Victoria González Chacín – 202320852

Zair Montoya – 202321067

Andrea Dávila –

Construcción del grafo:

Para la construcción del grafo tuvimos en cuenta las siguientes consideraciones en el juego de Metroid:

1. Va a existir un orden en donde la plataforma 0, o la plataforma en donde Samus se encuentra al inicio del juego, será la inicial, y la plataforma n-1 siempre será nuestra plataforma final o destino, o en donde se encontrará Raven Beak.
2. Samus puede saltar en cualquier dirección (para atrás o adelante) de una plataforma a la siguiente.
3. En ciertas plataformas hay poderes que le permite a Samus dar un salto de k plataformas, también en cualquier dirección.
4. Samus tiene la capacidad de teletransportarse en cualquier momento (por lo tanto, en cualquier plataforma), a la plataforma que desee (que consume el número de plataformas por las cuales se mueva de energía).
5. En ciertas plataformas hay robots asesinos, que no pueden ser alcanzados.

De esta forma sabemos que los nodos serán las n plataformas existentes, y las aristas serán las conexiones que se dan cuando Samus puede saltar a las plataformas subsiguientes, cuando puede usar poderes para saltar a la plataforma actual + k saltos en cualquier dirección, y cuando quiera teletransportarse de su posición actual a cualquier otra plataforma. El momento definitivo en donde no van a haber conexiones entre plataformas es cuando hay robots asesinos en una de ellas, ya que esto haría automáticamente que Samus pierda el juego.

De esta forma comenzamos con el proceso de construcción para los movimientos de caminar (C+ y C-):

* Si dentro de las plataformas, la plataforma i (que estamos viendo) esta dentro de las plataformas con robots, la ignoramos. Ya que Samus puede caminar para adelante y hacia atrás hacemos dos verificaciones; si la siguiente plataforma (i+1) no se sale de los rangos y tampoco tiene robots entonces podemos crear una arista entre ambas plataformas, y lo mismo para la plataforma anterior (i-1).

Seguimos con los momentos en donde Samus pueda requerir teletransportarse (Tx):

* Recorremos todas las plataformas a las cuales Samus podría teletransportarse (j) teniendo en cuenta que no puede ser la misma en la que estamos (i) ni tiene robots entonces podemos crear una arista entre ambas plataformas.

Y finalizamos con los poderes de saltos (S- y S+):

* Empezamos verificando nuevamente que la plataforma actual (i) no tenga robots asesinos. Proseguimos revisando que la plataforma a la cual Samus terminara después del salto (i + salto ó i - salto) no se salga del rango y tampoco tenga robots, y si es así entonces se puede crear la correspondiente arista.

Algoritmo seleccionado:

Ya con el grafo construido definimos que el problema que se busca solucionar es encontrar el camino más corto (menor numero de movimientos) desde la plataforma 0 hasta la plataforma n-1, siempre que tenga suficiente energía para hacerlo.

Teniendo en cuenta que tenemos aristas de igual costo y que solo nos interesa el minimizar el número de movimientos (no funcionaría DFS, ya que no garantiza el camino más corto), no el peso de las acciones (por lo que no nos serviría Dijsktra), podemos usar el algoritmo de BFS para explorar primero los caminos mas cortos, no atascarnos en caminos más largos innecesarios, evitar ciclos y caminos inválidos (considerando que nuestro grafo puede tener muchos caminos posibles).

Se sigue el algoritmo estándar de bfs, iniciando con un set de plataformas visitadas y una cola con una tripleta que contiene la plataforma de inicio (0), la energía que tiene Samus y una lista vacía para ir llenándola con los pasos de cada recorrido. Mientras haya elementos por explorar tomamos el siguiente nodo a explorar (plataforma actual, energía restante y el camino recorrido hasta ese punto). Si llegamos a la ultima plataforma, o en la que esta Raven Beak, devolvemos el camino. Por otro lado, si ya estuvimos en esta plataforma con esa energía exacta, ignoramos esta rama, y marcamos el estado como visitado. Recorremos todos los movimientos o acciones posibles desde la plataforma actual. En el caso de que la acción que estamos viendo es teletransportarse entonces le restamos el numero de pasos que se hacen a la cantidad de energía gastada, y si no hay suficiente energía descartamos esa acción. Si la plataforma vecina aun no ha sido visitada, agregamos ese nuevo estado a la cola, con la nueva energía y el camino actualizado. Si terminamos de explorar todo y no llegamos a la plataforma final, no hay camino valido, y devolvemos “NO SE PUEDE”.

**Complejidad:**

Este código implementa una solución basada en búsqueda en anchura (BFS) para encontrar una secuencia de acciones que permita alcanzar la última plataforma en un entorno representado como un grafo. Cada nodo del grafo representa una plataforma, y las aristas corresponden a las distintas acciones posibles desde una plataforma hacia otra: caminar, teletransportarse o usar un poder especial. Las decisiones están condicionadas por obstáculos (robots que bloquean plataformas) y por una cantidad limitada de energía que se consume al realizar ciertas acciones.

La complejidad temporal del algoritmo está determinada principalmente por dos funciones: construccion\_grafo y BFS. En la función construccion\_grafo, se itera sobre cada plataforma para construir las conexiones válidas. Esta función contiene un doble bucle anidado en el que por cada plataforma i, se evalúan todos los posibles destinos j con los que podría conectarse mediante un salto "T". En el peor caso, si no hay robots que bloqueen caminos y todos los saltos son válidos, se generan hasta n² conexiones, resultando en una complejidad temporal de O(n²) para la construcción del grafo. También se evalúan los poderes especiales (acciones "S+” y “S-”), que en el peor caso agregan una complejidad adicional de O(n), aunque esta no cambia el orden final.

Por su parte, la función BFS explora el grafo generado utilizando una cola que almacena tuplas de estado con la forma (plataforma\_actual, energia\_restante, camino\_corrido). La clave aquí es que el espacio de estados no está dado solo por la plataforma actual, sino también por la energía restante, lo que implica que podrían existir hasta n \* energía combinaciones únicas que deben visitarse. Por ello, la complejidad temporal del BFS en el peor de los casos puede alcanzar O(n \* energía \* d), donde d es el grado de cada nodo (es decir, el número de acciones posibles desde una plataforma). En un grafo completamente denso, donde cada plataforma puede conectarse con casi todas las demás, este grado puede ser O(n), llevando a una complejidad temporal de O(n² \* energía) para esta fase.

En cuanto a la complejidad espacial, se deben considerar varias estructuras. El grafo construido puede ocupar hasta O(n²) espacio si todas las conexiones posibles se almacenan. La cola usada por BFS podría llegar a contener hasta O(n \* energía) elementos en un escenario donde se exploren muchas combinaciones de plataforma y energía antes de alcanzar la meta. El conjunto visitado también podría almacenar hasta O(n \* energía) estados únicos. Además, cada entrada en la cola de BFS lleva consigo una copia del camino recorrido hasta ese punto, y aunque en muchos casos los caminos son cortos, en el peor caso podrían ocupar espacio O(n) cada uno, lo que puede llevar a un uso total de memoria de hasta O(n²) si no se optimiza adecuadamente esta estructura.

En resumen, este algoritmo presenta una complejidad temporal de O(n² + n² \* energía), resultado de la combinación de la construcción completa del grafo y la exploración exhaustiva de estados en la búsqueda. En cuanto a la complejidad espacial, se puede estimar como O(n² + n \* energía) por el almacenamiento del grafo, la cola de BFS, el conjunto de visitados y los caminos parciales. Este enfoque es adecuado para valores moderados de n y energía, pero podría ser mejorado para casos más exigentes evitando la construcción completa del grafo y generando dinámicamente las transiciones durante la exploración.