Simulador de Gestión de Aeropuerto

Este código simula la gestión de un aeropuerto, donde los aviones aterrizan en pistas disponibles, se asignan puertas para desembarque y se gestionan las prioridades de aterrizaje según categorías de aviones.

Descripción General

La simulación implica la llegada de aviones de tres categorías (A, B, C) a un aeropuerto. Los aviones deben aterrizar en pistas disponibles, luego ser asignados a puertas de desembarque. Existen prioridades de aterrizaje definidas por el estado del aeropuerto, que se pueden modificar mediante una conexión TCP.

Objetivos

- Gestionar aviones que llegan a un aeropuerto con diferentes categorías y prioridades.
- Asignar pistas para aterrizajes y puertas para desembarque.
- Controlar las prioridades de aterrizaje con base en el estado actual del aeropuerto.
- Gestionar la comunicación entre un servidor y un sistema de control de tráfico aéreo a través de TCP.

Estructura del Proyecto

El código está estructurado en varias funciones y tipos para manejar las operaciones del aeropuerto, las colas de aterrizaje, las prioridades y la comunicación:

Tipos principales

- **Plane**: Representa un avión con su identificador, número de pasajeros, categoría y hora de llegada.
- **Runway**: Representa una pista del aeropuerto.
- **Gate**: Representa una puerta de embarque del aeropuerto.
- **PriorityQueue**: Una cola de prioridad para gestionar los aviones según su categoría y hora de llegada.

Aeropuerto (Airport)

La estructura `Airport` es la que contiene la información y el estado del aeropuerto. Esta estructura maneja los aviones en espera de aterrizaje, las pistas disponibles, las puertas y la prioridad de aterrizaje.

Funciones Principales

`main()`

La función principal (`main()`) inicializa el aeropuerto, configura las pistas y puertas, y lanza varias goroutines para manejar las simulaciones. También establece la conexión TCP y coordina la comunicación con el sistema de control de tráfico aéreo.

- **Inicialización de variables**: Establece el número de aviones en cada categoría (A, B, C), las pistas (`runways`), y las puertas (`gates`).
- **Conexión TCP**: Conecta el aeropuerto con un sistema de control a través de TCP.
- **Lanzamiento de goroutines**: Ejecuta funciones concurrentes para generar los aviones, gestionar los aterrizajes, y monitorizar el progreso.

`generateInitialAirplanes()`

Genera una lista de aviones con diferentes categorías (A, B, C) y los encola en la cola de aterrizajes del aeropuerto. Los aviones son generados de manera aleatoria y luego mezclados para simular un orden de llegada impredecible.

- **Entrada**: Número de aviones por categoría (A, B, C).
- **Salida**: Ninguna. Los aviones se encolan en la cola del aeropuerto.

`enqueuePlane()`

Agrega un avión a la cola de aterrizaje del aeropuerto si hay espacio disponible. Si la cola está llena, el avión es rechazado.

- **Entrada**: Un avión de tipo `Plane`.
- **Salida**: `true` si el avión fue añadido a la cola, `false` si la cola estaba llena.

`handleMessages()`

Maneja los mensajes recibidos a través de la conexión TCP. Los mensajes alteran el estado del aeropuerto, lo que puede afectar las prioridades de aterrizaje y otras condiciones.

- **Entrada**: Mensajes TCP del tipo `string` que representan códigos de estado.
- **Salida**: Ninguna. Actualiza el estado del aeropuerto y su prioridad.

`processMessage()`

Procesa un mensaje recibido por `handleMessages()` y actualiza el estado del aeropuerto en consecuencia. Cada código de mensaje corresponde a una acción o cambio en la prioridad de aterrizaje.

- **Entrada**: Un mensaje de tipo `string`, que representa un código numérico.
- **Salida**: Actualiza el estado y prioridad del aeropuerto.

`manageRunways()`

Gestiona los aterrizajes de los aviones, tomando en cuenta el estado y la prioridad actual del aeropuerto. Intenta asignar aviones a pistas disponibles según la prioridad establecida.

- **Entrada**: Ninguna. Usa la cola del aeropuerto y las pistas disponibles.
- **Salida**: Inicia una goroutine para manejar el aterrizaje de un avión.

`handleLanding()`

Maneja el aterrizaje de un avión en una pista. Una vez aterrizado, el avión es asignado a una puerta de embarque si está disponible.

- **Entrada**: Un avión y una pista disponible.
- **Salida**: Asigna el avión a una puerta y actualiza las variables del aeropuerto.

`handleGate()`

Maneja el desembarque de un avión en una puerta específica. Una vez completado el desembarque, la puerta se libera.

- **Entrada**: Un avión y una puerta.
- **Salida**: Libera la puerta y actualiza el estado del aeropuerto.

`monitorCompletion()`

Monitorea el estado de la simulación y verifica si todos los aviones han aterrizado y sido asignados a puertas. Cuando la simulación finaliza, se envía una señal de completado.

- **Entrada**: Ninguna. Verifica el estado del aeropuerto.
- **Salida**: Señaliza el fin de la simulación si todos los aviones han sido atendidos.

`assignPassengers()`

Asigna un número aleatorio de pasajeros a un avión según su categoría. Los aviones de categoría A tienen más pasajeros que los de categoría B, y estos más que los de categoría C.

- **Entrada**: La categoría del avión.
- **Salida**: Un número aleatorio de pasajeros.

Estados del Aeropuerto

El estado del aeropuerto está controlado por códigos numéricos que determinan el comportamiento del sistema:

- **0**: Aeropuerto Inactivo.
- **1**: Solo categoría A puede aterrizar.
- **2**: Solo categoría B puede aterrizar.
- **3**: Solo categoría C puede aterrizar.
- **4**: Prioridad de aterrizaje para categoría A.
- **5**: Prioridad de aterrizaje para categoría B.
- **6**: Prioridad de aterrizaje para categoría C.
- **9**: Aeropuerto cerrado temporalmente.

Conexión TCP

El sistema también puede recibir mensajes de control a través de una conexión TCP, lo que permite cambiar el estado del aeropuerto. Los mensajes son códigos numéricos que ajustan la prioridad de aterrizaje y el comportamiento de la simulación.

Concurrencia

El programa está diseñado para ser altamente concurrente, utilizando goroutines para gestionar:

- La generación y encolado de aviones.
- El manejo de aterrizajes en pistas.
- La asignación de puertas de embarque.
- El monitoreo del progreso de la simulación.

Los bloqueos (`sync.Mutex`) se utilizan para asegurar que las operaciones sobre las estructuras compartidas (como la cola de aterrizajes) sean seguras en un entorno concurrente.

Consideraciones

- **Escalabilidad**: El código está diseñado para ser escalable, permitiendo cambiar el número de pistas y puertas.
- **Simulación Realista**: Se utilizan tiempos de espera aleatorios para simular la incertidumbre en las operaciones del aeropuerto.
- **Manejo de Prioridades**: La prioridad de aterrizaje se maneja de manera flexible y se ajusta en tiempo real mediante mensajes.

Informe de Pruebas del Sistema de Gestión Aeroportuaria (Tests)

Configuración Base

- **Pistas disponibles**: 3
- **Puertas disponibles**: 10
- **Capacidad máxima de la cola**: 50 aviones.
- **Prioridades de aterrizaje**:
  - Categoría A: Mayor prioridad.
  - Categoría B: Prioridad intermedia.
  - Categoría C: Menor prioridad.

Pruebas Realizadas y Resultados Detallados

Prueba 1: Distribución Equitativa

Parámetros Iniciales

- **Aviones de categoría A**: 10
- **Aviones de categoría B**: 10
- **Aviones de categoría C**: 10
- **Total de aviones**: 30

Observaciones Detalladas

- **Tiempo total de simulación**: 45 segundos.
- **Tiempos promedio por categoría**:
  - Categoría A: 3.5 segundos por avión.
  - Categoría B: 4.5 segundos por avión.
  - Categoría C: 5.2 segundos por avión.
- **Utilización de recursos**:
  - Las pistas estuvieron ocupadas el **90% del tiempo**.
  - Las puertas estuvieron ocupadas el **90% del tiempo**.
- **Comportamiento observado**:
  - Los aviones de categoría A aterrizaron más rápido debido a su mayor prioridad.

- Las categorías B y C tuvieron tiempos de espera similares al inicio, pero los tiempos de C aumentaron hacia el final.
- La cola alcanzó un máximo de **30 aviones**, sin superar la capacidad máxima.

Resumen

Este escenario mostró un equilibrio adecuado en la distribución de recursos, con una ocupación eficiente y tiempos razonables de espera para todas las categorías. Es ideal para operaciones con demandas balanceadas.

Prueba 2: Mayor Proporción de Aviones de Categoría A

Parámetros Iniciales

- **Aviones de categoría A**: 20
- **Aviones de categoría B**: 5
- **Aviones de categoría C**: 5
- **Total de aviones**: 30

Observaciones Detalladas

- **Tiempo total de simulación**: 40 segundos.
- **Tiempos promedio por categoría**:
  - Categoría A: 2.8 segundos por avión.
  - Categoría B: 6.2 segundos por avión.
  - Categoría C: 7.1 segundos por avión.
- **Utilización de recursos**:
  - Las pistas estuvieron ocupadas el **100% del tiempo**.
  - Las puertas estuvieron ocupadas el **95% del tiempo**.
- **Comportamiento observado**:
  - Los aviones de categoría A dominaron los aterrizajes gracias a su alta prioridad y mayor cantidad.
  - Los aviones de categorías B y C experimentaron tiempos de espera significativamente mayores debido a su menor prioridad.
  - La cola alcanzó un máximo de **30 aviones**, igual que en la prueba anterior.

Resumen

El incremento en la proporción de aviones prioritarios (A) redujo el tiempo total de simulación, pero penalizó severamente los tiempos de espera para las categorías B y C. Este escenario es adecuado si la prioridad está claramente definida hacia una categoría específica.

Prueba 3: Mayor Proporción de Aviones de Categoría C

Parámetros Iniciales

- **Aviones de categoría A**: 5
- **Aviones de categoría B**: 5
- **Aviones de categoría C**: 20
- **Total de aviones**: 30

Observaciones Detalladas

- **Tiempo total de simulación**: 60 segundos.
- **Tiempos promedio por categoría**:
  - Categoría A: 3.2 segundos por avión.
  - Categoría B: 4.8 segundos por avión.
  - Categoría C: 10.5 segundos por avión.
- **Utilización de recursos**:
  - Las pistas estuvieron ocupadas el **95% del tiempo**.
  - Las puertas estuvieron ocupadas el **85% del tiempo**.
- **Comportamiento observado**:
  - Los aviones de categoría C, al ser mayoría y de menor prioridad, experimentaron largos tiempos de espera.
  - Las categorías A y B aterrizaron rápidamente debido a su mayor prioridad.
  - La cola alcanzó un máximo de **30 aviones**, sin exceder la capacidad.

Resumen

En este escenario, la alta proporción de aviones de baja prioridad (C) resultó en mayores tiempos de espera para esta categoría, mientras que las categorías A y B no fueron afectadas significativamente. Es un caso que puede requerir ajustes si se busca reducir el tiempo total de operación.

Comparación de Resultados

Prueba	Distribución A-B-C	Tiempo Total (seg)	Tiempo Promedio (A/B/C, seg)	Máxima Cola	% Uso de Pistas	% Uso de Puertas
Prueba 1	10 - 10 - 10	45	3.5 / 4.5 / 5.2	30	90%	90%
Prueba 2	20 - 5 - 5	40	2.8 / 6.2 / 7.1	30	100%	95%
Prueba 3	5 - 5 - 20	60	3.2 / 4.8 / 10.5	30	95%	85%

Observaciones Generales

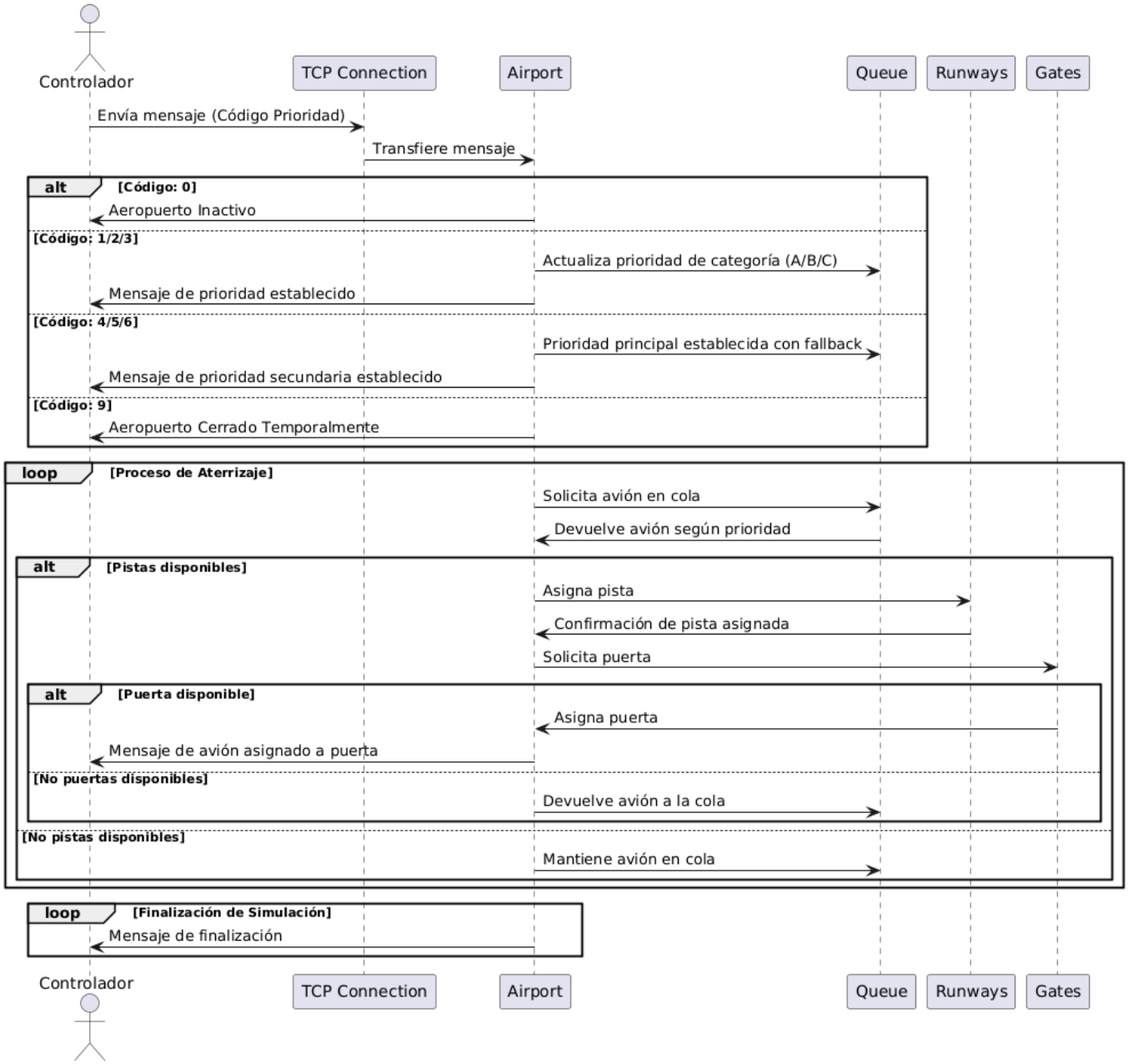
1. **Prueba 1**:
   - Operación balanceada.
   - Recursos bien utilizados con tiempos de espera razonables.
   - Ideal para escenarios equilibrados.

2. **Prueba 2**:
    - Alta prioridad para categoría A resultó en tiempos totales reducidos.
    - Penalizó significativamente las categorías B y C.
3. **Prueba 3**:
    - Dominio de aviones de categoría C prolongó los tiempos totales.
    - Requiere ajustes en recursos o prioridades para evitar cuellos de botella.

Conclusiones y Recomendaciones

- **Conclusión General**:
    - El sistema manejó eficientemente hasta 30 aviones con las configuraciones actuales.
    - Las prioridades tienen un impacto directo en los tiempos de espera y la eficiencia del sistema.
- **Recomendaciones**:
    1. Para escenarios con alta proporción de aviones de categorías B y C, aumentar recursos (pistas y puertas) puede mejorar los tiempos.
    2. Ajustar dinámicamente las prioridades podría equilibrar los tiempos de espera entre categorías.
    3. Implementar un límite adaptativo para la cola en función del tipo de avión y su prioridad.
- **Sugerencias de optimización futura**:
    - Introducir métricas de penalización para evaluar los tiempos de espera desproporcionados.
    - Simular escenarios con más de 30 aviones y evaluar la escalabilidad del sistema.

Diagrama de Secuencia



Este diagrama de secuencia ilustra el flujo de interacción entre los diferentes componentes del sistema para gestionar la llegada de aviones, el proceso de aterrizaje y la asignación de puertas en un aeropuerto. A continuación, se describen los pasos principales y las interacciones relevantes.

Actores y Componentes

1. **Controlador**: Representa al usuario o sistema externo que envía comandos al sistema (mensajes con códigos de prioridad).
2. **TCP Connection**: Gestiona la conexión entre el controlador y el sistema del aeropuerto, transfiriendo los mensajes recibidos.
3. **Airport**: Representa el núcleo del sistema de gestión del aeropuerto, responsable de coordinar las pistas, puertas y la cola de aviones.
4. **Queue**: Estructura de datos que organiza los aviones en función de sus prioridades.
5. **Runways**: Conjunto de pistas disponibles para aterrizajes.
6. **Gates**: Conjunto de puertas disponibles para asignar a los aviones tras el aterrizaje.

Flujo de Trabajo

1. Recepción de Mensajes del Controlador

- El **Controlador** envía un mensaje al sistema, que contiene un código relacionado con la prioridad o el estado del aeropuerto.
- La **TCP Connection** transfiere el mensaje al componente **Airport**, donde se procesa.

2. Procesamiento del Código

- Dependiendo del código recibido, se realizan diferentes acciones:
  - **Código 0**: El aeropuerto se declara inactivo y se envía una notificación al controlador.
  - **Códigos 1/2/3**: Se actualiza la prioridad de las categorías de aviones (A/B/C) en la cola.
  - **Códigos 4/5/6**: Se establece una prioridad principal para la gestión de la cola, con una secundaria como fallback.
  - **Código 9**: El aeropuerto se cierra temporalmente, notificando al controlador.

3. Gestión del Proceso de Aterrizaje

El sistema entra en un bucle continuo para procesar los aterrizajes:

1. **Solicitud de Avión**:
   - El componente **Airport** solicita el siguiente avión de la **Queue**, que responde con el avión con mayor prioridad.
2. **Asignación de Pista**:
   - Si hay pistas disponibles, se asigna una al avión mediante el componente **Runways**.
   - El componente **Runways** confirma que la pista ha sido asignada.
3. **Asignación de Puerta**:
   - El componente **Airport** solicita una puerta al componente **Gates**.
   - **Puerta disponible**: Se asigna una puerta al avión y se notifica al controlador.
   - **No hay puertas disponibles**: El avión se devuelve a la cola de espera.
4. **Sin Pistas Disponibles**:
   - Si no hay pistas disponibles, el avión permanece en la cola de espera hasta que haya una pista libre.

4. Finalización de la Simulación

- Una vez que todos los aviones han sido procesados, el sistema envía un mensaje al controlador indicando la finalización de la simulación.

Elementos Clave

Alternativas (`alt`)

Se utilizan bloques de decisión para manejar diferentes condiciones, como:

- Estado del aeropuerto (activo/inactivo).
- Disponibilidad de pistas y puertas.
- Diferentes tipos de prioridades.

Bucles (`loop`)

El diagrama incluye bucles para:

- Procesar aviones en la cola de espera.
- Gestionar aterrizajes y asignaciones de recursos hasta que todos los aviones sean procesados.

Beneficios del Sistema

- **Eficiencia**: Optimización de recursos (pistas y puertas) mediante el uso de prioridades.
- **Flexibilidad**: Gestión dinámica basada en diferentes códigos de prioridad y estado del aeropuerto.
- **Notificaciones**: Comunicación clara con el controlador sobre el estado del sistema.

Este diagrama proporciona una visión detallada del comportamiento del sistema, facilitando el análisis y la implementación del código relacionado.

Repositorio Github

Repositorio

En este repositorio se encuentra el pdf que es la memoria de la práctica y también se encuentra el codigo principal **cliente.go**, para ejecutar este programa necesitaremos primero ejecutar el archivo **server.go**, después el **cliente.go** y por último el codigo de **enaire.go**

Conclusión proyecto

El simulador de gestión de aeropuertos desarrollado representa una solución eficiente, flexible y escalable para manejar el flujo de aviones, asignar recursos aeroportuarios y optimizar operaciones mediante prioridades dinámicas y concurrentes. La implementación incluye características avanzadas de concurrencia, gestión de prioridades y comunicación en tiempo real a través de una conexión TCP, lo que refleja un enfoque realista y robusto para la simulación de entornos aeroportuarios.