Importaciones

1. **rclpy**: Biblioteca de ROS 2 para Python que permite la comunicación entre nodos.
2. **Node**: Clase base para todos los nodos en ROS 2.
3. **Fibonacci**: Interfaz de acción que define cómo calcular una secuencia de Fibonacci.
4. **yasmin** y **yasmin_ros**: Bibliotecas para manejar máquinas de estado y acciones en ROS 2.
5. **yasmin_viewer**: Herramienta para visualizar el estado de la máquina de estado.

Clase FibonacciState

Esta clase maneja el estado de la acción de Fibonacci.

- **init**: Constructor que configura el estado de acción.
    - `node`: El nodo que ejecuta esta acción.
    - `Fibonacci`: Tipo de acción que se ejecuta (calcular Fibonacci).
    - `"/fibonacci"`: Nombre del servidor de acción.
    - `self.create_goal_handler`: Función para definir la meta (el objetivo) de la acción.
    - `None`: Define los posibles resultados de la acción.
    - `self.response_handler`: Función que maneja la respuesta cuando la acción se completa.
- **create_goal_handler**: Método que establece el objetivo de la acción.
    - `blackboard.n`: El número de términos de la secuencia de Fibonacci a calcular.
- **response_handler**: Método que procesa la respuesta de la acción.
    - Guarda el resultado de la secuencia de Fibonacci en el `Blackboard`.
    - Retorna `SUCCEED` para indicar que la acción fue exitosa.

Funciones Auxiliares

- **set_int**: Configura un valor entero en el `Blackboard`.
    - Establece `blackboard.n` en 3.
    - Retorna `SUCCEED`.
- **print_result**: Imprime el resultado almacenado en el `Blackboard`.
    - Imprime `blackboard.fibo_res`.
    - Retorna `SUCCEED`.

Clase ActionClientDemoNode

Esta clase representa el nodo principal que maneja la máquina de estados.

- **init**: Constructor que configura el nodo y la máquina de estados.
    - Crea una máquina de estados `StateMachine` con un resultado final llamado "outcome4".
    - Añade tres estados a la máquina de estados:
        1. `SETTING_INT`: Estado para establecer un valor entero en el `Blackboard` usando `set_int`.

2. `CALLING_FIBONACCI`: Estado que llama a la acción de Fibonacci usando `FibonacciState`.
3. `PRINTING_RESULT`: Estado que imprime el resultado de la acción usando `print_result`.

- `YasminViewerPub`: Publica la máquina de estados para permitir su visualización.
- Ejecuta la máquina de estados y guarda el resultado.

Función main

Función principal que se ejecuta al iniciar el script.

- **main**: Inicializa ROS 2, crea un nodo `ActionClientDemoNode` y ejecuta su método `join_spin()`, luego cierra ROS 2.

Ejecución del script

- **if name == "main"**: Verifica si el script se está ejecutando directamente y, si es así, llama a `main()`.

Desglose del Flujo del Programa

1. **Inicialización**: El programa comienza importando las bibliotecas necesarias.
2. **Definición de Estados**: Define cómo manejar los estados de la acción de Fibonacci.
3. **Máquina de Estados**: Configura una máquina de estados que incluye:
   - Configuración de un número (`SETTING_INT`).
   - Llamada a la acción de Fibonacci (`CALLING_FIBONACCI`).
   - Impresión del resultado (`PRINTING_RESULT`).
4. **Publicación y Ejecución**: Publica la máquina de estados para su visualización y la ejecuta.
5. **Función Principal**: Gestiona la inicialización y finalización del nodo ROS 2.

Este programa demuestra cómo usar ROS 2 y la biblioteca `yasmin` para gestionar acciones complejas a través de una máquina de estados, mostrando cómo definir, ejecutar y visualizar los estados y transiciones.