PROYECTO 1 IA- LABERINTO MONONOKE

CRISTHIAN BOTERO RODRIGUEZ 1860054

JUAN CAMILO OBANDO RENDON 1859971

Para el algoritmo de búsqueda por profundidad iterativa y para el de búsqueda por costo usamos un array bidimensional 4x5 “tablero” para guardar la configuración del laberinto, donde estaba cada uno de los personajes que hacen parte del laberinto, usando un 1 para identificar a Mononoke, un 2 para identificar a los kodamas, un 3 para el venado Yakult, un 4 para el espíritu del bosque (meta) y un 5 para los obstáculos que decidimos representar gráficamente como árboles. Usamos también un array bidimensional 4x5 “posiciones” para asociar las posiciones del tablero con las coordenadas de cada casilla de laberinto y así poder poner las imágenes de los diferentes personajes en las posiciones correctas. Además definimos una clase llamada Árbol que tiene como constructor el elemento que se agrega, por debajo trabaja como un array y tiene los métodos buscarsubarbol que recibe un árbol y un elemento el cual retorna el subárbol y None en caso de no encontrarlo, el método agregarelemento, que recibe el árbol, el elemento a agregar y el padre de ese elemento, esta función hace uso de buscarsubarbol para decidir a qué elemento le agrega ese nuevo hijo, la función buscar padre que nos retorna el padre de un elemento y la función ancestros que me llena un array con el camino de un nodo hasta la raíz, yéndose por los padres.

La parte gráfica del laberinto la desarrollamos con la librería turtle, definiendo una screen con el fondo base del laberinto y agregando los personajes (Mononoke, Yakult, kodamas, árboles y espíritu del bosque) como turtles, para poder definir las posiciones que se configuran en el tablero.

La función “costo” y la función “iterativa” nos ejecutan los algoritmos deseados que son ejecutados al presionar un botón de la ventana creada en la función “configurarjuego”.

Para dibujar estas tortugas están las funciones dibujarMononoke, dibujarKodama, dibujarYakult, dibujarEspiritu, dibujarArbol las cuales nos leen el array tablero y con el array posiciones nos ubica las tortugas en sus posiciones iniciales. Dibujar Mononoke nos recibe un 1 o un 2 dependiendo del botón seleccionado para iniciar la ejecución de costos o profundidad iterativa

La función mover nos recibe una tortuga (Mononoke) y recorre de manera inversa el array que contiene el camino del nodo meta hasta la raiz y a su vez con la función goto de la librería turtle mover la Mononoke

Implementación técnica de búsqueda por costo:

Inicialmente creamos un árbol que contiene un “X” & “Y” iniciales con la posición de la Mononoke, un límite para diferenciar los nodos con igual posición y un 0 que sirve de bandera para saber si se tiene una montura puesta.

Creamos una cola de prioridad que guardará el costo, el límite que tendría el árbol en esa posición, las posibles posiciones y la flag que nos dice si está en una montura o no.

Apenas se llama la función, lo primero que hace es preguntar si la posicion actual es una meta, en caso de que lo sea, se agrega la posición de la meta al array recorrido, luego se llama a la función ancestros que me hace el camino del nodo meta hasta la raiz y luego llama a la función que mueve la Mononoke, posteriormente salta un mensaje con el aviso de que encontró la meta y el costo total de llegar.

Si la posicion actual tiene un venado, la flag pasa a ser 1 para identificar posteriormente los costos que tendrá pasar por los kodamas.

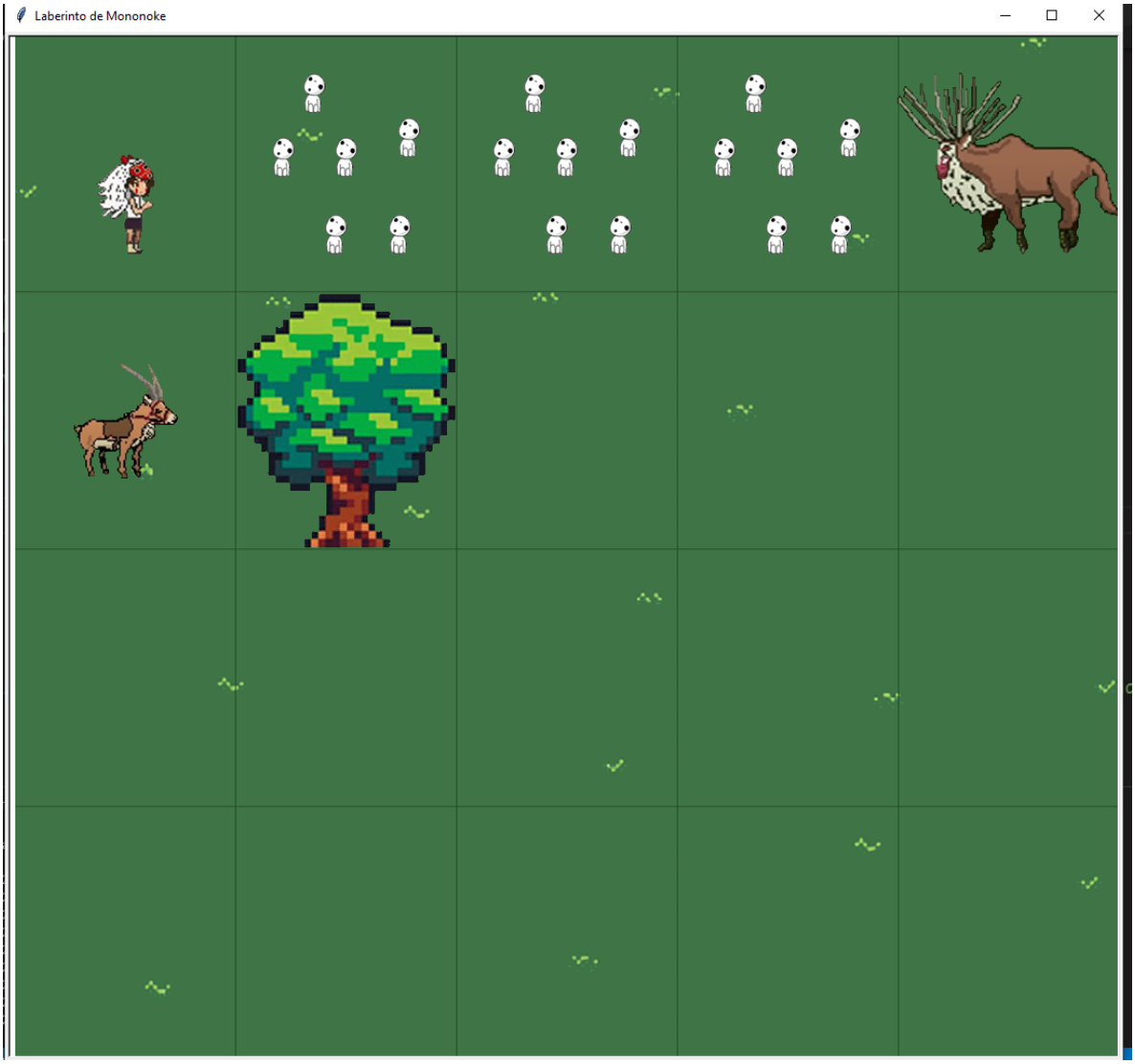
El orden de prioridad de los movimientos es derecha, abajo, izquierda y arriba, todos funcionan igual pero cambiando los X y Y según sea el caso. Primero validamos si la posicion no se sale de la matriz, de cumplirse la condición entonces preguntamos si en esa posicion del tablero hay un árbol, si lo hay se descarta ese movimiento y sigue con el siguiente, si no hay un árbol entonces preguntamos si hay algún kodama y si Mononoke está en su montura o no para calcular el costo de pasar por esa casilla, dependiendo de si se cumplen las condiciones o no, se agrega a la cola de prioridad el costo, el nivel del árbol en la siguiente posición (limite), la posicion en X y Y del posible movimiento y la flag(venado) también guardamos en el árbol el movimiento posible, la flag y el límite de ese posible movimiento que tendrá como padre la configuración actual donde está Mononoke.

Luego sacamos la mejor opción de la cola de prioridad, y hacemos la recursión ahora con esos valores en X y Y, el costo, la flag, el límite, y el árbol.

Implementación técnica de búsqueda por profundidad iterativa:
Inicialmente creamos un árbol que contiene un “X” & “Y” iniciales con la posición de la Mononoke, un límite para diferenciar los nodos con igual posición.
Apenas se llama la función, lo primero que hace es preguntar si la posición actual es una meta, en caso de que lo sea, se agrega la posición de la meta al array recorrido, luego se llama a la función ancestros que me hace el camino del nodo meta hasta la raíz y luego llama a la función que mueve la Mononoke, posteriormente salta un mensaje con el aviso de que encontró la meta y el nivel del árbol en que lo encontró, además cerramos el programa para que no siga haciendo más recursiones.
En caso de que no sea meta preguntamos si el iterador alcanzó al límite para saber si debemos crear más hijos de ese árbol
El orden de prioridad de los movimientos es derecha, abajo, izquierda y arriba, todos funcionan igual, pero cambiando los X y Y según sea el caso. Primero validamos si la posición no se sale de la matriz, de cumplirse la condición entonces preguntamos si en esa posición del tablero hay un árbol, si lo hay se descarta ese movimiento y sigue con el siguiente, si no hay un árbol es un movimiento posible, se aumenta el contador en uno para saber ir teniendo en cuenta todos los nodos que se van explorando, se agrega al árbol y se sigue explorando este movimiento, pero restándole uno al límite, hasta que el límite sea uno y sepa que no debe hacer más llamados recursivo. En cada ejecución si no es un movimiento posible se le va restando los nodos que tendría explorar esa solución
Se pregunta si aún no ha encontrado la solución “flag==0” y ya exploró todos los nodos, si esto se cumple amplia el límite en uno, calcula los costos a explorar en el siguiente nivel,

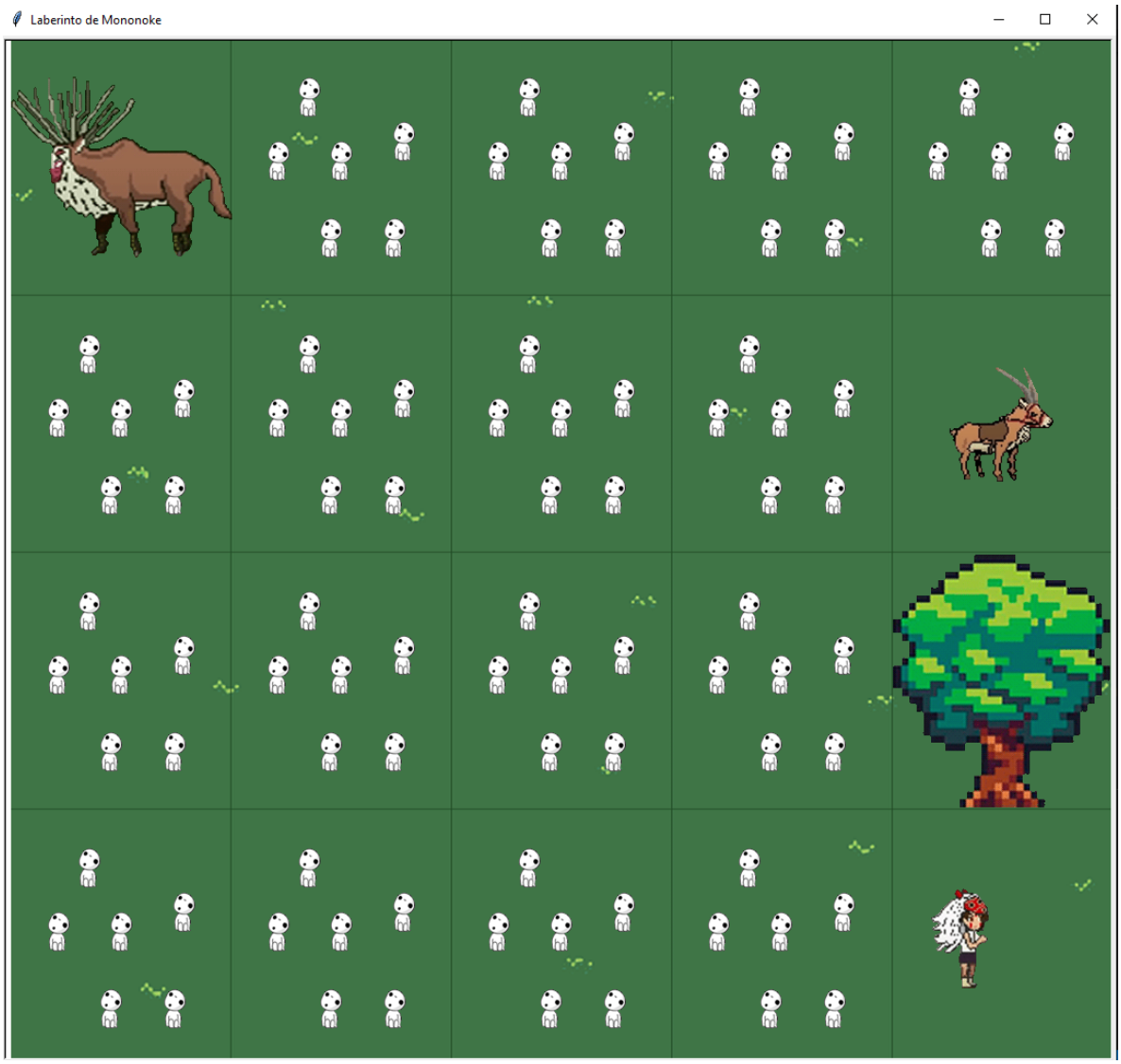
se crea el árbol solo con el primer nodo y se hace un llamado recursivo haciendo la expansión en el siguiente nivel en busca de la solución
Se hace un return None al final para que finalizar cada llamado recursivo donde tenga limite 1, sabiendo que estos no los va a expandir más.
La implementación de esta solución se hizo a partir de que cada vez que el límite se expande se sabe que tiene 4limite+1, ya que se tienen 4 posibles movimientos en cada expansión, pero como hay movimientos que se salen del tablero o que no se van a realizar porque hay un árbol allí entonces se irían descartando esas ramas y restándose para saber que ya exploramos todos, teniendo que hacer la expansión a un nuevo nivel.

Pruebas:



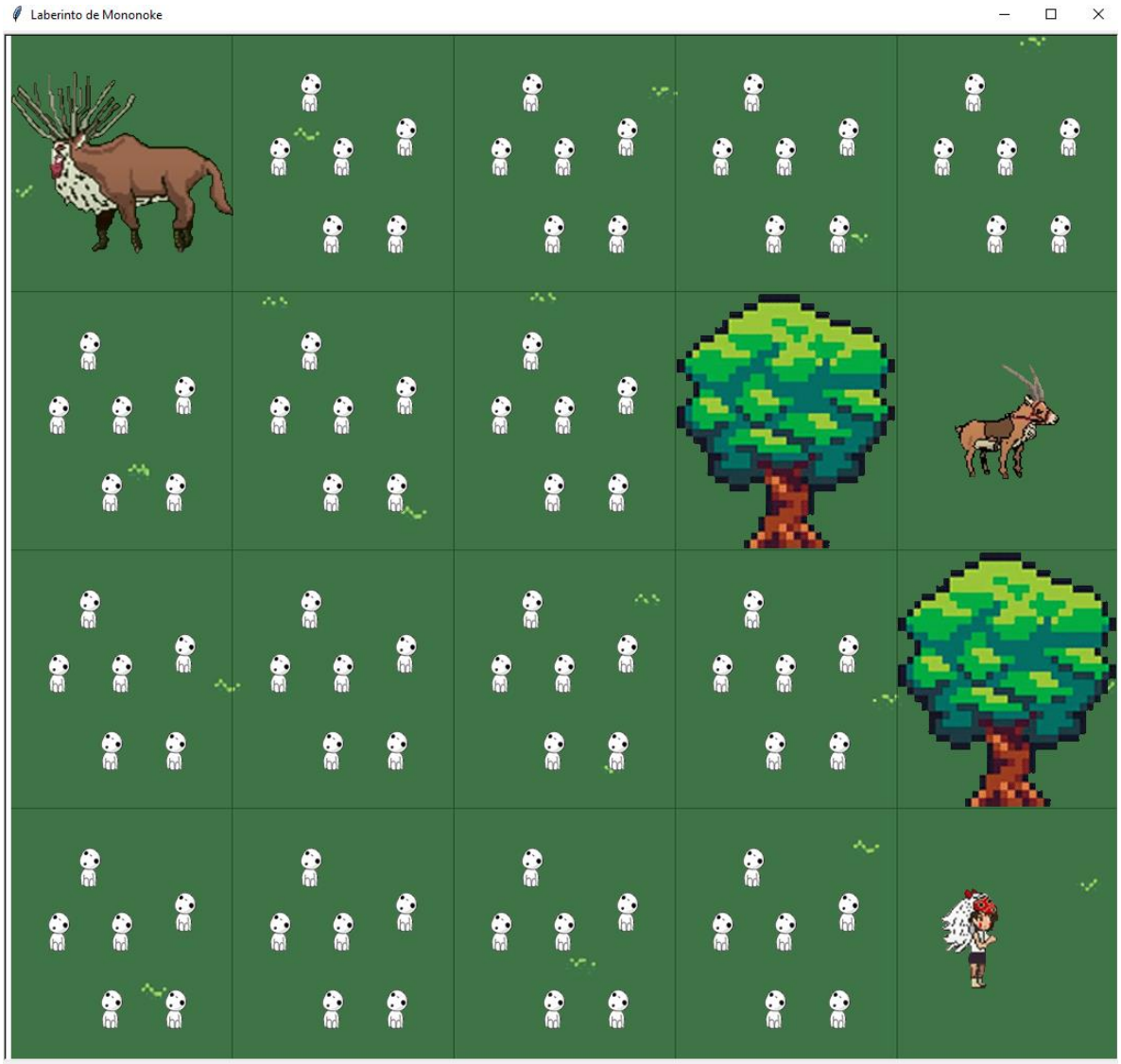
Costo: este escenario hace que Mononoke, baje por el venado vuelva a subir y se vaya por la derecha hasta llegar a la meta teniendo un costo de 6.

Profundidad iterativa: Con este escenario Mononoke simplemente va hacia la derecha porque no le interesan los costos, llegar a la meta le costaría 7.



Costo: Mononoke busca primero el venado ya que le va a restar todos los costos de pasar por los kodamas al final tendrá un costo de 12.

Profundidad iterativa: al estar priorizado derecha -> abajo -> izquierda -> arriba Mononoke va a irse primero por la izquierda y luego subir hasta encontrar al espíritu, lo que le supondrá un costo de 13.



Costo: Mononoke ya no busca el venado ya que ir a cogerlo le supone un costo mayor que ir a la meta directamente, en este caso se comportará de igual manera que profundidad iterativa, ya que los movimientos están priorizados de la misma manera en caso de empate

Profundidad iterativa: al estar priorizado derecha -> abajo -> izquierda -> arriba Mononoke va a irse primero por la izquierda y luego subir hasta encontrar al espíritu, lo que le supondrá un costo de 13.