Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

IC-1801 Taller de Programación

Prof. Mauricio Avilés

Proyecto Programado – Autómatas celulares

Introducción

Los autómatas celulares son modelos que consisten en una cuadrícula de células, cada una con un conjunto de estados posibles y células vecinas (vecindario) que determinan su próximo estado. La cuadrícula de células avanza en el tiempo a través de una serie de iteraciones, a partir de un estado inicial predefinido. En cada iteración se determina el estado de cada célula a partir de una función que considera el estado propio y el de sus vecinos. Dependiendo de las reglas utilizadas y el estado inicial las células pueden llegar a formar patrones que persisten a través del tiempo y tienen comportamientos bien definidos.

El estudio de los autómatas celulares ha llevado a aplicaciones como simulaciones de tráfico, diseño de estructuras, composición musical, criptografía, generación de números aleatorios, vida artificial, explicar comportamientos en la naturaleza como el crecimiento de cristales, evolución, comportamiento de fluidos; entre otros.

Software a desarrollar

El proyecto consiste en la implementación de varios sistemas de autómatas celulares por medio del lenguaje de programación Python y alguna biblioteca gráfica como Pygame. Los sistemas a implementar son:

- El cerebro de Brian
- Autómata celular cíclico
- Hormiga de Langton
- Modelo de tráfico Biham-Middleton-Levine
- Autómata adicional

Debe haber una variable global que permita indicar la cantidad de filas y columnas de la matriz. El tamaño de cada célula debe calcularse a partir del tamaño de la pantalla y la cantidad de filas y columnas.

Para todos los autómatas celulares se desean desarrollar las siguientes características:

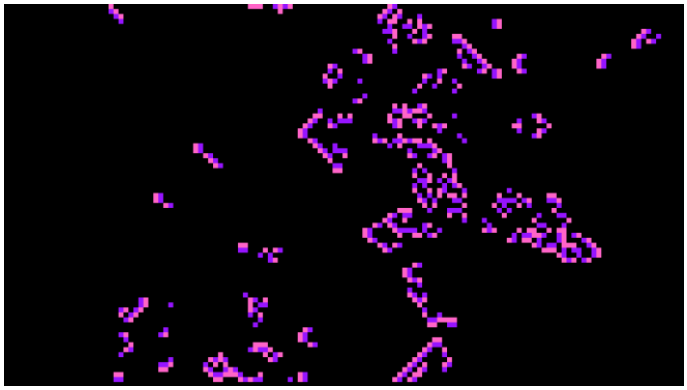
1. Solicitar parámetros iniciales al usuario
    a. Cantidad de filas y columnas
    b. Tamaño de las celdas
    c. Parámetros específicos de cada autómata
2. Pausar y continuar con la tecla espacio.
3. Cambiar el estado de las células con el clic del mouse (con cada clic cambiar la célula al siguiente estado)
4. Guardar el estado completo del autómata en un archivo mediante la tecla G (se recomienda usar pickle)
5. Cargar el estado completo del autómata desde un archivo mediante la tecla C (pickle)
6. Reiniciar la matriz con valores aleatorios con la tecla R
7. Reiniciar la matriz con valores neutros con la tecla B (ceros o equivalente)

El cerebro de Brian

Este autómata celular es parecido al juego de la vida de Conway, pero modificando las reglas y la cantidad de estados. Fue creado por Brian Silverman. Al igual que el juego de la vida de Conway, cada célula tiene 8 vecinos. Las células tienen 3 estados: viva, muriendo o muerta. Se utilizan las siguientes reglas:

1. Una célula muerta que tiene exactamente 2 vecinos vivos se convierte en célula viva.
2. Una célula viva, sin importar la cantidad de vecinos vivos, se convierte en célula que está muriendo.
3. Una célula que está muriendo, sin importar la cantidad de vecinos vivos, se convierte en célula muerta.

A la hora de representar las células de este autómata deben diferenciarse con colores el estado vivo del estado muriendo.



Debe utilizarse y adaptarse el código del autómata de la vida de Conway que se desarrolló en clase, adaptando las siguientes subrutinas para su implementación.

- generar_aleatoria(filas, columnas)
  Genera una matriz de las dimensiones indicadas con estados aleatorios de los permitidos en el autómata.
- generar_nula(fias, columnas)
  Genera una matriz de las dimensiones indicadas con todos los estados en cero.
- obtener_vecinos(M, fila, columna)
  Recibe la matriz del autómata y una posición. Retorna los ocho vecinos de la célula en esa posición. Debe interpretar la matriz como un toroide.
- transicion_celula(estado, vecinos)
  Recibe el estado de una célula y la lista de estados de sus vecinos. Retorna el estado que tendrá la célula en el siguiente paso del autómata.
- transicion(M)
  Aplica la función de transición a cada célula de la matriz M y genera una nueva matriz con el resultado.
- cambiar_estado(M, fila, columna)
  Recibe la matriz del autómata y cambia el estado de la célula en la posición indicada al siguiente estado permitido en autómata. Si se encuentra en el último estado, debe cambiarla al primero.

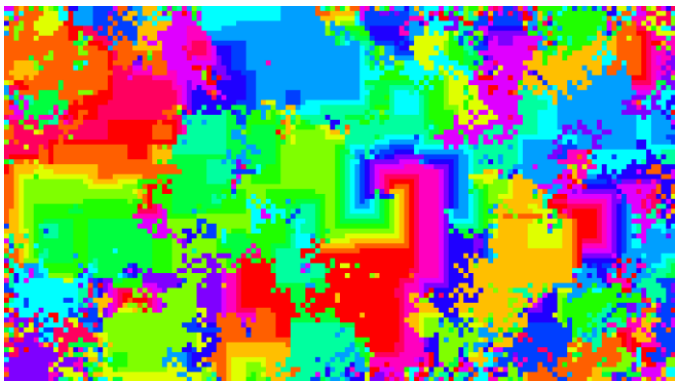
Autómata celular cíclico

En este autómata utilizaremos 16 estados para cada célula (de 0 a 15). Cada estado va a estar asociado a un color. Los valores contiguos deben tener colores similares, para dar la sensación de cambio gradual. El siguiente valor después de 15 es 0. Por esto lleva el nombre de cíclico. Pueden utilizarse los siguientes colores, pero hay libertad de escoger el gradiente que se desee.

Se utiliza la siguiente regla:

1. Una célula con estado n toma el valor (n + 1) % 16 si tiene al menos un vecino con ese valor.

La matriz también debe iniciar con estados aleatorios para cada célula.



Debe utilizarse y adaptarse el código del autómata de la vida de Conway que se desarrolló en clase, adaptando las siguientes subrutinas para su implementación:

- generar_aleatoria(filas, columnas)
  Genera una matriz de las dimensiones indicadas con estados aleatorios de los permitidos en el autómata.
- generar_nula(fias, columnas)
  Genera una matriz de las dimensiones indicadas con todos los estados en el primer color.
- obtener_vecinos(M, fila, columna)
  Recibe la matriz del autómata y una posición. Retorna los ocho vecinos de la célula en esa posición. Debe interpretar la matriz como un toroide.
- transicion_celula(estado, vecinos)
  Recibe el estado de una célula y la lista de estados de sus vecinos. Retorna el estado que tendrá la célula en el siguiente paso del autómata.
- transicion(M)
  Aplica la función de transición a cada célula de la matriz M y genera una nueva matriz con el resultado.
- cambiar_estado(M, fila, columna)
  Recibe la matriz del autómata y cambia el estado de la célula en la posición indicada al siguiente estado permitido en autómata. Si se encuentra en el último estado, debe cambiarla al primero.

Hormiga de Langton

Este autómata tiene reglas muy simples, pero genera un comportamiento emergente complejo. Fue inventado por Chris Langton en 1986 y utiliza células con dos estados. En este autómata se maneja una posición dentro de la matriz que representa una hormiga. La hormiga también tiene dirección, puede ir hacia arriba, abajo, izquierda o derecha.

Las reglas giran en torno al estado de la célula donde se encuentra la hormiga y la dirección que tiene:

1. Si se encuentra en una célula blanca, gira a la derecha 90 grados, cambia el color de la célula y avanza una célula.
2. Si se encuentra en una célula negra, gira a la izquierda 90 grados, cambia el color de la célula y avanza una célula.

En este caso la matriz inicia con células blancas y la hormiga ubicada en una posición en el centro de la matriz.

Es importante recalcar que la posición de la hormiga y su orientación son parte del autómata, por lo que si se utiliza la opción de guardar, estos datos deben incluirse en el archivo.

Para este autómata es necesario modelar la posición de la hormiga y su dirección aparte de la matriz del autómata. Para esto debe utilizarse un diccionario que encapsule sus valores.

No es necesario utilizar todas las funciones del autómata de Conway, ya que en cada iteración sólo requiere revisarse la célula en la que se encuentra la hormiga.

Deben implementarse las siguientes funciones para el funcionamiento del autómata. Puede agregar parámetros adicionales a las subrutinas en caso de necesitarlo.

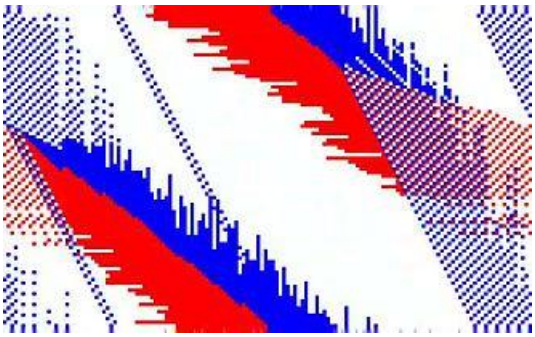
- generar_aleatoria(filas, columnas)
  Genera una matriz de las dimensiones indicadas con estados aleatorios de los permitidos en el autómata.
- generar_nula(fias, columnas)
  Genera una matriz de las dimensiones indicadas con todos los estados en blanco.
- girar_hormiga(direccion)
  Permite actualizar la hormiga para realizar un giro hacia la derecha o la izquierda.
- avanzar_hormiga()
  Actualiza la posición de la hormiga de acuerdo con su posición y su dirección actual.
- siguiente()
  Analiza el color de la celda donde se encuentra la hormiga y realiza la acción correspondiente para actualizar la matriz y la hormiga.

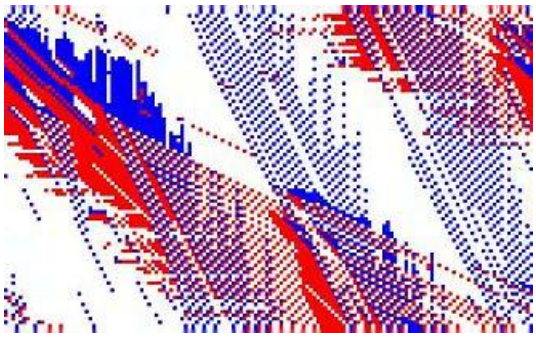
Al guardar y cargar este autómata en disco, debe considerarse el estado de la hormiga.

Modelo de tráfico Biham-Middleton-Levine

Este es un autómata celular con patrones de autoorganización. Una celda puede estar vacía o contener un automóvil. Los automóviles pueden ser de dos tipos: los que se mueven hacia abajo (azul) y los que se mueven hacia la derecha (rojo). Un automóvil azul cambiará su posición en la siguiente generación si la celda ubicada en la parte inferior se encuentra libre, de otro modo permanecerá en la misma posición. De la misma manera, un automóvil rojo cambiará su posición en la siguiente generación si la celda ubicada a la derecha se encuentra libre, si no permanecerá en la misma posición. En este autómata es necesario que los bordes de la matriz estén conectados, es decir, un auto rojo que llega a la última columna aparecerá en la columna 0, y un auto azul que llega a la última fila, aparecerá en la fila 0.

Para que este autómata funcione fluidamente, es necesario garantizar que exista alrededor de un 60% de espacio libre, no ocupado por vehículos. De otra forma el autómata entrará en un estado donde no hay movimiento.





No es necesario utilizar todas las funciones del autómata de Conway, ya que en cada iteración sólo requiere revisarse la célula en la que se encuentra la hormiga.

Deben implementarse las siguientes funciones para el funcionamiento del autómata. Puede agregar parámetros adicionales a las subrutinas en caso de necesitarlo.

- generar_aleatoria(filas, columnas)
  Genera una matriz de las dimensiones indicadas con estados aleatorios de los permitidos en el autómata.
- generar_nula(fias, columnas)
  Genera una matriz de las dimensiones indicadas con todos los estados en vacío.
- avanzar_rojos(M)
  Retorna una nueva matriz del autómata donde se han actualizado las posiciones de los automóviles rojos.
- avanzar_azules(M)
  Retorna una nueva matriz del autómata donde se han actualizado las posiciones de los automóviles azules.

Autómata adicional

Esta parte consiste en investigar sobre algún otro autómata de funcionamiento similar a los anteriores que sea del interés del grupo o en crear un autómata con reglas propias que produzca algún comportamiento interesante.

La documentación del código debe incluir datos sobre el autómata adicional, como por ejemplo la persona que lo creó, cantidad de estados, reglas que sigue y cualquier otra característica importante sobre su funcionamiento.

Menú principal

Como punto de entrada al programa debe crearse una ventana o pantalla principal donde se pueda escoger el tipo de autómata a visualizar, preferiblemente con el uso del mouse o con alguna opción gráfica (no en consola). Una vez elegido, se muestra el funcionamiento de este.

Documentación

Toda subrutina debe llevar como documentación interna comentarios con lo siguiente:

- Descripción de la función
- Entradas
- Salidas
- Restricciones

En cuanto a la documentación externa, debe entregarse un manual de usuario en formato PDF que contenga los siguientes puntos:

1. Explicación del propósito del programa.
2. Instrucciones para la instalación del programa y cualquier requisito que tenga. Con imágenes de ejemplo.
3. Instrucciones de utilización de todas las características del programa. Con imágenes de ejemplo.

Forma de trabajo

El proyecto será desarrollado en equipos de dos personas. No se aceptarán trabajos individuales, salvo previa autorización del profesor y con alguna justificación especial que lo amerite.

Entrega

Debe entregarse un archivo **archivo ZIP** que contenga dos cosas:

1. Los archivos Python con los módulos necesarios para el proyecto.
2. Un archivo PDF con el manual de usuario

Evaluación

La tarea tiene un valor de 20% de la nota final, en el rubro de Proyectos Programados.

Desglose de la evaluación de la tarea programada:

Documentación:	20%
Programación:	80%

Recomendaciones adicionales

Pruebe cada función individualmente. No implemente todo el programa sin verificar el funcionamiento por separado de cada una de sus funciones. Esto dirige a errores que son más difíciles de encontrar.

Recuerde que el trabajo es en equipos, es indispensable la comunicación y la coordinación entre los miembros del subgrupo.

Plagios no serán tolerados en ninguna circunstancia. Cualquier intento de fraude será evaluado con una nota de cero y se llevará a cabo el proceso estipulado en el Reglamento de Enseñanza-Aprendizaje del Instituto Tecnológico de Costa Rica. Siempre escriba su propio código.

Referencias

Cellular automaton. (2016, May 19). In *Wikipedia, The Free Encyclopedia*. Retrieved 00:19, May 27, 2016, from https://en.wikipedia.org/w/index.php?title=Cellular_automaton&oldid=720992591

Brian's Brain. (2015, September 2). In *Wikipedia, The Free Encyclopedia*. Retrieved 00:19, May 27, 2016, from https://en.wikipedia.org/w/index.php?title=Brian%27s_Brain&oldid=679148776

Cyclic cellular automaton. (2014, December 21). In *Wikipedia, The Free Encyclopedia*. Retrieved 00:19, May 27, 2016, from https://en.wikipedia.org/w/index.php?title=Cyclic_cellular_automaton&oldid=638978681

Langton's ant. (2015, December 16). In *Wikipedia, The Free Encyclopedia*. Retrieved 00:20, May 27, 2016, from https://en.wikipedia.org/w/index.php?title=Langton%27s_ant&oldid=695460564

Biham–Middleton–Levine traffic model. (2017, April 7). In Wikipedia, The Free Encyclopedia. Retrieved 18:26, May 21, 2017, from https://en.wikipedia.org/w/index.php?title=Biham%E2%80%93Middleton%E2%80%93Levine_traffic_model&oldid=774340444