Técnicas de Programación Avanzada

TECNICATURA UNIVERSITARIA EN DESARROLLO DE SOFTWARE

academia CIMNE IBER

UPATECO UNIVERSIDAD PROVINCIAL

Ministerio de Economía y Servicios Públicos
Gobierno de Salta

Tarea 6: Sincronización en Java

Matchmaking en Videojuego

En los videojuegos, generalmente se plantea un sistema para emparejar partidas de manera justa. Estos sistemas varían en su implementación y reglas para cumplir con las expectativas de los jugadores. En esta tarea, implementaremos un sistema de emparejamiento básico que creará partidas con una cantidad determinada de jugadores.

Nuestro sistema cuenta con las siguientes características más relevantes:

- Cada jugador tendrá un id numérico secuencial para evitar riesgos de generar el mismo
- Cada jugador puede tener un ranking entre 1000 y 24000 puntos
- Una partida se juega entre solo 2 jugadores
- Las partidas pueden terminar en victoria/derrota o un empate
- El sistema nos dirá cuántas partidas hay en curso
- El jugador al llegar espera a que haya otro jugador en cola con un rango similar

Los detalles sobre cada parte del sistema se especificarán en la definición de cada una de las clases necesarias para el funcionamiento correcto.

Clase Jugador

Esta clase representa a una instancia de un jugador que participará en nuestro videojuego. Cada jugador cuenta con la siguiente información:

- `Id`: Un valor numérico único para el jugador.
- `Ranking`: El ranking del jugador con un valor entre 1000 y 24000.
- `TiempoEspera`: El tiempo que lleva esperando este jugador por una partida.

Nota ayuda: Para TiempoEspera, usaremos la función de Java `System.currentTimeMillis()` para obtener la hora del instante en que el jugador queda en cola y se usará más adelante para otras operaciones.

Aquí solo necesitamos getters de estos atributos y no más métodos adicionales.

Clase Partida

Esta clase representa una partida entre dos jugadores, aquí tendremos el resultado si fue victoria de uno de los jugadores, o un empate. Al tratarse de una tarea a realizar un hilo, esta clase implementará Runnable con este propósito.

Detalles de implementación

- `Jugador1`: Instancia de la clase Jugador.
- `Jugador2`: Instancia de la clase jugador.
- `SistemaJuego`: Instancia del sistema que detallaremos más adelante.

El constructor traerá las instancias de estos objetos.

El método run() de la clase Partida tendrá las siguientes tareas en el orden que se mencionan.

1. Aumentar la cantidad de partidas activas en el sistema (método a especificar en esta clase)
2. Indicar que empezó una partida entre los jugadores correspondientes.
3. Simular la partida con una espera de 3 segundos.
4. Generar un puntaje aleatorio para cada uno con un valor aleatorio entre 0 y 5.
5. Calculamos quién fue el ganador según los resultados.
6. Mostramos el resultado final de la partida con el siguiente formato:
    a. Resultado de la partida: Jugador [id] + (puntos) + vs Jugador [id] + (puntos) - Resultado del cálculo en (5)
7. Disminuir la cantidad de partidas activas en el sistema.

Clase GeneradorJugador

Esta clase tiene como tarea principal generar los jugadores que se usarán en el videojuego. Como tarea que es, implementará la interface Runnable. Cuenta con la siguiente arquitectura.

- SistemaJuego: Instancia del sistema que detallaremos más adelante.
- JugadoresTotales: Entero que nos dirá cuántos jugadores debemos generar.

Su constructor recibirá ambos objetos desde afuera.

El método run() de la clase GeneradorJugador tendrá las siguientes tareas en el orden que se mencionan.

1. Contenemos el código siguiente en un ciclo for con la cantidad de jugadores determinada.
2. Primero esperamos un tiempo al azar de hasta 5 segundos para generar un jugador.
3. Generamos una instancia de jugador con id=i, y un ranking aleatorio entre 1000 y 24000 puntos.
4. Agregamos al jugador al sistema (mediante métodos a detallar en esta clase).
5. Mostramos un mensaje de información con el siguiente estilo o similar:
    a. El Jugador [id] (Ranking: [ranking]) se unió a la cola de espera.
6. Terminado el ciclo, estableceremos en falso la generación de jugadores en el sistema (mediante métodos a detallar en esta clase).
7. Mostramos un mensaje que se terminaron de generar los jugadores indicados.

Clase SistemaJuego

Todo nuestro videojuego tiene un sistema general que se encargará de que todo funcione según lo previsto, esta clase tiene la funcionalidad principal de nuestro programa y se detallará los atributos y métodos necesarios para poder trabajar con ella.

El sistema necesita los siguientes atributos:

- jugadoresEspera: Una lista de jugadores que esperan a que se cree una partida.
- partidasActivas: Valor entero que indica las partidas en curso en un momento determinado.
- RANGO_INICIAL: Entero constante con valor 500. Este valor indica la máxima diferencia entre rankings para que se emparejen dos jugadores, ejemplos:
    - Jugador1 23200, Jugador2 23300. Válido
    - Jugador1 10000, Jugador2 10400. Válido
    - Jugador1 15000, Jugador2 16000. Inválido
- AUMENTO_POR_SEGUNDO: Entero constante con valor 100. Este valor indica el incremento del rango de diferencia permitido. En otras palabras, si pasaron 10 segundos desde que un jugador esperaba su partida, su rango pasó de 500 a 1500, y puede ser emparejado con alguien que tenga esta diferencia de ranking.
    - Este atributo trabaja en conjunto con el tiempo de espera de cada jugador, se darán detalles al llegar a los métodos.
- Monitor: Objeto a utilizar como monitor para manejar el acceso a recursos compartidos.
- GenerandoJugadores: Valor booleano inicialmente en true.
- SistemaEnEjecución: Valor booleano inicialmente en true.

Esta clase no necesita un constructor dado que no necesita instancias externas para crear su propia instancia.

Métodos necesarios

La clase SistemaJuego cuenta con una variedad de métodos que nos permiten trabajar el sistema.

agregarJugador

Método que recibe un jugador y lo agrega a los jugadores en espera. Es un método que trabaja con un recurso compartido.

revisarEstadoSistema

Metodo que actualiza el booleano SistemaEnEjecución a falso cuando se dan ciertas condiciones en conjunto:

1. Ya no se están generando jugadores.
2. La lista de jugadores en espera está vacía.
3. Las partidas activas son iguales a 0.

Setter de generandoJugadores

Método que es llamado por la clase generadora de jugadores cuando finaliza y cambia su valor a false.

AumentarPartidasActivas

Método que aumenta el valor de las partidas activas. Es un método que trabaja con un recurso compartido.

DisminuirPartidasActivas

Método que disminuye el valor de las partidas activas. Es un método que trabaja con un recurso compartido.

Nota: Ambos métodos deben mostrar las partidas activas luego de actualizar su valor.

encontrarPartida

Este método recibe un jugador como parámetro y trabaja como sigue:

1. Calcula el tiempo esperado con la siguiente fórmula:
    a. (HoraActual - TiempoEspera del jugador recibido) / 1000
2. Calcula el rango de oponente permitido de la siguiente forma:
    a. RANGO_INICIAL + (int) tiempoEsperado * AUMENTO_POR_SEGUNDO
    b. rangoPermitido = resultado del cálculo en a)
3. Ahora, para cada jugador en los jugadores en espera hacemos lo siguiente:
    a. Si jugador no es el mismo que el recibido, y la diferencia de rankings está dentro del rango permitido, entonces devolver a este oponente.
4. Si luego del ciclo no encontró oponente, solo devolvemos null.

Método run del SistemaJuego

Este es el método más importante de todo nuestro sistema, a continuación se da en detalle su funcionamiento y comportamiento esperado.

1. El método encapsula todo en un while que se repite mientras el sistema esté en ejecución, terminado esto, se da un mensaje que el sistema se detuvo.
2. Comenzamos con dos jugadores inicializados como null para utilizar en todo este ciclo.
3. Pasamos a un bloque sincronizado con las siguientes tareas:
    a. Comprobar si hay 2 o más jugadores en espera
    b. Si es así, el jugador1 será el primero de la lista, y el jugador2 será lo que responda nuestro método encontrarPartida(jugador)
    c. Si nuestro método encontrarPartida() no nos respondió null, entonces eliminamos a ambos jugadores de la lista de jugadores en espera
4. Luego de esto, comprobamos que jugador2 no sea null:
    a. Si no es nulo, crearemos una partida con ambos jugadores e iniciaremos la ejecución de este hilo.
    b. Si es nulo, entonces esperamos 100 milisegundos hasta volver a intentar emparejar jugadores.
5. Llamamos a revisarEstadoSsistema para actualizar el estado si se cumplen las condiciones.
6. Luego del while mostraremos un mensaje indicando que finalizó el sistema del juego.

Ejecución de todo el sistema del videojuego

Ya con todo esto implementado, tenemos todas las herramientas necesarias para iniciar nuestro juego, los pasos a seguir son los siguientes:

1. **Instanciar nuestro sistema de juego**
2. **Crear un hilo para emparejamiento con nuestro sistema, e iniciarlo**
3. **Establecer la cantidad de jugadores, por ejemplo 20**
4. **Crear un hilo para generar jugadores e iniciarlo**
5. **Pausamos el programa con join al hilo de emparejamiento para esperar a que este finalice**
6. **Indicamos el fin de todo el programa.**