# 75.41 - Algoritmos y Programación II

*Cátedra Ing. Patricia Calvo - 2do cuatrimestre 2022*

# Trabajo Práctico 1: Juego de la Vida V1.0

## Objetivo

Realizar una aplicación que lleve adelante la ejecución del juego de la vida en un tablero limitado, de 20 filas por 80 columnas, a partir de una configuración inicial de células vivas indicadas por el usuario.

## Enunciado

El juego de la vida es en realidad un juego de cero jugadores, lo que quiere decir que su evolución está determinada por el estado inicial y no necesita ninguna entrada de datos posterior.

El "tablero de juego" es una malla formada por cuadrados ("células"), que en nuestro caso estará limitado a 20x80.

Cada célula tiene hasta 8 células vecinas, que son las que están próximas a ella, incluso en las diagonales.

Las células tienen dos estados: están "vivas" o "muertas" (o "encendidas" y "apagadas"). El estado de la malla evoluciona a lo largo de unidades de tiempo discretas (se podría decir que por turnos). El estado de todas las células se tiene en cuenta para calcular el estado de las mismas al turno siguiente. Todas las células se actualizan simultáneamente.

Las transiciones dependen del número de células vecinas vivas:

- Una célula muerta con exactamente 3 células vecinas vivas "nace" (al turno siguiente estará viva).
- Una célula viva con 2 ó 3 células vecinas vivas sigue viva, en otro caso muere o permanece muerta (por "soledad" o "superpoblación").

### Requerimientos

1. La aplicación debe solicitar al usuario que indique las posiciones en las que se colocarán células vivas al inicio del juego; todas las otras están muertas.

2. El usuario debe poder indicar tantas células vivas como desee, desde ninguna hasta 1.600.
3. Las células vivas se indicarán por fila y columna, siendo (1, 1) la celda superior izquierda y la (20, 80) la celda inferior derecha.
4. Una vez finalizado el proceso de carga de células vivas debe comenzar el juego, mostrando el estado inicial del tablero por consola utilizando caracteres e indicando la cantidad de células vivas.
5. A partir de este momento el usuario debe decidir entre:
    - ejecutar un turno,
    - reiniciar el juego o
    - terminar.
6. Con cada ejecución de un turno se debe mostrar:
    - el estado del tablero (usando caracteres para representarlo)
    - la cantidad de células vivas
    - la cantidad de células que nacieron en el último turno
    - la cantidad de células que murieron en el último turno
    - el promedio de nacimientos a lo largo del juego
    - el promedio de muertes a lo largo del juego
    - si el juego se congeló, es decir si no sufrió modificaciones en dos turnos consecutivos.

## Interfaz de usuario

Toda la interfaz de usuario debe estar basada en texto. El estado de una célula tiene que mostrarse usando caracteres dispuestos en filas y columnas.

No es necesario que se limpie la pantalla, simplemente escribir el estado de todas las células luego de cada turno. La cantidad de turnos límites la ingresa el usuario por pantalla.

## Referencias


- Wikipedia: Juego de la Vida http://es.wikipedia.org/wiki/Juego_de_la_vida
- Búsqueda en Google: Conway Juego de la Vida

# Cuestionario

Responder el siguiente Cuestionario:

1) ¿Qué es Debug?
2) ¿Qué es un “Breakpoint”?
3) ¿Qué es “Step Into”, “Step Over” y “Step Out”?

# Normas de entrega

Trabajo práctico individual: 1 persona.

Reglas generales: respetar el apéndice A.

Se deberá subir un archivo comprimido al campus, en un link que se habilitará para esta entrega. Este archivo deberá tener un nombre formado de la siguiente manera:

Padron-TP1.zip

Deberá contener los archivos fuentes (no los binarios), el informe del trabajo realizado, las respuestas al cuestionario, el manual del usuario y el manual del programador (Todo en el mismo PDF).

La fecha de entrega vence el día lunes 09/09/22 a las 23.59hs.

Se evaluará: funcionalidad, eficiencia, algoritmos utilizados, buenas prácticas de programación, modularización, documentación, gestión de memoria y estructuras de datos.

# Apéndice A

1) Usar las siguientes convenciones para nombrar identificadores.
    a) Clases, structs y Enum: Los nombres de clases y structs siempre deben comenzar con la primera letra en mayúscula en cada palabra, deben ser simples y descriptivos. Se concatenan todas las palabras. Ejemplo: Coche, Vehiculo, CentralTelefonica.
    b) Métodos y funciones: Deben comenzar con letra minúscula, y si está compuesta por 2 o más palabras, la primera letra de la segunda palabra debe comenzar con mayúscula. De preferencia que sean verbos. Ejemplo: arrancarCoche(), sumar().
    c) Variables y objetos: las variables siguen la misma convención que los métodos. Por Ejemplo: alumno, padronElectoral.
    d) Constantes: Las variables constantes o finales, las cuales no cambian su valor durante todo el programa se deben escribir en mayúsculas, concatenadas por "_". Ejemplo: ANCHO, VACIO, COLOR_BASE.
2) El lenguaje utilizado es C++, esto quiere decir que se debe utilizar siempre C++ y no C, por lo tanto una forma de darse cuenta de esto es no incluir nada que tenga .h, por ejemplo #include <iostream> .
3) No usar sentencias 'using namespace' en los .h, solo en los .cpp. Por ejemplo, para referenciar el tipo string en el .h se pone std::string.
4) No usar 'and' y 'or', utilizar los operadores '&&' y '||' respectivamente.
5) Compilar en forma ANSI. Debe estar desarrollado en linux con eclipse y g++. Utilizamos el estándar C++98.
6) Chequear memoria antes de entregar. No tener accesos fuera de rango ni memoria colgada.
7) Si el trabajo práctico requiere archivos para procesar, entregar los archivos de prueba en la entrega del TP. Utilizar siempre rutas relativas y no absolutas.
8) Entregar el informe explicando el TP realizado, manual de usuario y manual del programador.
9) Comentar el código. Todos los tipos, métodos y funciones deberían tener sus comentarios en el .h que los declara.
10) Modularizar el código. No entregar 1 o 2 archivos, separar cada clase o struct con sus funcionalidades en un .h y .cpp

11) No inicializar valores dentro del struct o .h.
12) El tablero en el TP 1 tiene que ser un array, tanto dinámico como no.
13) Si cualquier estructura de control tiene 1 línea, utilizar {} siempre, por ejemplo:

```
for(int i = 0; i < 10; i++) {
      std::cout << i;
}
```