

# TPG

## El bullicio del Callejón Aurum

### El contexto:

Están llegando al final del desarrollo de la **1.0** de *Honkai Star Rail* y, como es costumbre en todas las versiones, su última tarea es la implementación del evento temporal principal de la versión: **el bullicio del Callejón Aurum**. Para esto, dado el alcance del evento, *Hoyoverse* les asignó un grupo de trabajo para aliviar la carga laboral.

La zona comercial del Callejón Aurum, que ha estado en decadencia durante mucho tiempo, ahora tiene una oportunidad de revitalizarse. Puede ser financiada por la Corporación Interastral de la Paz (**IPC**) o revitalizada por el gremio local y tú (o, mejor dicho, los jugadores) tienen el poder de influir en esta decisión.

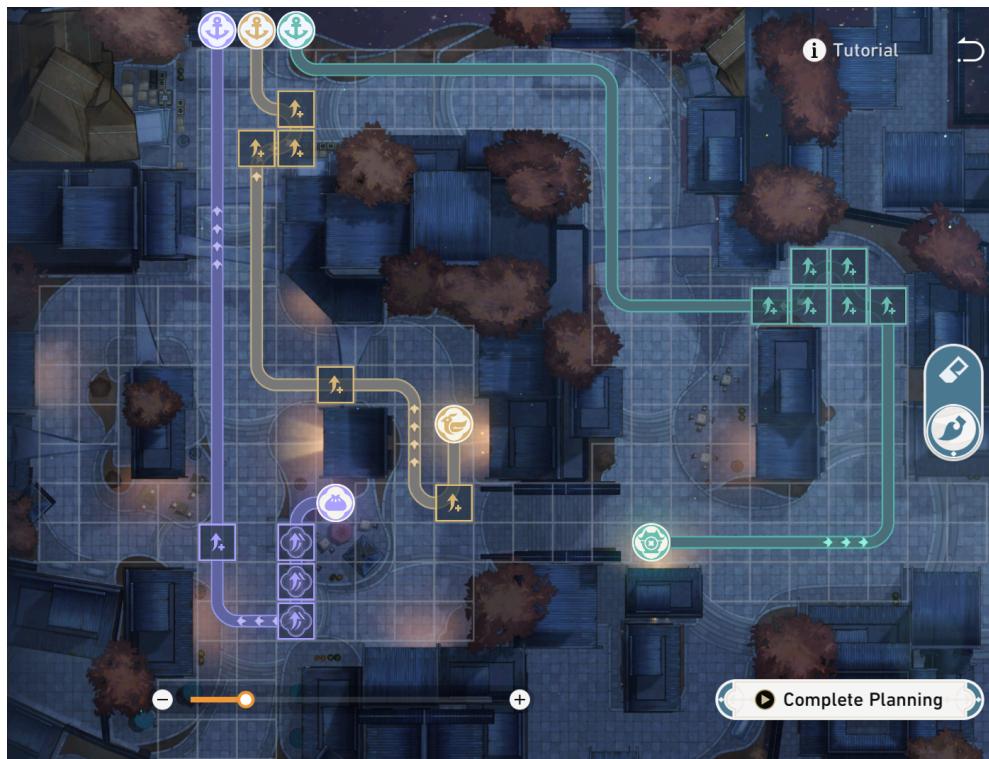


Imagen 1: Visualización del minijuego de planeamiento de logística.

## La temática del evento:

En este evento estaremos situados en el **Callejón Aurum** del **Xianzhou Luofu**, uno de los planetas del juego, con ambientación asiática. Este callejón es un centro turístico, principalmente con locales de comida y antigüedades, pero que en los últimos años entró en declive. Ante la posibilidad de bancarrota, muchos locales accedieron a un trato económico con la **IPC**, que utilizaría al callejón como uno más de sus depósitos, asegurando la estabilidad económica pero perdiendo en el proceso toda autonomía.

En este contexto, la misión del jugador es ayudar a los locales del callejón a distribuir diferentes paquetes, renovar la infraestructura, hacer publicidad y finalmente aumentar la cantidad de clientes para poder declinar la oferta de la **IPC**.



Imagen 2: Poster publicitario para un local de comida.

# El minijuego principal:

El *gameplay loop* principal es el **planeamiento de logística**. El jugador deberá optimizar el camino que tomará para entregar un paquete asociado a un **pedido**, minimizando la cantidad de casilleros recorridos.

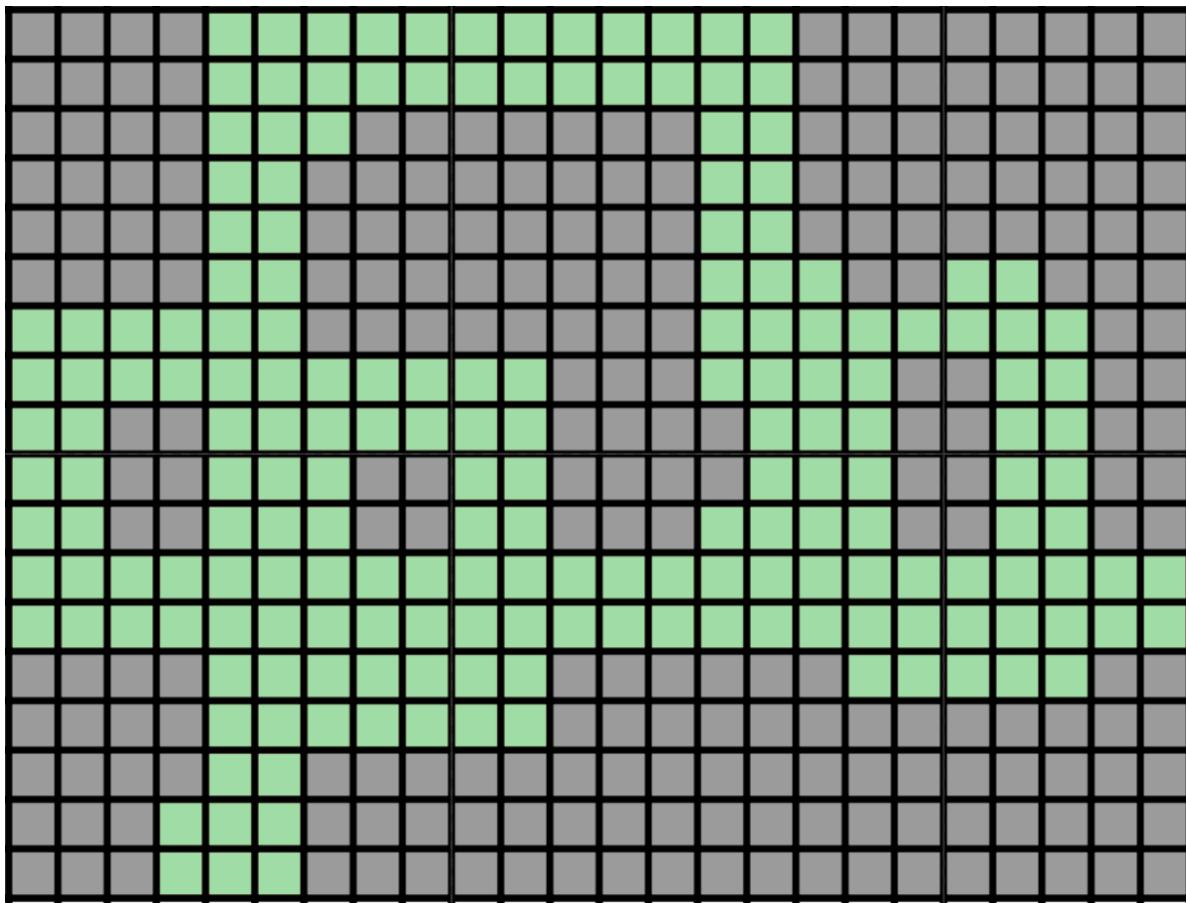


Imagen 3: Mapa del callejón, con los casilleros caminables en verde.

El jugador podrá moverse a los casilleros aledaños horizontales y verticales (arriba, abajo, izquierda, derecha) pero **no podrá moverse en diagonal**.

Sin embargo, como ustedes son desarrolladores y no jugadores, deberán implementar algún algoritmo que les permita *buscar automáticamente el camino óptimo*, con el objetivo de no tener que pensar soluciones y así probar más rápidamente el minijuego. Como la cantidad de casilleros en el mapa es alta, sus superiores en Hoyoverse necesitan que optimicen el proceso de búsqueda (usando el algoritmo **A\*** con una *heurística* adecuada).

## Sobre los locales:

Los **locales** serán los que crean los **pedidos**. Están identificados por su *nombre*, tendrán una *prioridad* (un número positivo, a mayor valor, mayor prioridad) y una *ubicación* en el mapa, que debe ser un casillero caminable (verde). Naturalmente, como un local ocupa espacio físico, el casillero en el que esté ubicado pasará a ser **no caminable** (gris).

Se desea poder buscar los locales por nombre de forma eficaz, considerando que son *ordenables alfabéticamente*. Para esto, se usará un **diccionario** implementado con un **ABB**.

## Sobre los pedidos:

Los **pedidos** serán creados por un local y tendrán como destino otro local. Es decir, que un **pedido** tendrá como *casillero inicial* la ubicación del local que encargó el pedido y como *casillero final* la ubicación del local destino. Además, tendrán el *peso del paquete*.

La resolución de **pedidos** será por prioridad, la cual es calculada de la siguiente forma:

$$\text{prioridad\_pedido} = \text{prioridad\_local} * \text{peso\_paquete}$$

Se desea que la búsqueda del **pedido** con *mayor prioridad* sea óptima en todo momento, por lo que se usará un **heap**.

## Sobre los obstáculos:

Como el callejón empezará a tener más actividad y considerando que las calles son angostas, podrán aparecer en el mapa, por cada entrega, **hasta cuatro clientes** (es decir, de 0 a 4 clientes), ocupando casilleros aleatorios caminables en el mapa y obstaculizando la entrega de paquetes. Como el cliente siempre tiene la razón, es nuestro deber cambiar el camino de entrega para esquivarlos y evitar accidentes.

Si la ubicación de los clientes *imposibilita que un pedido sea completado*, se debe mostrar e informar acordemente y luego volver a intentarlo, hasta que se pueda completar.

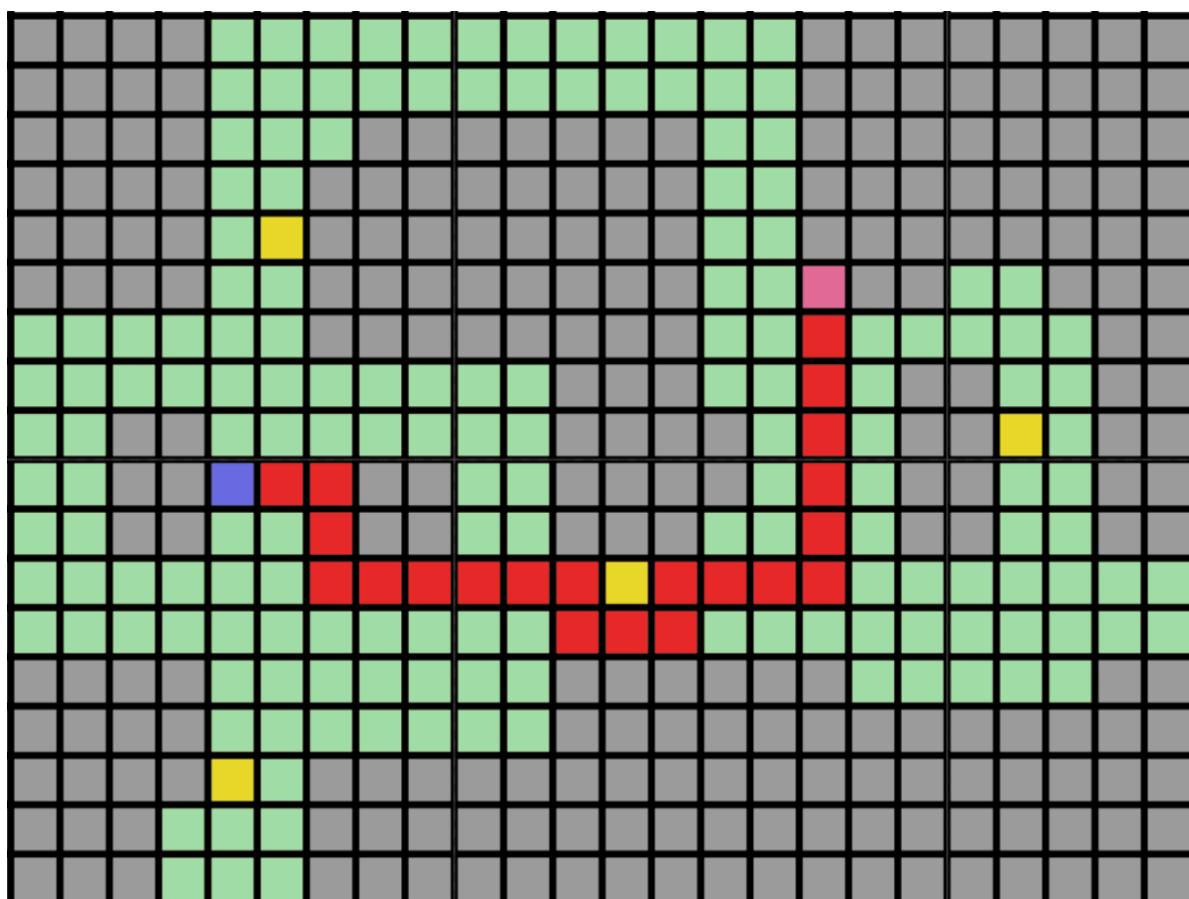


Imagen 4: Ejemplo de resolución de un pedido. Azul: origen, rosa: destino, rojo: camino, amarillo: clientes.

## El minijuego secundario:

Como parte de las tareas de renovación del callejón, el jugador deberá optimizar el presupuesto de reparaciones, priorizando las tareas en los *senderos más transitados* y conectando *todos los locales*. Para esto, necesitarán implementar un **algoritmo de optimización de red** (*Minimum Spanning Tree*).

Los senderos más transitados están dados por la cantidad de **pedidos** entre **dos locales**. Por ejemplo, optimizar las tareas de renovación en la siguiente situación:

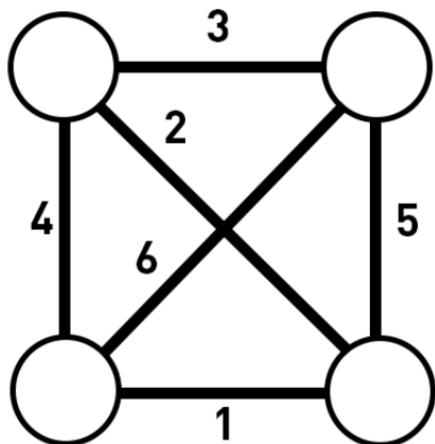


Imagen 5: Representación de los locales y los pedidos entre locales.

Deberá dar como resultado:

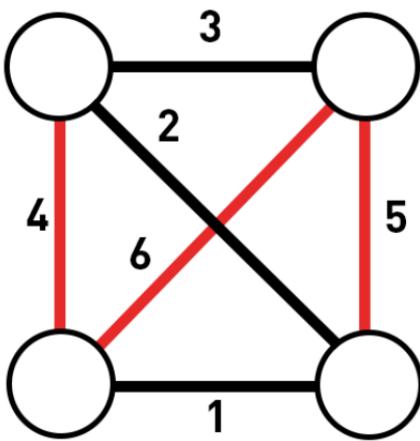


Imagen 6: Resultado esperado de los senderos a restaurar.

**NOTA:** No es necesario que tenga correlación con el mapa planteado anteriormente. Solo se debe encontrar e informar qué senderos deben ser priorizados.

# Desarrollo del TP:

Se deberán desarrollar las estructuras de datos necesarias para el proyecto. Las plantillas están en el repositorio base del Trabajo Práctico.

Se deberán desarrollar los sistemas necesarios para el funcionamiento de los dos minijuegos, incluyendo una interfaz de usuario completa y una aplicación totalmente funcional e interactuable, utilizando el paradigma de objetos. El mapa del minijuego de **planeamiento de logística** debe mostrarse como se ve en las imágenes: debe mostrarse claramente qué camino se debe recorrer, la posición de los clientes, locales, casilleros caminables, etc.

La interfaz de usuario queda a criterio del grupo. Hay libertad artística total: pueden usar librerías, emojis, colores, entre otros recursos.

Adicionalmente, se deberán desarrollar dos documentos entregables:

1. **Informe sobre complejidad algorítmica:**

Nuestros superiores de *Hoyoverse* están muy preocupados por la eficiencia temporal de nuestras estructuras y algoritmos, por lo que nos piden un informe que muestre la cantidad de operaciones realizadas en los métodos principales de nuestras estructuras.

Concretamente, se deberá analizar la complejidad algorítmica del **heap**, mostrando la cantidad de *upheaps* al realizar una alta y de *downheaps* al realizar una baja. Se deberá calcular:

- 1.1. Promedio de *upheaps* en 100 altas en un heap con 100/1000/10000 elementos. El elemento a agregar debe ser aleatorio entre iteraciones.
  - 1.2. Promedio de *downheaps* en 100 bajas en un heap con 100/1000/10000 elementos.
  - 1.3. Entre iteraciones, se debe generar un heap de forma aleatoria con números del 0 al 100000.
- Finalmente, se deberá graficar los resultados obtenidos y comparar con el orden teórico.

[Plantilla para el informe](#)

2. **Video de demostración del programa:**

El video deberá mostrar el correcto funcionamiento de los dos minijuegos y del programa en general.

No debe superar los **cinco minutos**.

[Enlace al repositorio base del Trabajo Práctico](#)

# Aclaraciones y criterios de corrección:

Para que el trabajo tenga buena recepción por nuestros superiores en Hoyoverse, se deberán cumplir las siguientes pautas:

1. Debe implementar las clases de manera totalmente original y propia. La detección de copias resultará en la **pérdida de la regularidad de los involucrados**.
2. El trabajo debe compilar con las flags **-Wall -Werror -Wconversion**. Un trabajo que no compila es un trabajo que *no funciona* (y por lo tanto no pasa ninguna prueba).
3. El código **debe pasar** los tests de la cátedra y también respetar sus firmas (es decir, no se puede *modificar* las pruebas).
4. El código **debe estar escrito** en *snake\_case*.

Adicionalmente:

1. **No se puede utilizar** la librería STL de C++ para las estructuras a implementar en este trabajo (*std::map*, por ejemplo). Se puede utilizar para estructuras de los TPs individuales (*std::vector*, *std::list*).
2. Se pueden utilizar otras librerías.

Los criterios de evaluación y corrección por parte de la cátedra son:

1. **Funcionalidad** (40% de la nota grupal):

Compilación: sin warnings ni errores.

Funcionalidad: que el código pase las pruebas.

Memoria dinámica: el código no debe perder memoria. Tampoco debe haber errores de acceso a memoria.

Manejo de archivos.

2. **Estilo de código** (30% de la nota grupal):

Uso del paradigma.

Eficiencia espacial.

Eficiencia temporal.

Modularización.

Precondiciones y postcondiciones.

Buenas prácticas de programación propuestas por la cátedra.

3. **Entregables** (30% de la nota grupal):

Video de demostración.

Informe de complejidad algorítmica.

4. **Participación del integrante** (porcentaje sobre el total de la nota grupal, 0% a 100%):

Aportes de código al repositorio.

Defensa oral.

# Formato de entrega:

Se deberá subir el código a la branch *main* del repositorio de GitHub del grupo. Adicionalmente, se deberá completar el *README.md* con la información de los integrantes del grupo, el enlace al video de demostración y el enlace al informe de complejidad algorítmica. Se corregirá el **último commit** del *main* del repositorio remoto.

Como formalidad, se deberá subir al campus un único archivo comprimido **.zip** en la sección TPs. En la carpeta comprimida, se deberá entregar el repositorio de GitHub. Esto se puede hacer descargándolo de forma manual desde la página o generando una Release.

El nombre del archivo **debe tener** el siguiente formato:

**tpg-1c2024-NOMBRE\_GRUPO.zip**

El código entregado en el **.zip** no será revisado (salvo casos borde).

Se deberá coordinar una defensa oral con su corrector antes de la fecha límite. En la defensa oral, se deberá mostrar el trabajo realizado, el funcionamiento del programa, el informe de complejidad algorítmica y, asimismo, demostrar conocimiento del código del proyecto en su totalidad y de los temas teóricos evaluados.

El plazo de defensa vence el día **JUEVES 27 DE JUNIO**.

Puntaje: **100** puntos.

