleccionó el botón cambiar prioridad y se le pide al usuario número de grupo y prioridad
};
```

Figura 3: Tipo de dato enum que nos permite llevar un flujo de ejecución

1.2.3. Estructura movimiento

Esta estructura está hecha para controlar el movimiento de las colas a la hora de realizar la animación de que avanza la cola y el grupo entra, mediante el boleano enMovimiento, se crean dos objetos movVIP para saber si la cola VIP se está moviendo y otro movRegular para la cola Regular. Justo debajo de esta estructura se encuentra una variable llamada velocidadMovimiento la cual controla la velocidad a la que se realiza la animación de movimiento hasta desaparecer de los rectángulos (Grupos).

```
//Estructura para realizar la animación de que los grupos se desplazan
struct Movimiento {
    bool enMovimiento = false; //Estado que indica si el grupo se está desplazando
    float offset = 0;
};

Movimiento movVIP, movRegular;
float velocidadMovimiento = 0.1f; // Velocidad en pixeles por frame de los grupos
```

Figura 4: Estructura que indica que controla si una cola se está moviendo

1.2.4. Funciones

En esta sección se hablará de que hacen y como funcionan cada una de la funciones mencionadas a continuación.

- **AgregarElementoACola**: Esta función discrimina entre usuarios VIP y regulares y crea dos colas, esto lo hace mediante el parámetro bool VIP, esta función también recibe un vector de string donde se encuentran los miembros del grupo, y ademas recibe un parámetro llamado Prioridad el cual es indicado por el usuario mediante la interfaz en el caso de que el grupo sea VIP, si los grupos son regulares todos tienen la misma prioridad.

```
//Función para agregar los grupos a la cola VIP o regular en base a un parámetro
void AgregarElementoACola(bool VIP, const vector<string>& miembros, int Prioridad){
    if (VIP == true){
        CVIP.enqueue(miembros, Prioridad);
    }else{
        CRegular.enqueue(miembros);
    }
}
```

Figura 5: Función que permite agregar un grupo a la cola VIP o Regular

- **ColaPorPrioridad**: Esta función se encarga de inicializar el movimiento de la cola VIP, e iniciar el movimiento de la cola Regular con movVIP y movRegular respectivamente hasta que la cola VIP esté vacía mediante dos condicionales y verificando que la cola VIP esté vacía.

```
//Función para que la cola regular se comience a desplazar hasta que la cola VIP esté vacía
void ColaPorPrioridad() {
    if (!CVIP.isEmpty() && !movVIP.enMovimiento) {
        movVIP.enMovimiento = true;
        movVIP.offset = 0;
    } else if (CVIP.isEmpty() && !CRegular.isEmpty() && !movRegular.enMovimiento) {
        movRegular.enMovimiento = true;
        movRegular.offset = 0;
    }
}
```

Figura 6: Función que empieza el movimiento de las colas y empieza el movimiento de la cola Regular hasta que la cola VIP se halla vaciado

- **CalcularTiempoEspera**: Esta función recibe la posición del grupo en la cola y mediante el tiempo que tarda cada grupo en entrar calcula el tiempo estimado de atención para cada grupo y retorna este valor para posteriormente cuando el usuario haga click sobre cada grupo pueda visualizar esta información.

```
//Función para calcular el tiempo de espera en base a la posición del grupo en la cola y el tiempo que tarda cada grupo en avanzar
int CalcularTiempoEspera(int posicion){
    int TiempoPorGrupo = 5;
    return posicion * TiempoPorGrupo;
}
```

Figura 7: Función que calcula el tiempo de espera

- **DetallesGrupos**:Esta función de Detallesgrupos a grandes rasgos lo que hace mostrar detalles de los grupos ya sea en la cola VIP o regular en una ventana SFML, con información

de los miembros del grupo el tiempo estimado y la posición de dicho grupo.
Primero selecciona la cola dependiendo el grupo y regular eso se realiza mediante un operador, luego se obtiene el grupo correspondiente a través de cola.getGroup(groupID), pero si el grupo no existe no realiza ninguna acción adicional. Si el grupos válidos obtiene la lista de los miembros mediante el método getMembers que de igual un vector de cadenas además de ellos se calcula el tiempo estimado de espera para el grupo usando CalcularTiempoEspera(groupID). Se crea el objeto de texto para demostrar la ventana asignándole las propiedades visuales como la fuente el tamaño el color y la posición, se construye que contiene información relevante del grupo la cual va a presentar y se agreguen dinámicamente mediante el ciclo for. Cómo de costumbres se limpia la ventana se dibuja el texto y finalmente se actualiza la pantalla y también se encuentra un método dentro del bucle de eventos que espera que presione la tecla para interrumpir el bucle y finalizar la función permitiéndose el usuario y regresar al flujo principal del programa.

```
//Función para visualizar desde la interfaz la información de cada grupo
void DetallesGrupos(int groupID, bool isVIP, sf::RenderWindow& window, sf:: Font& font){
    Queue& cola = isVIP ? CVIP : CRegular;
    LinkedList* grupo = cola.getGroup(groupID);

    if (grupo != nullptr) {
    vector<string> members = grupo->getMembers(); //Se obtienen los miembros del grupo seleccionado
    int TiempoEstimado = CalcularTiempoEspera(groupID); //Se llama la función que calcula el tiempo de espera

    sf::Text DetallesGrupo; //Se crea el texto que se verá en la interfaz con la información
    DetallesGrupo.setFont(font); //Se le asignan las caracteristicas al texto
    DetallesGrupo.setCharacterSize(21);
    DetallesGrupo.setFillColor(sf::Color::White);
    DetallesGrupo.setPosition(300, 150);

    string texto = "Grupo: " + to_string(groupID) + "\n"; //String con la información
    texto += "Tiempo estimado de espera: " + to_string(TiempoEstimado) + " minutos\n";
    texto += "Integrantes: \n";
    for (const string& member : members) { //Se agregan los miembros
        texto += member + "\n";
    }
    DetallesGrupo.setString(texto);

    window.clear(); //Se limpia la pantalla
    window.draw(DetallesGrupo); //Se dibuja el texto
    window.display(); //Se muestra en pantalla

    sf::Event event; //Se crea el evento para mostrar la infomación
    while (window.waitEvent(event)) {
        if (event.type == sf::Event::KeyPressed) {
            break;
        }
    }

    }
}
```

Figura 8: Función que dibuja

- **Elementos**: Esta función dibuja los grupos mediante rectángulos de colores, primero en la parte superior dibuja la cola VIP con rectángulos de color rojo y en la parte inferior dibuja la cola Regular con rectángulos azules, esta función recibe ambas colas por referencia,

la posición en el eje Y de cada una la cual va a variar de una cola a otra, el color y la variable de movimiento mediante referencia, primero verifica que las colas no esté vacías, después se realiza un puntero temporal que apunte al primer nodo de la lista enlazada el cual corresponde al primer grupo, esto lo hace para recorrer toda la lista y dibujar todos los grupos, se instancia una variable x, la cual indica la posición de cada grupo en el eje X, esta variable permite tanto distanciar los distintos grupos como simular que las colas se mueven en este eje, después de esto se carga la fuente y se comprueba que esta halla cargado correctamente, y posteriormente se procede a caracterizar tanto el rectángulo de cada grupo como el texto que indica en número de grupo de cada uno y se dibuja en la interfaz mediante window.draw, se actualiza la variable x con un ancho previamente establecido, esta será la separación entre grupos en la cola. Se recorren todos los nodos (Grupos) de la lista para dibujarlos y por último mediante un condicional que se activa cuando la cola está en movimiento es que se realiza el avance de la cola hasta llegar a cierto punto donde el grupo es eliminado simulando que entró al parque o a la atracción.

```
//Función para dibujar los rectángulos(Grupos)
void Elementos(sf::RenderWindow &window, Queue &queue, float y, sf::Color color, Movimiento &mov) {
    if (queue.isEmpty()) return; //Comprueba si las colas están vacías

    LinkedList* temp = queue.PrimerElemento(); //Seleccionada el primer elemento de la lista enlazada (Primer grupo)
    float x = 250 - mov.offset; // Se aplica el desplazamiento con mov.offset cuando enMovimiento sea activado pasado un tiempo seleccionado

    sf::Font font; //Se carga la fuente a utilizar
        if (!font.loadFromFile("assets/arial.ttf")) {
            cout << "No cargó la fuente correctamente" << endl;
            return;
        }

    while (temp != nullptr) { //Se recorren los nodos de la lista enlazada para dibujar cada grupo
        sf::RectangleShape Elemento(sf::Vector2f(AnchoElemento, LargoElemento)); //Se asignan las caracteristicas de los rectangulos
        Elemento.setPosition(sf::Vector2f(x, y));
        Elemento.setFillColor(color);
        //Se asignan las características del texto correspondiente al número de grupo de cada rectángulo
        sf::Text numero;
        numero.setFont(font);
        string NG = to_string(temp->groupID);
        numero.setString(NG);
        numero.setCharacterSize(20);
        numero.setFillColor(sf::Color::White);
        numero.setPosition(sf::Vector2f(x+80, y+10));

        //Se asignan las características del texto correspondiente a "Grupo: " Mostrado en cada rectángulo
        sf::Text Grupo;
        Grupo.setFont(font);
        string G = "Grupo: ";
        Grupo.setString(G);
        Grupo.setCharacterSize(20);
        Grupo.setFillColor(sf::Color::White);
        Grupo.setPosition(sf::Vector2f(x+10, y+10));

        window.draw(Elemento); //Se dinuja el rectángulo
        window.draw(numero); //Se dibuja el níumero
        window.draw(Grupo); //Se dibula la string

        x += AnchoElemento + DistanciaEntreElementos;
        temp = temp->next;
    }
```

Figura 9: Función que dibuja y realiza la animación de las colas en movimiento

- **upgradeToVip**: Esta función permite realizar un upgrade de la cola Regular a la cola VIP, esto lo logra eliminando el grupo de la cola Regular que se va a mover y añadiéndolo a la cola VIP.

```
// Metodo que permite que un grupo regular haga 'upgrade' a vip
void upgradeToVip(int groupID, int prioridad){
    // Borra el grupo de la cola regular
    vector<string> integrantes = CRegular.deleteGroup(groupID);

    // Ingresa el grupo en la VIP
    CVIP.enqueue(integrantes, prioridad);

}
```

Figura 10: Función que permite hacer Upgrade de la cola Regular a la VIP

1.2.5. Main

Aquí se encuentra primero el vector nombres en el cual se guardarán los nombres de los integrantes de los grupos al ser añadidos, este vector se llama names, unas variables para la prioridad en la cola VIP llamada Priority, para saber que tipo de grupo es si VIP o Regular, número de grupo y se inicializa el estado INICIAL. Inmediatamente después de esto utilizando sf::RenderWindow se crea la ventana donde se visualizará todo, con sf::VideoMode se asigna el ancho y largo de la ventana, con sf::Clock se inicializa un reloj para tener un control sobre la ejecución de ciertas acciones y para la estimación del tiempo de atención o de espera, se carga la fuente del texto de los botones con sf::Font y se comprueba que haya cargado correctamente. Se inicializan los diferentes botones de la interfaz con sus características, las características del texto de cada uno y el mensaje con la instrucción que se le indicará al usuario mediante la interfaz al precionar un botón, posteriormente se asignan varias variables, la primera bool CapturandoTexto que permite tener un control de cuando el usuario está digitando texto, una variable bool mostrarMensajeInstruccion para controlar cuando se debe mostrar el mensaje de Instrucción, otra variable string inputText para guardar el texto que va digitando el usuario, y con sf::Time actual y actual = relog.getElapsedTime() se lleva un control del tiempo.

Posterior a eso si utilizas una estructura de control while la cual la condición es que se ejecute continuamente mientras la ventana está abierta, seguidamente a una variable llamada actual se le asigna el tiempo transcurrido desde que el reloj comenzó a contar y luego se declara una variable para manejar eventos de la ventana que en cuanto a eventos son las situaciones que se van a presentar en interfaz.

La siguiente estructura de control extrae los eventos que ocurren en la ventana y los almacena en la variable event y esto se ejecuta mientras haya eventos pendientes, Además se compró que

si el evento indica que a la ventana fue cerrada se cierra la ventana finalizando el bucle principal. Seguidamente se verifica si el evento que está sucediendo corresponde a una entrada de texto además asegurando que capturandoTexto sea true para asegurar de que se está en modo escritura ; se procede a convertir el código del carácter ingresado en un valor char permitiendo obtener el carácter real de lo que se escribió. Se comprueba por medio del condicional letra == $'b'$ Si el usuario presiona la tecla espacio y en conjunto con un booleano & se asegura que en la entrada no esté vacía para evitar errores finalmente para eliminar el último carácter de la cadena simulando el borrado de texto y una caja de entrada. Nuevamente con el condicional letra == r' Se verifica si el usuario presionó enter para ingresar, por medio de un if se comprueba si el sistema está en el estado de ingresar nombres y si es así se crea un flujo de entrada a partir de la cadena que se ingresó declarando una variable para almacenar cada palabra extraída en donde Para ello el siguiente estructura de control while y trae cada palabra de la entrada y la agrega el vector names posterior a ello se desactiva el modo de captura de texto y se procede a borrar el contenido para limpiar evitando problemas para futuras entradas.

Se verifica que es el grupo que se está registrando es el VIP se activa el estado de ingresar prioridad en donde se lo solicita la prioridad que va a tomar ese grupo además de ellos se activa la captura de texto y la presentación de mensajes, pero si no es así, nos indica que es un grupo regular por lo que no es necesario asignarle una prioridad a la función para agregar al grupo a la cola después de eso se vacía el vector names, se cancela la captura de texto y se regresa al estado inicial, se cancela la presentación de los mensajes y se limpia donde se almacenan los mensajes; este procedimiento se realiza de manera repetitiva por lo cual posteriormente le llamaremos solamente como limpieza.

Posteriormente se observa de qué si el estado actual es ingresar prioridad el usuario e ingresar un número para definir esa prioridad por lo cual se intenta convertir esta variable en un número entero usando el comando stoi, si la conversión resulta exitosa se agrega el grupo VIP con su prioridad llamando al método agregar elemento a cola con sus respectivos parámetros y se procede a hacer la limpieza que antes se explicó. Si la comercial falla se muestra un mensaje de error y se limpia la entrada.

Seguido se observa lo que sucede si el usuario está en el estado de ingresar al grupo lo que indica que se está procesando la eliminación del grupo, el texto ingresado lo convierte un entero para obtener el número del grupo el cual se desea ahorrar y si es un grupo VIP tiene su método distintivo para eliminarlo al igual que el caso contrario al ser regular también cuenta con un método específico para eliminar el grupo para después de ello realizar la limpieza detallada anteriormente.

El siguiente bloque que se ejecuta es cuando se encuentra en el estado de mover el grupo, primeramente se convierte el texto ingresado a un numero entero representando un número del grupo; posterior a ello se elimina el grupo con el número indicado de la cola regular pero se obtiene sus integrantes y se guarden un vector así que verificando que existan estos integrantes se procedan de una prioridad de uno y moverlos a la cola VIP usando el método desarrollado para ello, para finalmente realizar la limpieza.

El siguiente caso es cuando se encuentra en el estado de cambiar prioridad en donde igual-

mente la entrada se convierte en números enteros asignando número al grupo y prioridad se llama al CVIP.updatePriority(numgrupo, prioridad) para buscar el grupo identificado en la cola VIP y asignarle una nueva prioridad procede a hacer limpieza pero sí no pueden convertirse o se encuentra algún error en este proceso se muestra un mensaje y luego se limpia el inputText.

Asimismo cuando ninguna condición especial está activa cada letra que el usuario ingrese añade el contenido existente y se actualiza un elemento visual en interfaz para mostrar en tiempo real lo que el usuario está escribiendo siendo útil para el retroalimentación inmediata al usuario.

Luego con la estructura de control Eve y el condicional que busca un evento que registra si el usuario presiono uno de los botones del ratón se utilizará para el uso un interfaz capturando la posición del ratón relativo con respecto a la ventana activa usando las coordenadas de los pixeles por medio de un Victor que cuenta con dos enteros representando la posición X y Y; en razón detecta cuando el usuario hace clic en la ventana y registra exactamente donde se encuentra el cursor cuando hizo ese clic para manejar interacciones con interfaz. Seguido se encuentra el comportamiento de cada botón presenta la interfaz, con eso nos referimos a qué estado se debe acudir al presionar cierto botón y si en ese momento ocurre la necesidad de capturar texto o mostrar un mensaje como por ejemplo al hacerte en el botón de agregar va al estado de seleccionar tipo en donde no es necesario la captura de texto ni mostrar un mensaje en ese preciso instante de ahí en adelante lo demás se encarga y ha sido ocurre con los diversos como el de eliminar de mover o en cambio prioridad cada uno con sus características determinadas.

En este mismo bloque una sección posterior a el análisis de su botones se encuentra una estructura de código en la cual detectas el usuario coloca el cursor sobre un grupo, primeramente en la cola VIP en donde si se colocan encima del grupo en la cola que se encuentra en movimiento lo presiona dándole clic al mouse calcula su tiempo de espera y muestra los detalles como los miembros y la posición en la que se encuentra para ello se utilizó una estructura for itera sobre todos los presentes en la cola y luego se calculó la posición de cada grupo en función de su índice en la cola y en una posición base inicial controlada además de ello por el desplazamiento animado que fue programado. Se crea los rectángulos que representará visualmente el área del grupo en la cola para si se coloca el modo sobre esta posición significa que se está indicando un grupo y al hacerse recuperar la información del grupo correspondiente esto por medio de un stream que recopila en imprime los datos Actualizando el mensaje en interfaz con la información del grupo. Para la cual regular se utilizó exactamente la misma lógica solamente que varía es esa característica.

Cómo antes mencionó el presionar botones se traslada a otro estado el cual tiene otras métodos de transmisiones entre los cuales de seleccionar tipo despliega botones VIP y el regular que al ser presionados van al Estado de ingresar el nombre dando apertura en la captura y al mostrar mensajes correspondientes para cada uno de ellos. En cuanto al estado eliminar grupo nuevamente se nos despliega otro botones de pie irregular que nos llevarán al Estado ingresar al grupo activando la captura de texto y la presentación de los mensajes de instrucción en ambos casos es igual variando solamente los mensajes. En el estado de mover grupo y inmediatamente se apertura a las capturas de texto y a la presentación del mensaje siendo igual a lo realizado al cambiar la prioridad con eso me referimos al estar en ese estado.

Luego encontramos una estructura en la cual se espera un tiempo determinado para activar una animación de movimiento que con esto lo que se realiza es la eliminación del primer grupo y al realizar esto reinicia el tiempo para volver a hacer el mismo procedimiento y mantenerse en esa iteración constante.

Nos encontramos con la limpieza de ventana es decir la pantalla para posterior a ellos dibujar visualmente las dos colas VIP irregular en posiciones fijas con diferentes colores y también colocarle los botones interactivos en la ventana que es una parte esencial para mostrar la interfaz en el usuario; encontramos un condicional que nos busca si central estado de seleccionar tipo o eliminar el grupo para ver si es necesario dibujar los botones de VIP y regular; manteniéndonos en el ambiente de la interfaz el otro condicional muestra de qué manera se dibuja el mensaje de instrucción que debe seguir el usuario al estar ingresando el texto a través de interfaz y finalmente se captura ese texto que va ingresando el usuario.

2. Resultados

Al correr el programa se muestra la siguiente interfaz:

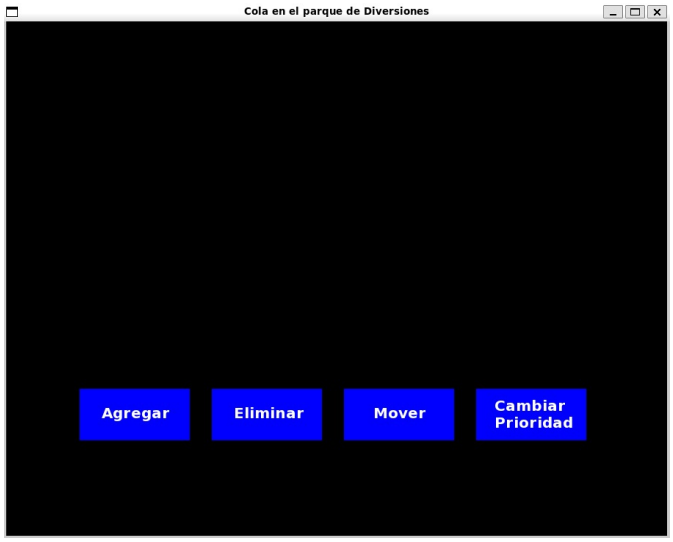

Figura 11: Apariencia de la interfaz al iniciar el programa.

El botón Agregar solicita a cuál cola se va a agregar el grupo:

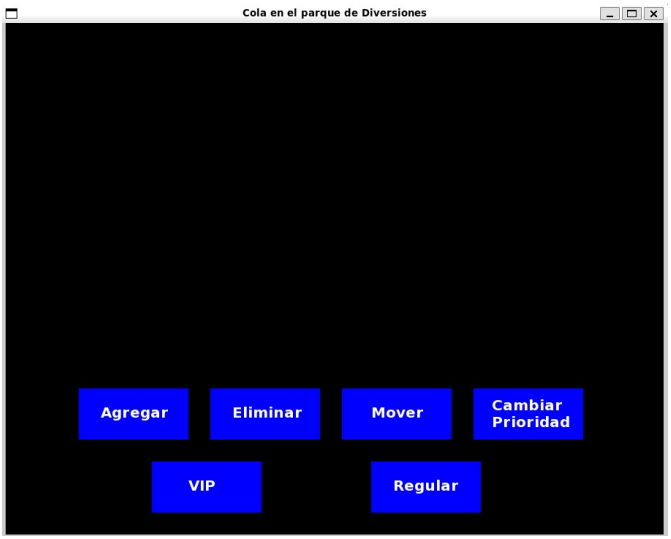

Figura 12: Botón agregar.

Esta opción permite ingresar el nombre de los integrantes del grupo:

Figura 13: Solicitud de nombre de integrantes para agregar un grupo.

Y en la cola VIP además de los integrantes solicita la prioridad:

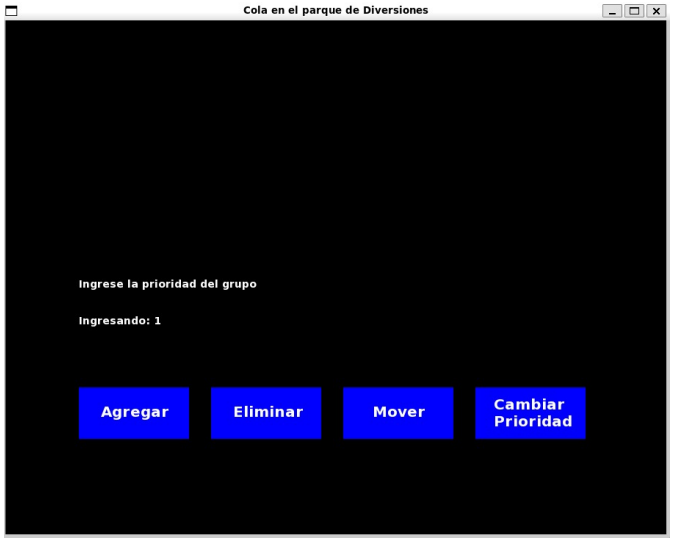

Figura 14: Solicitud de prioridad para agregar un grupo en la cola VIP.

Al completar el proceso de agregar, este es el resultado:

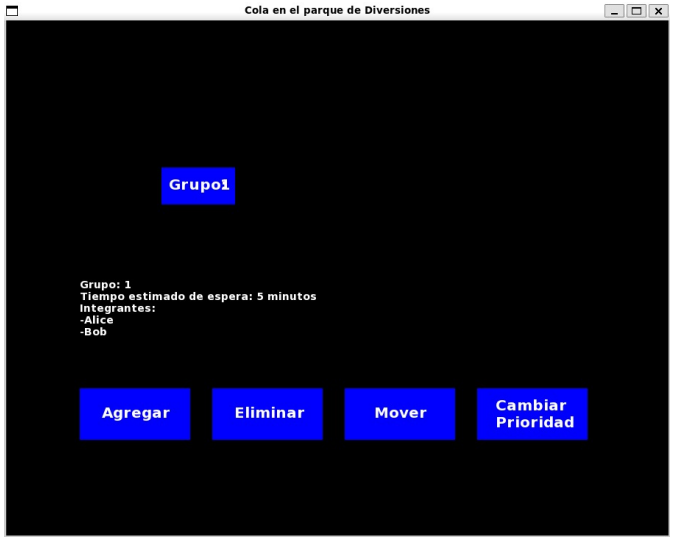

Figura 15: Resultado de la opción agregar.

El botón eliminar, de igual forma solicita de cuál cola se va a eliminar, y solicita el número de grupo:

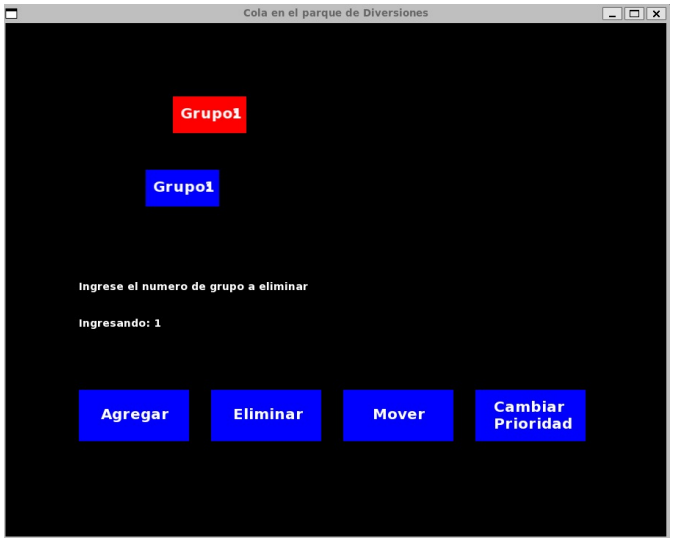

Figura 16: Solicitud de identificador de grupo para eliminarlo.

Al ingresar el identificador del grupo lo elimina de la cola correspondiente:

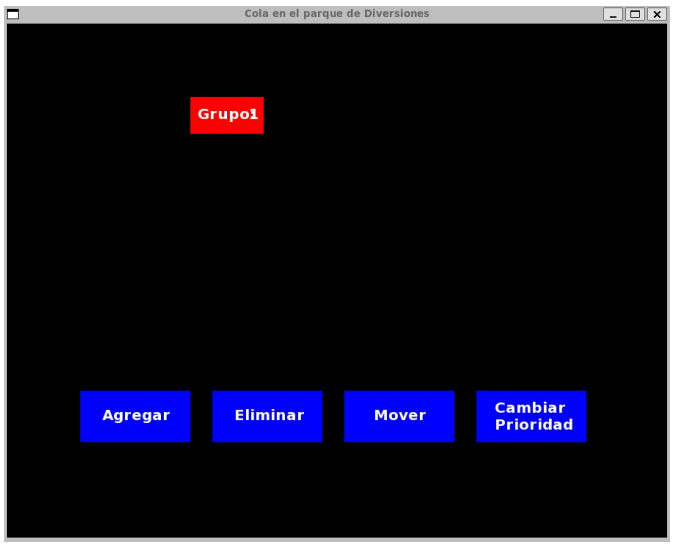

Figura 17: Resultado de la opción de eliminar.

El botón mover igual solicita el identificador de grupo:

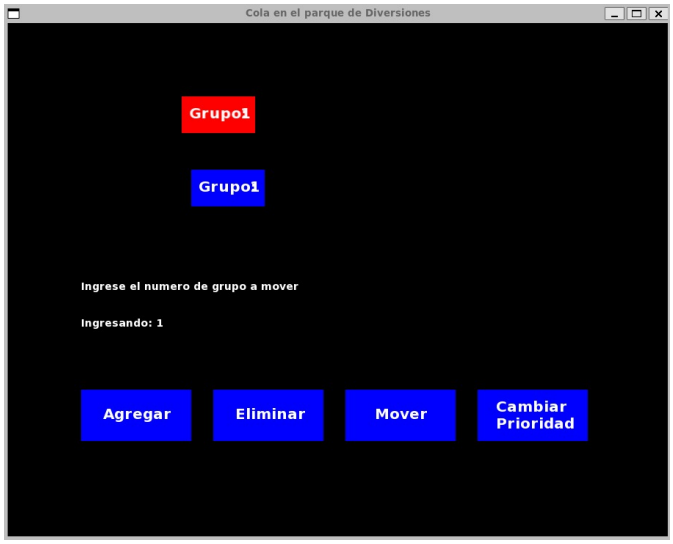

Figura 18: Botón Mover.

Al ingresarlo, mueve el grupo al final de la cola VIP:

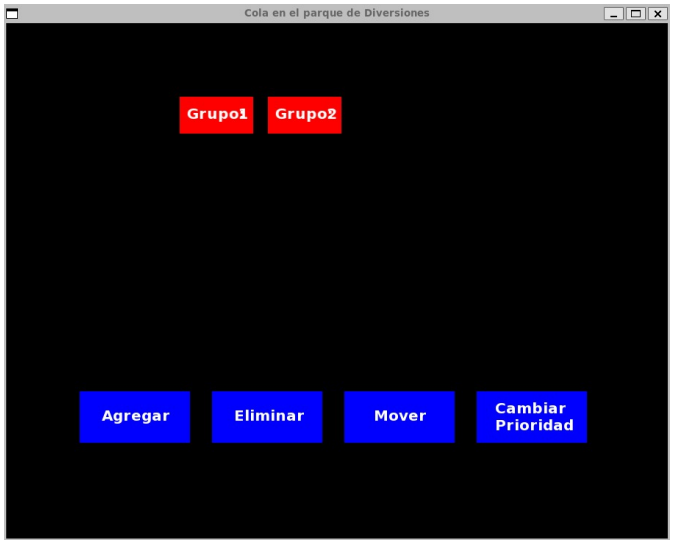

Figura 19: Resultado de la opción mover.

El botón de cambiar prioridad solicita el identificador de grupo y su nueva prioridad:

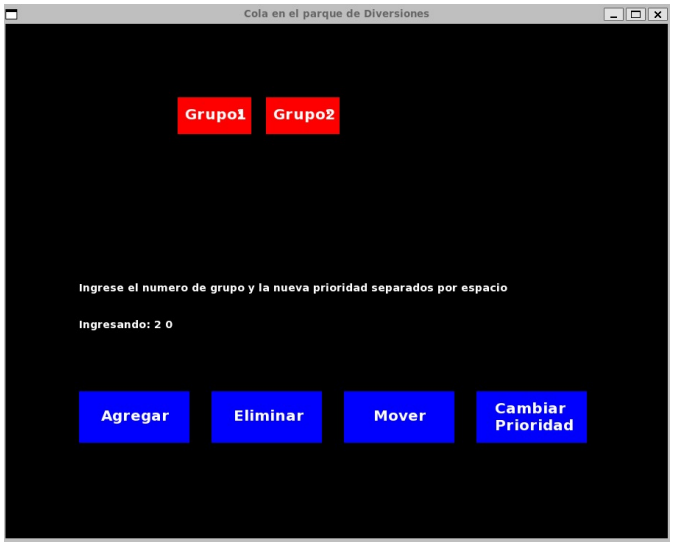

Figura 20: Botón Cambiar prioridad.

Al ingresar los datos cambia la posición del grupo según su prioridad:

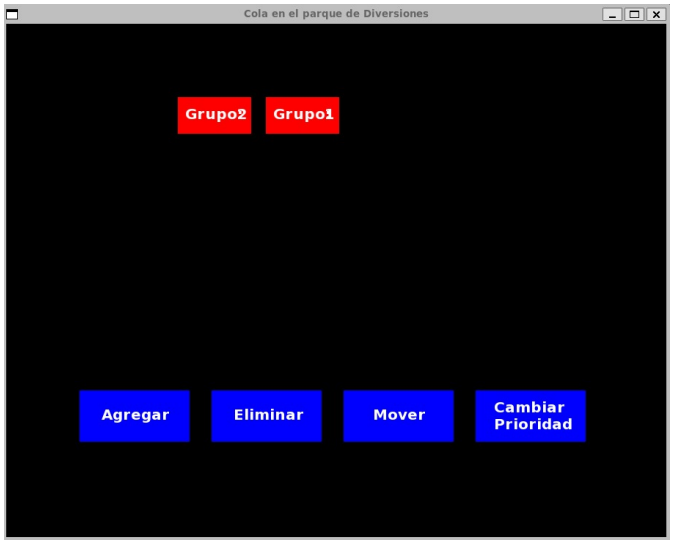

Figura 21: Resultado de la opción de cambiar prioridad.