**¿Qué se aprendió en el taller?**

**1. Comportamientos Emergentes de Patrones Simples**

A través de los diferentes archivos PBM, se observó cómo configuraciones iniciales simples generan comportamientos complejos:

* **Blinker (3 células)**: Oscila entre vertical y horizontal, demostrando que un patrón puede tener estados cíclicos
* **Block (4 células)**: Permanece estático, mostrando que algunas configuraciones son estables
* **Glider (5 células)**: Se desplaza diagonalmente, evidenciando que los patrones pueden "moverse" sin propulsión
* **Toad (6 células)**: Cambia de forma manteniendo su posición, ilustrando osciladores más complejos
* **Pulsar (48 células)**: Oscila con período 3, mostrando patrones simétricos complejos

**Aprendizaje clave**: La complejidad del comportamiento no depende del número de células, sino de su configuración espacial.

**2. Algoritmo de Evaluación de Vecinos**

Se implementó la lógica de conteo de vecinos en las 8 direcciones:

[-1,-1] [0,-1] [1,-1]

[-1, 0] [X, Y] [1, 0]

[-1, 1] [0, 1] [1, 1]

**Conceptos aprendidos**:

* Manejo de condiciones de borde (células fuera del tablero se consideran muertas)
* Importancia de no contar la célula central
* Cómo un algoritmo local (solo ve vecinos inmediatos) genera patrones globales

**3. Las 4 Reglas de Conway**

Se comprendió cómo reglas simples generan comportamientos complejos:

1. **Supervivencia (2-3 vecinos)**: Simula equilibrio poblacional
2. **Muerte por soledad (<2 vecinos)**: Simula aislamiento
3. **Muerte por sobrepoblación (>3 vecinos)**: Simula escasez de recursos
4. **Nacimiento (exactamente 3 vecinos)**: Simula reproducción

**Insight**: Estas reglas crean un balance entre crecimiento y muerte que permite patrones dinámicos sostenibles.

**4. Evolución Simultánea vs Secuencial**

Se aprendió la importancia de:

* Calcular el nuevo estado basándose en el estado actual completo
* No modificar el tablero mientras se evalúa
* Usar el método clonar() para mantener dos tableros (actual y siguiente)

**5. Reutilización del FrameBuffer**

La integración con FrameBuffer enseñó:

**Conversión de representaciones**:

cpp

*// Tablero (lógico) → FrameBuffer (visual) → PPM (archivo)*

bool célula → float RGB → valores 0-255

**Ventajas de la reutilización**:

* No reinventar la generación de imágenes
* Aprovechar código probado y optimizado
* Flexibilidad para futuros colores (edad de células, densidad, etc.)

**Proceso de adaptación**:

1. El tablero usa booleanos (vivo/muerto)
2. FrameBuffer espera valores RGB flotantes
3. Se mapeó: vivo = negro (0.0), muerto = blanco (1.0)
4. FrameBuffer maneja la conversión a PPM automáticamente

**6. Patrones y Sus Propiedades**

Se identificaron categorías de patrones:

* **Estáticos**: Block - enseñan sobre configuraciones estables
* **Osciladores**: Blinker, Toad, Pulsar - muestran periodicidad
* **Naves espaciales**: Glider - demuestran movimiento sin propulsión

**Descubrimiento**: El mismo algoritmo produce comportamientos cualitativamente diferentes según el patrón inicial.

**7. Visualización de Procesos Discretos**

Se aprendió cómo:

* Convertir estados discretos en animación continua
* La relación entre FPS y percepción de movimiento
* Generar secuencias de frames numeradas correctamente
* Crear visualizadores web para ver la evolución

**8. Autómatas Celulares como Modelo Computacional**

El taller demostró:

* Cómo reglas locales generan comportamiento global
* La naturaleza determinista pero impredecible del sistema
* Aplicaciones en simulación de sistemas complejos
* Paralelismo implícito (todas las células se actualizan "a la vez")

**Conclusión del Aprendizaje**

El Juego de la Vida enseñó que:

1. **Simplicidad genera complejidad**: 4 reglas simples crean universos dinámicos
2. **El estado inicial importa**: Pequeños cambios generan evoluciones totalmente diferentes
3. **La computación es universal**: Conway demostró que es Turing-completo
4. **La visualización es clave**: Ver la evolución ayuda a entender el algoritmo
5. **La modularidad facilita**: Separar lógica (JuegoVida) de visualización (FrameBuffer) permite flexibilidad

El proyecto mostró cómo un concepto matemático abstracto (autómata celular) se convierte en una aplicación visual interactiva mediante la programación modular y el uso inteligente de herramientas existentes como FrameBuffer.