República Bolivariana de Venezuela
Ministerios del Poder Popular para la Educación Universitaria
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Vista de Cazador

Proyecto 1 Algoritmo y Estructuras de Datos

Auxiliares Docentes:

Bryan Silva

Erimar Reis

Estudiante:

Elsa Pestana

Caracas, 10 de diciembre de 2025

INFORME

El Equipo Delta reporta que ha culminado el sistema Techno-Arcano. Tras extensas pruebas de simulación, el sistema de visión está operativo y listo para su integración en los autómatas de batalla X78A-Cazador. Se ha logrado implementar la detección correcta de los radios de ataque, además de priorizar los blancos para maximizar el daño sin exceder la capacidad de munición disponible. De esta manera, se logró asegurar la preservación de miles de vidas en las posibles futuras guerras.

1- Análisis del problema

El primer desafío técnico abordado consistió en la interpretación de los datos de entrada (*objectives*). Para el programa, cada imagen no es simplemente una visualización gráfica, sino una matriz bidimensional donde cada celda (píxel) almacena un valor numérico entero que representa la intensidad de color en una escala de grises.

Los archivos de entrada siguen el estándar PGM (Portable Graymap) en el formato ASCII, identificado en la cabecera del archivo por la firma “P2”. La estructura de estos archivos se organiza de la siguiente manera:

- Formato: Identificador "P2".
- Dimensiones: Valores enteros que definen el ancho y el alto de la matriz.
- Valor Máximo: Un entero que establece el tope de intensidad de color
- Cuerpo de Datos: La secuencia de valores numéricos que conforman la imagen píxel a píxel.

Dentro de la matriz de datos se encuentran diversos números que representan los radios de ataque o “círculos” Estos pueden variar enormemente de tamaño: desde un número rodeado de cero hasta círculos masivos que abarquen la totalidad de la imagen (256 x 256). En teoría, el programa define estos “círculos” como un conjunto de píxeles de un mismo color que poseen otros píxeles adyacentes de igual color.

Para cada uno de estos círculos se calculan unas características geométricas básicas: el radio, el diámetro y la circunferencia. Un punto clave es que cada círculo posee un valor propio de “π” (Pi Individual), el cual se obtiene dividiendo su circunferencia entre su diámetro.

Una vez obtenidas las propiedades individuales de los círculos, el sistema calcula el “Pi normalizado”. Este valor se basa en la suma de todos los PI individuales dividida entre la cantidad total de círculos presentes en la imagen. Luego se calcula el área normalizada de cada círculo multiplicando este Pi Normalizado por el radio al cuadrado.

Por otro lado, para la neutralización de los "radios de ataque", el sistema opera bajo una restricción operativa crítica: una capacidad limitada de disparos (T). El

objetivo principal es maximizar el daño total generado. Para lograr esto, el algoritmo calcula la sumatoria de las áreas normalizadas de los círculos detectados y evalúa diversas combinaciones posibles (subconjuntos) hasta encontrar aquella que produzca el mayor impacto posible sin exceder el límite T.

Asimismo, se establecieron criterios de desempate para situaciones donde múltiples combinaciones generan el mismo daño máximo:

- Eficiencia: Se selecciona la lista que contenga la menor cantidad de radios de ataque.
- Prioridad por Intensidad: En caso de que la cantidad también coincida, se prioriza la combinación que incluya el objetivo más oscuro.

2- DISEÑO

2.1 Gestión de Memoria

Uno de los desafíos técnicos más críticos fue garantizar la estabilidad del sistema al procesar imágenes de alta densidad (256 x 256 píxeles). El manejo inicial de estos volúmenes de datos mediante arreglos estáticos conllevaba un riesgo latente de desbordamiento de la pila de memoria (*Stack Overflow*), lo que comprometía la operatividad del programa.

Para solucionar esta limitación, se optó por la implementación de arreglos dinámicos. En lugar de definir estructuras de datos rígidas, componentes como "**Región**" fueron rediseñados para utilizar punteros y asignación de memoria en tiempo de ejecución (*Heap*). Esta arquitectura permite que el sistema procese regiones masivas de hasta 65,536 píxeles sin fallos, garantizando así la robustez operativa requerida para los autómatas X78A-Cazador.

2.2 Algoritmo de Detección (“Estanque”)

Para la identificación y delimitación de los “radios de ataque”, se diseñó la función denominada “**estanque**”. Está ejecuta un proceso de expansión por ondas. Su nombre radica en la similitud operativa con la acción física de arrojar una piedra a un cuerpo de agua; al igual que el impacto genera ondas que se desplazan por la superficie, el algoritmo propaga la exploración desde un píxel inicial (elemento análogo a “la piedra”) hacia sus vecinos inmediatos, expandiéndose progresivamente hasta cubrir la totalidad de la figura o chocar con un límite (un píxel de valor 0 o el borde de la imagen).

Profundizando en su arquitectura interna, la función “**estanque**”, implementa el siguiente procedimiento técnico:

Inicialización de parámetros

```
void estanque(int x, int y, Imagen &imagen, Region &region, bool visitado[256][256] ){
```

La ejecución de la función depende de cuatro parámetros fundamentales que definen la expansión:

1) **Coordenadas (x, y):** Se reciben dos valores enteros que establecen el punto de inicio de la expansión (el equivalente a "la piedra" en la analogía). Estas coordenadas determinan el origen desde el cual se propaga la onda de búsqueda.
2) **El Entorno (`Imagen &imagen`):** Se pasa la estructura completa de la imagen. Esto permite al algoritmo consultar los límites de la matriz ancho/alto y la intensidad de color de los píxeles
3) **Contenedor Final (`Region &region`):** Es la estructura donde se almacenarán, paso a paso, los píxeles que conforman el "radio de ataque" detectado.
4) **Control de Búsqueda (`bool visitado[256][256]`):** Su función es marcar cada píxel procesado para asegurar que el algoritmo nunca evalúe el mismo punto dos veces.

Gestión de Olas de Expansión

```
int maxP = imagen.ancho * imagen.alto;

int* ondaActualx = new int[maxP]; // onda de pixeles actuales en x
int* ondaActualy = new int[maxP]; // onda de pixeles actuales en y
int* ondaSiguientex = new int[maxP]; // onda de pixeles siguientes en x
int* ondaSiguientey = new int[maxP]; // onda de pixeles siguientes en y

int tamOndaActual = 0;
int tamOndaSiguiente = 0;

int colorObjetivo = imagen.pixeles[y][x]; // color del primer pixel que no sea 0

ondaActualx[tamOndaActual] = x; // el primer pixel se agrega a la onda actual (x)

ondaActualy[tamOndaActual] = y; // el primer pixel se agrega a la onda actual (y)

tamOndaActual++;// aumenta el tamaño de la onda actual

visitado[y][x] = true;// se marca como visitado
```

Debido a que una sola “ola” podría abarcar teóricamente la totalidad de la imagen (hasta 65,536 píxeles) el sistema utiliza memoria dinámica, la cual es flexible y es capaz de adaptarse. Para organizar el movimiento de la onda, se configuran dos grupos de arreglos:

- **Arreglos de la Ola Actual (`ondaActualx, ondaActualy`):** Aquí se almacenan las coordenadas de los píxeles que se están procesando en el momento presente.
- **Arreglos de la Ola Siguiente (`ondaSiguientex, ondaSiguientey`):** Aquí se irán anotando las coordenadas de los nuevos píxeles vecinos que se descubran durante el proceso.

Además se declara “**tamOndaActual**” y “**tamOndaSiguiente**” como tamaño de las respectivas ondas y por último se declara “**colorObjetivo**” que es el color del primer píxel encontrado que no sea 0. Finalmente el sistema agrega a las ondas actuales los valores de “**x**” y “**y**” respectivamente y marca ese primer píxel como visitado en el mapa de control.

Proceso de expansión

Una vez preparados los arreglos de la "ola actual", el sistema entra en un ciclo donde se ejecuta tres acciones fundamentales:

1) **Primero**, se toma cada píxel del arreglo actual y se guarda su información (coordenadas y color) dentro de la estructura "**Region**". Esto permite construir la región completa de "radio de ataque".
2) **Segundo**, para cada píxel, el algoritmo calcula las coordenadas de sus 4 vecinos directos (arriba, abajo, izquierda, derecha). Luego para que ese vecino se agregue a la "siguiente ola" se pregunta si el vecino está dentro de los límites de la imagen, si este vecino ya fue visitado antes, y por último si es del mismo color del color objetivo. Si el vecino cumple todas estas condiciones, se agrega a la onda siguiente y se marca como visitado.
3) **Tercero** y último paso, al terminar de revisar todos los píxeles del arreglo actual, el algoritmo realiza un traspaso de datos de toda la información que está en el arreglo de la onda actual, al arreglo de la onda siguiente. De esta forma el ciclo vuelve a empezar hasta que ya no queden más píxeles conectados por encontrar.

2.3 Cálculo de Propiedades

Una vez delimitada la región de píxeles, el sistema procede a realizar una serie de cálculos para cada región.

1. **Detectar el centro:** El centro de cada región se calcula obteniendo la media aritmética de las coordenadas "x y "y" de todos los píxeles que conforman la región.
2. **Cálculo del Radio:** El radio se define como la distancia máxima entre el centro y el píxel más alejado del borde del círculo. Para círculos de un solo píxel se creó una condición donde el radio sea 1, para así evitar áreas nulas.
3. **Pi (π)**: En vez de utilizar la constante matemática estándar (3.1416..), el sistema calcula un Pi Individual para cada círculo cuya fórmula es Circunferencia / Diámetro (se utiliza la circunferencia y diámetro de cada círculo) y posteriormente se calcula el Pi normalizado, el cual se obtiene de promediar todos los Pi individuales de cada círculo presentes en la imagen.
4. **Área Normalizada:** Finalmente, se calcula el área normalizada de cada círculo mediante la fórmula Área Normalizada = Pi Normalizado por el radio al cuadrado.

2.4 Backtracking

La fase final del sistema aborda el problema de seleccionar qué objetivos atacar para maximizar el daño sin exceder la capacidad de tiros (T). Para garantizar la mejor solución, se implementó un algoritmo de Backtracking definido en la función "**mochilero_circulos**" el cual sigue el siguiente procedimiento:

Inicialización y parámetros

```
bool conjuntoWin_ciruclos[32];
int bestSuma_circulos = -1; //
int bestCantidad_circulos = -1;
int MasOscuro_circulos = -1; //
```

```
void mochilero_circulos(Circulo circulos[], int totalCirculos, int T, int index, int
    sumaActual, int cantidadActual, int IOactual, bool conjuntoActual[]){
```

Para almacenar la mejor solución Primero se declararon variables globales que actúan como registros “**bestSuma_circulos**” Almacena el máximo daño conseguido hasta el momento, “**bestCantidad_circulos**”, “**MasOscuro_circulos**”, guardan los valores necesarios para el desempate, y por último “**cojuntoWin_circulos**” Es el arreglo que guarda la combinación ganadora.

Por otro lado, la función “**mochilero_circulos**” recibe una serie de parámetros los cuales transportan la información de la simulación actual. El arreglo de Círculos, el total de círculos y el total de disparos (T), son datos fijos que no cambian, “**index**” indica qué círculo se está procesando en la llamada actual. Las variables “**sumaActual**” y “**cantidadActual**” llevan la cuenta del daño generado y los disparos gastados en la llamada actual, mientras que “**IOActual**” guarda el círculo más oscuro incluido en la combinación actual, y por último “**conjuntoActual**” que es un arreglo booleano que actúa como almacén, marcando los círculos incluidos o excluidos en esa llamada recursiva.

Mejor Solución Encontrada

La primera parte del algoritmo se ejecuta cuando la recursión llega a evaluar todos los círculos existentes. En este punto se tiene una combinación completa de círculos y procedemos a compararla con la mejor solución registrada hasta el momento. Para saber si esta combinación es la mejor tiene que pasar 3 niveles de filtros.

A. **Nivel 1:** Si la “**sumaActual**” es estrictamente mayor a la “**bestSuma**” registrada, esta nueva combinación se convierte en la ganadora.
B. **Nivel 2:** Si hay un empate en el daño, el sistema revisa la cantidad de disparos. Si “**cantidadActual**” es menor que la “**bestCantidad**”, la nueva combinación gana por ser la que ahorra munición (menor cantidad de “radios de ataque”).
C. **Nivel 3:** Si también hay empate en la cantidad, revisamos el color. Si “**IOActual**” (círculo más oscuro) de la nueva combinación es menor que el “**MasOscuro**” registrado esta nueva combinación gana por prioridad de objetivo.

Si la combinación actual pasa cualquiera de estos filtros, el sistema actualiza todas las variables de registros (con los valores de la combinación que salió ganadora) y hace una copia del “**conjuntoActual**” hacia “**cojuntoWin_circulos**” guardando así la nueva combinación ganadora.

Backtracking

Si no se ha llegado al caso base (“**index == totalCirculos || sumaActual > T**”), el algoritmo debe decidir qué hacer con el circulo actual: ¿Se incluye o no en el ataque? Para tomar esta decisión se exploran dos ramas:

1) **Inclusión**

   Primero, el algoritmo verifica si el círculo actual cabe dentro de la capacidad restante (“**sumaActual + circulos[index].areaNormal <= T**”) De ser válido, se marca el círculo como incluido en el arreglo “**conjuntoActual**”, luego se decide que variable sera indicador de oscuridad en la siguiente llamada recursiva para eso se declaro una

variable “**newIO**” entonces se pregunta si el círculo actual es el primer elemento que se agrega a la combinación entonces su índice se registra como el nuevo indicador de oscuridad de caso contrario se conserva el indicador “**IOActual**”. Acto seguido se ejecuta la llamada recursiva sumando el índice (“**index+1**”) la suma actual (“**sumaActual + circulos[index].areaNormal**”) y la cantidad actual (“**cantidadActual+1**”)

2) **Exclusión**

Independientemente de que si el círculo actual entre o no entra en la “mochila”, el algoritmo debe explorar la posibilidad de excluirlo de la solución. Lo primero que realiza el algoritmo es desmarcarlo del “**conjuntoActual**” seguido a ellos realiza la llamada recursiva sin modificar ningun parametro a excepción del indice (“**index+1**”)

Conclusión

En síntesis, a lo largo de este informe se logra detallar el desarrollo exitoso del sistema de visión para los autómatas de batalla X78A-Cazador. El objetivo principal fue crear un software capaz de interpretar imágenes en formato PGM para detectar "radios de ataque" (círculos) y seleccionar estratégicamente cuáles neutralizar para maximizar el daño con una munición limitada.

Siguiendo con el orden de ideas del párrafo anterior, el sistema es operativo y logra priorizar blancos eficientemente, cumpliendo con el objetivo de maximizar el impacto estratégico y preservar vidas en futuros conflictos.

De esta forma culmina el reporte técnico del equipo Delta. La misión de optimización ha sido completada satisfactoriamente.

“Vista de Cazador” El presente informe fue hallado transcrito en un pergamino antiguo dentro de las ruinas de la Oficina Sub-regional de Therion. A su lado, reposan intactos los planos originales de los autómatas X78A-Cazador, testigos mudos de la tecnología que definió una era