Registro de partida:
El código a continuación, es el que usamos para registrar la partida:

```cpp
#include <iostream>
using namespace std;
#include "src/proyecto.h"
#include "src/unidades.h"
#include "src/Mapa.h"
#include "src/Terreno.h"
#include "src/Edificios.h"
#include "src/Clases_baseMapa.h"

int main() {
    // 1. INICIALIZACIÓN
    Context ctx;
    Mapa gameMap(filas: 5, columnas: 5);

    // Uso de Sobrecarga de Operadores (Recursos)
    cout << "-> Contexto Inicializado (Turno: " << ctx.turn << ")\n";
    cout << "   - Recursos Iniciales: " << ctx.playerResources << endl;

    // Uso de Sobrecarga de Operadores (Mapa)
    cout << "\n-> Estado del Mapa Inicial (Sobrecarga << Mapa):\n";
    cout << gameMap << endl;

    // 2. DEMOSTRACIÓN DE POO
    // Inicializar Unidades para la demostración
    Unidad* soldado = new Soldado(c: Coord(x: 0, y: 0), o: Owner::PLAYER);
    Unidad* arquero_enemigo = new Arquero(c: Coord(x: 1, y: 1), o: Owner::SYSTEM);

    // Llamada Polimórfica (Unidad base)
    int dano_normal = soldado->atacar(arquero_enemigo);

    // Llamada Polimórfica (Subclase específica)
    soldado->habilidad_especial(arquero_enemigo); // Esto imprimirá el LOG de Soldado
```

```cpp
// 3. CICLO DE TURNO (Flujo Polimórfico)
ctx.controllers.push_back(new JugadorControl());
ctx.controllers.push_back(new SystemController());

// El bucle principal que activa el polimorfismo del flujo de control
for (controles* c : ctx.controllers) {
    c->update([&]ctx); // Esto imprimirá los LOGS de los Controllers
}
ctx.turn++;

// 4. SISTEMAS DEL MUNDO (Eventos)
Evento* evento_recursos = new RecursosEvento(ctx.turn);
evento_recursos->apply([&]ctx); // Esto imprimirá el LOG del Evento

// ... (Limpieza de memoria) ...
return 0;
```

Y el registro que obtuvimos del log fue el siguiente:

1. El Turno 1 se inicializa usando los recursos base. La estrategia inicial debe enfocarse en la eficiencia del combate para asegurar la ventaja de la posición.

```
-> Contexto Inicializado (Turno: 1)
   - Recursos Iniciales: [Food:20 Metal:10 Energy:5]

-> Estado del Mapa Inicial (Sobrecarga << Mapa):
Mapa (5x5):
[P.][P.][P.][P.][P.]
[P.][P.][P.][P.][P.]
[P.][P.][P.][P.][P.]
[P.][P.][P.][P.][P.]
[P.][P.][P.][P.][P.]
```

2. El Soldado inicia el combate priorizando al Arquero. Es un movimiento clave para eliminar la amenaza de daño a distancia del enemigo.

```
[LOG: COMBATE] Soldado ((0,0)) ATACA a Arquero ((1,1)). Vida Inicial: 70
[LOG: COMBATE]   - Daño infligido: 15. Vida restante del objetivo: 55
```

3. El uso inmediato del Polimorfismo (Ataque x1.5) permite un ataque especial. Esto asegura la baja de la unidad en el Turno 1 y optimiza la acción del Soldado, ahorrando turnos de combate.

```
[LOG: POLIMORFISMO] **Soldado** usa Habilidad Especial (Ataque x1.5).
[LOG: POLIMORFISMO]   - Daño especial: 25. Nueva vida: 30
```

El Arquero debería tener 30 de vida restante, confirmando que la unidad ha sido eliminada o está a punto de serlo por un ataque posterior.

```
[LOG: FLUJO] [JugadorControl] Inicia turno del jugador (Turno: 1)
[JugadorControl] Turno del jugador (Turno: 1)

[LOG: FLUJO] [SystemController] Ejecutando lógica del sistema (Turno: 1)
[SystemController] Ejecutando logica del sistema (Turno: 1)
```

4. Activación de evento programado. La ganancia de 5 de Comida en el siguiente turno (Turno 2) asegura que el jugador tendrá recursos para una acción de alto costo al inicio del juego.

```
[LOG: EVENTO] **EVENTO ACTIVADO**: RecursosEvento (Turno programado: 2)
[LOG: EVENTO]   - Recursos ANTES: [Food:20 Metal:10 Energy:5]
[Evento] Recursos modificados: Player Resources +5 de comida.
[LOG: EVENTO]   - Se agregan 5 de Food.
[LOG: EVENTO]   - Recursos DESPUÉS: [Food:25 Metal:10 Energy:5]

Process finished with exit code 0
```

En conclusión, el turno y el registro (log) han demostrado la correcta ejecución y secuenciación de los principales módulos del sistema en el Turno 1:

1. **Módulo de Combate:** Se verifica la aplicación secuencial de daño (daño base, seguido de daño especial por habilidad), resultando en la modificación esperada de la variable vida restante del objetivo. Su funcionalidad es gestionar la interacción directa entre unidades y actualizar sus estados.
2. **Módulo de Flujo y Control :** Se confirma el cambio de fase de control del sistema al control del jugador y la posterior ejecución de la lógica del sistema, manteniendo el orden cronológico del juego. Su funcionalidad es marcar los límites de acción para cada entidad.
3. **Módulo de Eventos :** Se valida la ejecución programada de la función RecursosEvento al finalizar el turno, lo que resultó en la modificación directa de las variables de recursos (food pasó de 20 a 25). Su funcionalidad es ejecutar cambios predefinidos o aleatorios en el estado del juego.

En resumen, el registro demuestra que el código funciona como un sistema de gestión de estado eficiente y que va registrando con precisión las transformaciones de las unidades y los recursos a lo largo del turno.